모래내 미래 Vol.1

차 례

트레이딩컨설팅그룹 이음

2017.05.10

● 자산관리

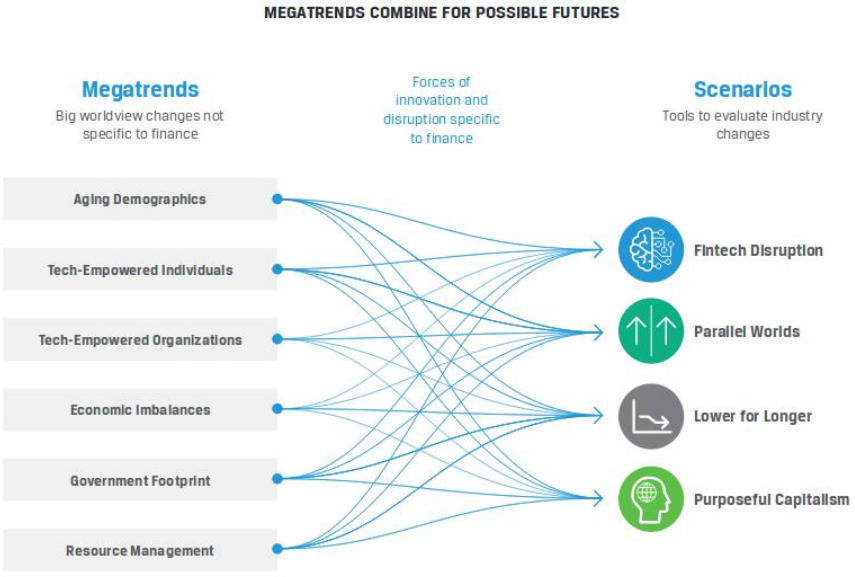
투자전문직의 미래

금융산업이 커다란 변화에 직면하였습니다. 매일경제신문 4월 24일 기사에 따르면 디지탈전략에 따라 대대적인 구조조정을 준비중이라고 합니다.

디지털금융이 본격화 하면서 은행권에 전통적인 산업구조를 지탱하던 칸막이들이 하나 둘씩 무너지는 '와해적 혁신' 바람이 불고 있다. 이러한 현상으로 전통적인 창구 영업에 집중하던 은행들은 디지털전략을 발표함과 동시에 점포를 축소하거나 인력 구조조정 작업에 돌입했다.

디지탈금융은 은행산업뿐 아니라 금융투자산업 및 자산운용산업 나아가 보험에 이르기까지 전방위적으로 영향을 미치고 있습니다. 이런 변화를 상징하는 단어가 '핀테크'입니다. 핀테크가 가져오는 변화중 직무도 한 부분입니다. Robotics Automation Process 가 기본적인 바탕입니다. 그러면 금융투자와 관련한 직무중 투자전문직의 미래는 어떨까요? 미국 CFA 가 2017 년 4 월에 발간한 Future State of the Investment Profession: Pursuing better outcomes—forthe end investor, the industry, and society 입니다.

보고서는 5년에서 10년의 트렌드를 예상하고 이에 대응하는 비즈니스모델을 예상합니다. 보고서가 트레드로 예상하는 것은 Aging Demographics, Tech-Empowered Individuals, Tech-Empowered Organizations, Economic Imbalances, Government Footprint(정부 발자국, 시장이나 기업에 대한 정부의 역할 혹은 개입으로 이해), Resource Management 입니다.



보고서가 그리는 4가지의 시나리오를 보면 Fintech Disruption,Lower for Longer, Parallel Worlds, Purposeful Capitalism 입니다. Parallel Worlds 를 평형우주를 뜻하지만 사회가 다양화하고 개인화면서 세대, 집단, 지역으로 수평적인 관계를 맺는 사회로 이해할 수 있습니다. 이에 따라 개인화, 단순화, 속도를 갖춘 상품이나 서비스가 중요해집니다.

Fintech Disruption

New technologies promote new business models; disruption and creative destruction are endemic; challengers do better than incumbents; major disruptions to the world of work

Major Elements

- Quickening flow of disruptions from technological innovation in digitization and digitalization
- Fintech develops globally with a particularly strong Asia-Pacific element
- Regulatory infrastructure in finance gradually integrates technology-driven models
- Disruptions to investment organization business models; success with technological advancement is critical
- Traditional active management shrinks; some growth in alternatives, smart betas, and outcome-oriented solutions
- Smart machines and systems, data analysis, and inference play a disruptive role in finance's evolution
- Financial services becomes highly personalized and digitalized everywhere
- Robo-advice and its "cyborg" variants become preferred style or tool for delivering investment advice

Parallel Worlds

Different segments—by geography, generation and social group—engage in society differently; a higher baseline for financial services participation with wider dispersion; product preferences for personalization, simplicity and speed

Major Elements

- Better worldwide education, healthcare and telecoms increase societal engagement
- Social media carries potency to bring people together and to divide, legitimately and illegitimately
- Potential for mass disaffection; consequences in anti-globalization, populism, and authoritarian nationalism
- New-style financial institutions enabling personalized, simple, and speedy engagement; trust is also needed
- Big data serves customiza-tion of investment products to specific segments; more reflection of personal values
- Improvement in financial literacy and empowerment produce better financial participation
- The "have-nots" act on their disillusionment with the system
- The trustworthiness of the tech model with tangible products and immediate gratification is tested in investment contexts

Lower for Longer

New normal low interest rates and returns become embedded for the foreseeable future (5–10 years), accentuated by lower levels of global growth and higher levels of political instability

Major Elements

- Limited success with interest rate normalization; natural interest rates stay low
- Growth challenges: indebtedness, adverse demography, excess savings, China/EM, companies hoard cash
- Large gaps in pension coverage with longevity; pension poverty
- Moves to lower-cost, higher-tech investment solutions; premium on innovation; industry consolidates
- Private markets carry growing weight in capital raising; issues with opaque-ness, liquidity, agency, overcrowding
- Corporate and public pension costs rise to pay for increased longevity and reduced returns
- Disappointment with outcomes rubs off on trust; investment skill under pressure to demonstrate its value
- Geopolitical instability connects with social instability; inequality fissures; negative feelings deepen; job fears; immigration challenges

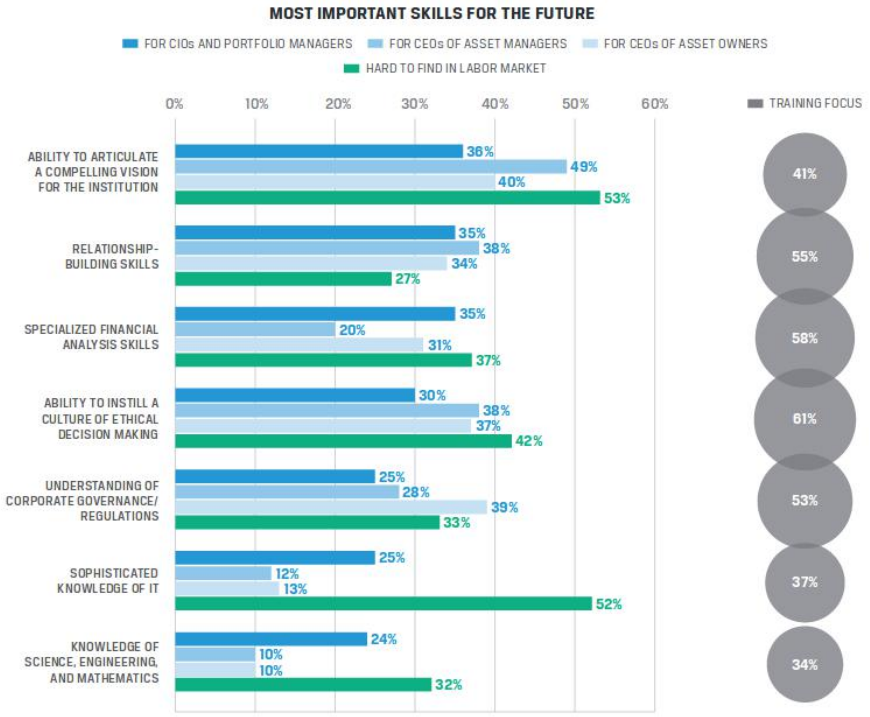
Purposeful Capitalism

Capitalism's way of working evolves; the investment industry raises its game with more professional, ethical, and client-centric organizations acting in aligned-to-purpose, lower-cost, and efficient ways

Major Elements

- Governments and firms work toward a more positive direction of travel for capitalism with more respect for wider stakeholders
- Markets for publicly listed equity and private equity are more fair, efficient, and deep over time and grow as a result
- Firms and investment organizations integrate their wider purpose alongside their profit motivations
- Asset owners are more influential; they add focus to longer-term value creation and sustainability
- There is an increased attention to fiduciary responsibility in investment with better alignment
- Fierce competition for leadership talent among investment organizations; diversity and culture are draws
- Investment providers need to have a "clean license to operate" including ESG principles

ORGANIZATIONAL GAME CHANGERS중 Game Changer 1: New Skills for New Circumstances은 미래를 위한 전문직의 기술을 정리하고 있습니다. 기장 많은 비율을 차지한 것이 SPECIALIZED FINANCIAL ANALYSIS SKILLS입니다. 기술, 데이타에 의한 디지탈 대전화의 결과로 이해할 수 있습니다.



보고서는 총 120 쪽에 이를 정도로 방대합니다. 본 보고서는 요약본만 첨부합니다. 원문 전체는 앞서 URL 을 참조하시길 바랍니다.

● 기술

BMLL Technologies 의 비지니스모델

미국과 유럽에서 만들어진 '핀테크'가 물을 건너오면서 핀테크가 가지는 혁신성과 창조성은 사라지고 모방만 남았습니다. 한국일보 4 월 25 일자 이종필의 제 5 원소 우리가 놓치고 있는 것들을 보면 한국의 창업이 선도자가 아니라 추격자자인지를 설명하고 있습니다.

이스라엘이 자랑하는 세계 최고수준의 위성, 로켓, 항공기 등의 기술은 대부분 군사적 목적에서 개발되었다. 인접국들과의 전쟁위협을 늘 이고 살아야 하는 이스라엘이 자기가 처한 문제를 가장 적극적으로 해결하는 과정이 낳은 산물이다. 나노 낙하산도 그 중의 하나이다. 2년쯤 전 어느 교수로부터 바이스 교수의 이야기를 처음 전해 들었을 때 나는 왜 한국에서는 이런 기술이 나오지 않을까 생각해 보았다. 당시에 한국에서도 드론이나 3D 프린터, 인공지능 같은 첨단기술에 대한 관심이 무척 높았다. 한 가지 큰 차이점은 문제에 접근하는 방식이었다. 대체로 과학기술 선진국에서는 자신이 해결해야 할 과제부터 명확하게 설정하고 그 과제를 해결하는 데 도움이 되는 가능한 모든 수단과 아이디어를 총동원한다. 우리는 순서가 반대이다. 선진국에서 이 기술이 뜬다고 하면 그것으로 우리는 무엇을 할 수 있는가를 고민한다. 우리가 직면한 과제를 해결하는 과정에서 나온 기술이 아니라 남이 좋다고 하니까 모방해서 얻은 기술이다. 첨단 기술이 나올 때마다 언제나 '한국형'이라는 수식어가 나붙는 것도 이 때문이다. 훌륭한 추격자는 될지언정, 결코 선도자는 되지 못한다.

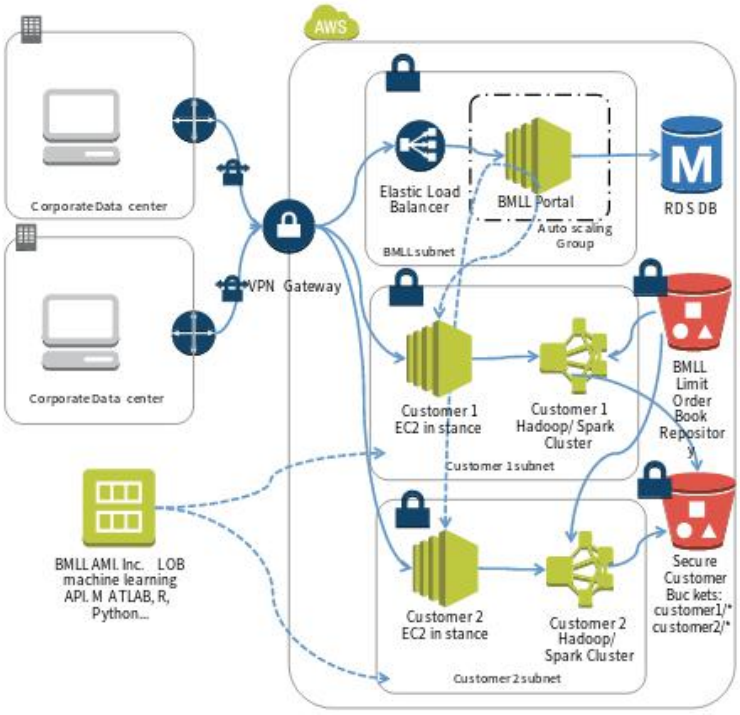
핀테크라고 하면 인터넷은행, 로보어드바이저, 지불(Payment)을 떠올립니다. 금융투자산업과 관련한 핀테크는 로보어드바이저뿐인 것처럼 보여집니다. 창조는 하늘에서 떨어지는 것이 아니라 익숙한 것을 낯설게 보는 끝없는 도전에서 이루어지는 것으로 이해합니다. 그런데 영국 bmll technologies 의 비지니스모델은 익숙한 것과 새로운 것을 결합한 서비스입니다. 익숙한 것은 Limit Orderbook 입니다. HFT 와 알고리즘트레이딩이 각광을 받을 때 수많은 사람들에게 회자하였지만 지금은 무관심한 부문입니다.

BMLL Technologies 는 크게 세가지 기술을 결합하여 사업모델을 만들었습니다.

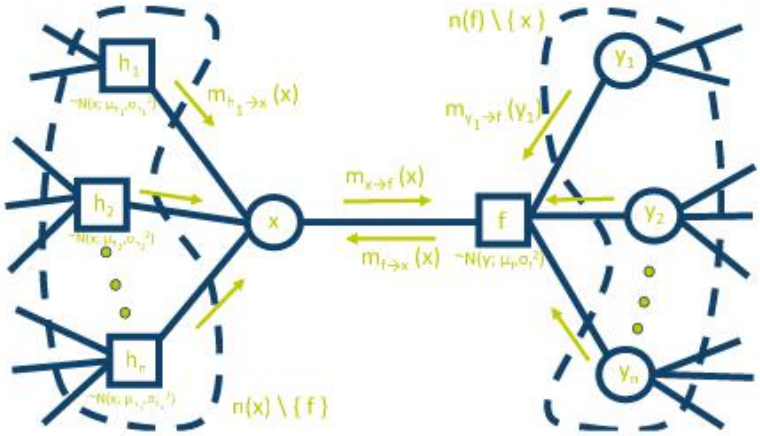
- Big Data Analytics
- Bayesian Machine Learning
- Limit Order Book Data

먼저 빅데이타분석환경을 구축하기 위하여 Apache Spark, Hadoop, Elastic search을채택하였고 PostgreSQL을 이용하고 RESTful microservice로 구현

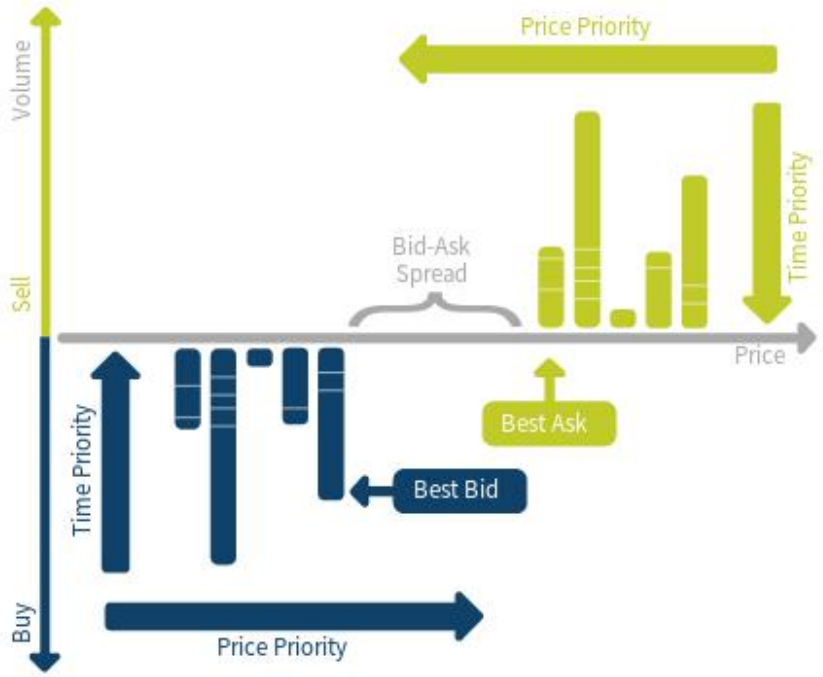
하였고 AWS를 이용하고 있습니다.



데이타를 분석하는 방법은 Bayesian Machine Learning을 적용하였습니다. 여러가지 논문의 이론적 모형을 적용하였고 그중 하나가 Rebuilding the Limit Order Book: Sequential Bayesian Inference on Hidden States"입니다.



마지막으로 유럽의 거래소에서 실시간 호가데이타를 받습니다.



BMLL Technologies가 모형으로 도입한 논문중 Rebuilding the limit order book: sequential Bayesian inference on hidden states은 회원만 원문을 구할 수 있습니다. 대신 The Journal of Trading 8 (3), 68-95 (2013)에 기고한 Prediction of Hidden Liquidity in the Limit Order Book of GLOBEX Futures의 원문을 첨부합니다.

● 전략

Tensorflow을 이용한 주가예측모형

지난 4월 18일 코스콤이 주최하는 자본시장IT컨퍼런스가 있었습니다. 주제는 '4차산업혁명시대의 자본시장IT전략'이었습니다. 발표중 '금융분야 인공지능 활용분야 및 사례'는 Deep Learning을 주제로 하였습니다. 딥 런닝을 적용한 트레이딩 기술 에서 소개하였던 논문이 주요한 사례로 언급되었습니다.

"금융분야 인공지능 활용분야 및 사례"를 발표한 김석원 지능정보기술연구원 박사는 "인공지능이 인간 일자리를 위협할 것이라는 얘기가 있는데 앞으로 5~10년 정도는 큰 영향을 미치지 않을 것"이라고 하면 아래와 같은 제안을 하였습니다.

"인공지능(AI)의 경우 아카이브 사이트에 하루 50편씩 관련 논문이 올라오는데, 이때 오픈소스도 함께 공개돼 논문에 대한 검증이 전 세계적으로 이뤄지고 있다. 이러한 과정을 통해 AI가 혁신적인 상황에 접어든 만큼 자본시장도 공개 검증 대회 등을 마련해 내가 가지고 있는 문제를 다른 사람도 풀게 해야 한다"

실제로 SSRN이나 ARXIV와 같은 논문 아카이브 사이트를 살펴보면 인공지능과 관련한 수많은 논문을 접할 수 있습니다. 아래는 초보자를 위한 요약논문들입니다.

Leslie N. Smith
Best Practices for Applying Deep Learning to Novel Applications

Martin Zinkevich
Rules of Machine Learning: Best Practices for ML Engineering

Brett Wujek, Patrick Hall, and Funda Güneş
Best Practices for Machine Learning Applications

TensorFlow 는 기계 학습과 딥러닝을 위해 구글에서 만든 오픈소스 라이브러리로 가장 큰 관심을 받고 있는 오픈소스프로젝트입니다. 구글이 발표한 Machine Learning with Financial Time Series Data on Google Cloud Platform 은 기계학습과 자본시장예측이 결합할 수 있는지를 보여주는 사례입니다. 구글이 발표한 자료를 기초로 주가예측에 적용한 사례가 TensorFlow で株価予想シリーズ입니다. 소스는 tensorflow-stock-index 에서 확인하실 수 있습니다.

0 - Google のサンプルコードを動かしてみる

1 - 終値が始値よりも高くなるかで判定してみる

2 - 日経平均225銘柄の株価予想正解率ランキング～

3 - 日本3506銘柄の株価予想ランキング

4 - 実際に売買したら儲かるのかシミュレーションしてみる

5 - 大きく上がると予想されたときだけ買ってみるシミュレーション

6 - 学習データの項目を増やす！隠れ層のサイズも増やす！

7 - 株価が何％上昇すると予測したら買えばいいのか？

8 - どの銘柄を買うか

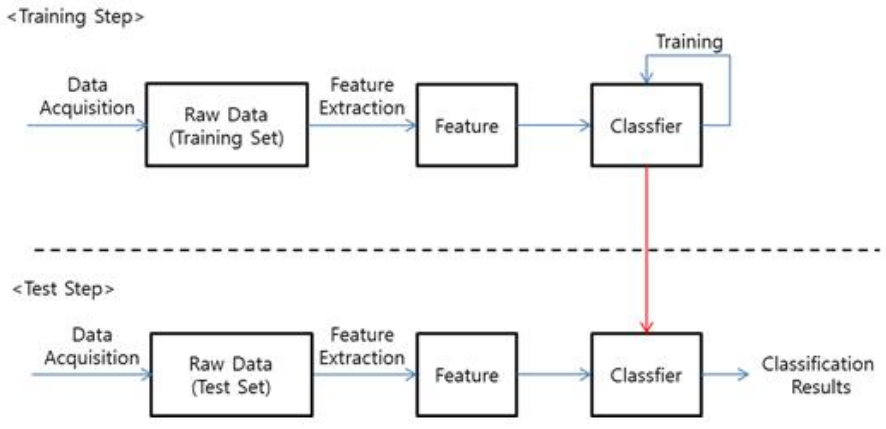
9 - 年利6.79%

Tensorflow를 이용한 또다른 사례는 Quantified Classifier를 구축하여 예측모형을만든 ML Quantized classifier in GO:A general purpose, high performance machine learning classifier입니다. Joe Ellsworth는 Bayes Analytic의 CTO 및 Algorithms research scientist를 맡고 있습니다. 구현한 모형에 대한 소개는

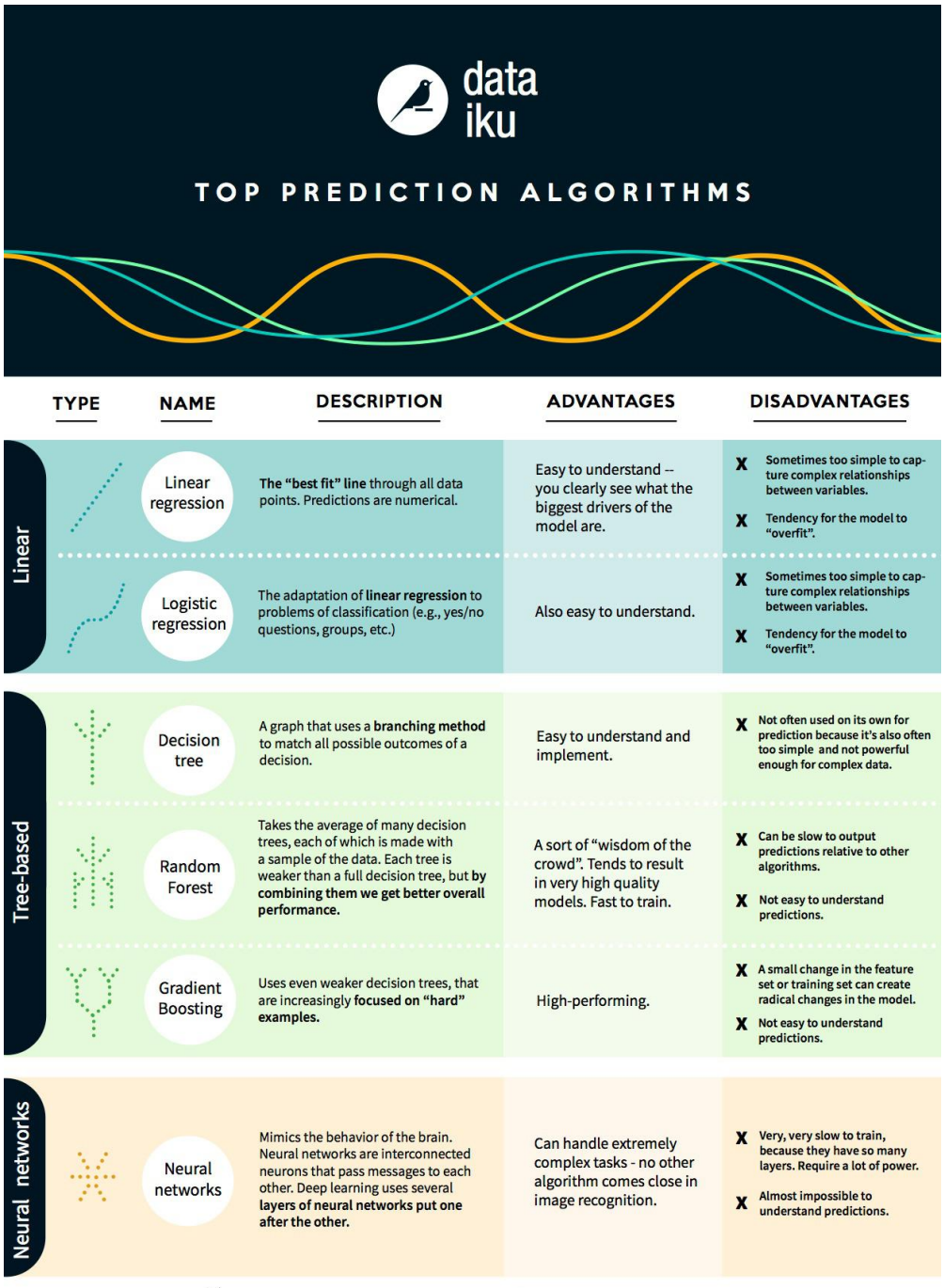
Stock Price Prediction tutorial
Analyze Predictive value of Features in stock price prediction
How can I use quantized classifier to make money?

으로 하고 있고 소스코드는 https://bitbucket.org/joexdobs/ml-classifier-gesture-recognition/src 에 위치합니다.

이 글은 LinkedIn에서 읽었는데 Classification 혹으 Classifier라는 개념이 낯설었습니다. Machine Learning 스터디 (8) Classification Introduction (Decision Tree, Naïve Bayes, KNN)을 읽어보니까 Classification을 다음과 같이 설명합니다.



Classification은 Supervised Learning의 일종으로, 기존에 존재하는 데이터와 category와의 관계를 learning하여 새로 관측된 데이터의 category를 판별하는 문제이다. 스팸 필터를 예로 들어들어보자. 스팸 필터의 데이터는 이메일이고, category, 혹은 label, class는 spam메일인지 일반 메일인지를 판별하는 것이 될 것이다. 스팸필터는 먼저 스팸 메일, 그리고 일반 메일을 learning을 한 이후, 새로운 데이터 (혹은 메일)이 input으로 들어왔을 때 해당 메일이 스팸인지 일반 메일인지 판별하는 문제를 풀어야하며, 이런 문제를 classification이라고 한다.



● 회사 소개

트레이딩컨설팅그룹 이음

)

트레이딩컨설팅그룹 이음은 개인 기업이지만 남들과 다른 사업방식을 택하고 있는 기술회사입니다. 뜻을 함께 하는 여러 파트너들을 모아 공동사업을 기획하고 추진하는 협동(Co-Opperative)방식으로 사업을 추진합니다. 그렇다고 협동조합처럼 소유도 협동으로 하지 않습니다. 사전에 합의한 계약에 따라 위험과 이익을 나눕니다. 주력사업이라고 할 수 있는 ZeroAOS는 기획/영업/전략, 주문및위험관리, 클라이언트(C++) 및 웹디자인부문의 파트너가 있습니다. 트레이딩컨설팅그룹 이음이 하는 구체적인 일은 아래에서 확인하실 수 있습니다.

트레이딩컨설팅그룹 이음이 하는 일

연락처는 아래와 같습니다.

twitter: @smallake
email: smithkim.kr@gmail.com
facebook: www.facebook.com/smallake

● 보고서

논문 및 보고서

보고서에서 소개한 논문의 원문을 첨부형식으로 제공합니다.모든 첨부물에 대한 저작권은 각 보고서에 따릅니다.

CFA Institute

APRIL 2017

FUTURE STATE OF THE INVESTMENT PROFESSION

PURSUING BETTER OUTCOMES—FOR THE END INVESTOR, THE INDUSTRY, AND SOCIETY

Executive Summary

PREPARING NOW FOR A DIFFERENT FUTURE

The future of the investment industry is important for the functioning of the global economy, for the approximately 2 million workers it employs, and for the clients and end investors that depend on it to manage around $100 trillion in assets.

This report, which includes findings from a survey of 1,145 industry leaders, addresses the issues that keep investment management executives up at night; they are the same issues that matter for CFA Institute as the largest association of investment professionals. Major shifts are underway that will likely result in significant change, and leaders need a better way to think through the implications of these shifts in various combinations—for their clients, the health of the industry overall, and the ongoing sustainability of their own firms.

Relevant megatrends include technological advances, redefined client preferences, new macroeconomic conditions, different regulatory regimes reflecting geopolitical changes, and demographic shifts. The industry's potential future state is further complicated by important issues that are very specific to investment organizations, such as trends in digitization and commoditization, downward pressure on fees, pressures from sustainability, new tech-centric business models, and other investment innovations. The scenarios and analysis in the pages that follow offer a road map for leaders in their strategic decision making as they seek to chart a course for the future of their firms.

This report also provides insights for professionals interested in becoming future industry leaders by identifying the traits and abilities that will be prized by future investment management organizations. Finally, it suggests ways that the *possible future states* of the investment industry could be influenced so that the *actual future state* provides the best possible outcomes, by fulfilling client objectives, serving end investors, and contributing to societal wealth and well-being.

Business Model Faces Pressure

84% of investment leaders expect consolidation of the industry

70% expect investors will increase their allocations to passive investment vehicles

52% of CFA charterholders surveyed expect substantial or moderate contraction of profit margins for asset management firms

57% expect institutional investors will look to reduce costs by insourcing more investment management activities

Changing Client Profiles

73% of investment leaders expect environmental, social and governance factors will become more influential

70% of investment leaders expect Asian financial centers will become more influential

Opportunities on the Horizon

55% of investment leaders expect globalization will offer new opportunities for investment professionals

48% of investment leaders expect technology will offer new opportunities for investment

UNDERSTANDING THE INDUSTRY'S PURPOSE THROUGH ITS MANY MOVING PARTS

The fundamental purpose of finance is to contribute to society through increases in societal wealth and well-being. Looking at finance as an ecosystem reveals important interconnections and points of friction in how finance currently works in relation to this purpose. The financial ecosystem is:

- **Connected:** It reflects the multiple diverse participants, people and organizations and their connections with each other and with the wider landscape. While the system is served by many specialists, there is a need to understand the bigger picture.
- **Reflexive:** It incorporates the two-way nature of those connections and dependencies. Specifically, it allows for reflexivity, where landscape changes affect and are affected by participants' beliefs and actions.
- **Non-linear:** It allows for the jumps, or tipping points, that characterize some of the properties of the system and are difficult to explain with traditional theory. Simply put, crises happen.

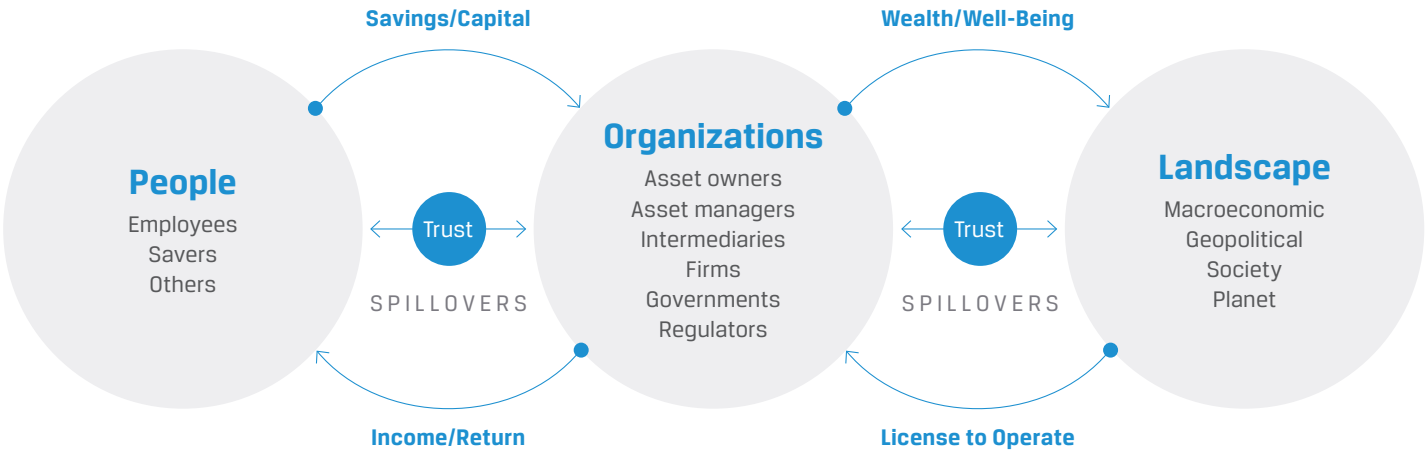
The financial ecosystem rests on a singular fundamental transaction: those with a surplus of capital but a deficit of ideas (investors) provide capital to those with a deficit of capital but a surplus of ideas (inventors, entrepreneurs, businesses, firms, etc.). When those ideas are successful, then both the providers and users of capital benefit by earning investment returns. CFA Institute argues that an investment industry of enormous value to society has grown from this kernel; but its sustainability is dependent on the nature of the value delivered and the quality of trust between the end investor and the organizations involved.

Uncontrollable forces consistently exert influence (sometimes extreme influence) in the financial ecosystem, just as they do in the natural world. The challenges the investment industry will face in the future are currently being shaped by a number of megatrends that already have significant momentum: people are living longer and demographic structure is altering markedly, technology is empowering individuals and organizations, economic imbalances continue to grow in markets and society, the regulatory pendulum is swinging faster, and natural resources are under stress.

Our focus here is on the investment function of finance, which lies alongside the payment, lending, and insurance functions. More specifically, the core purposes of the investment industry lie in two overlapping areas:

- **Wealth creation:** Mobilizing capital for jobs and growth; the capital managed in this chain creates wealth and well-being.
- **Savings and investments:** Deploying investment services for wealth and risk management; the savings and investments managed in this chain allow inter-temporal (over time) risk management and increases in wealth.

THE FINANCIAL ECOSYSTEM



THE ART AND SCIENCE OF SCENARIO PLANNING

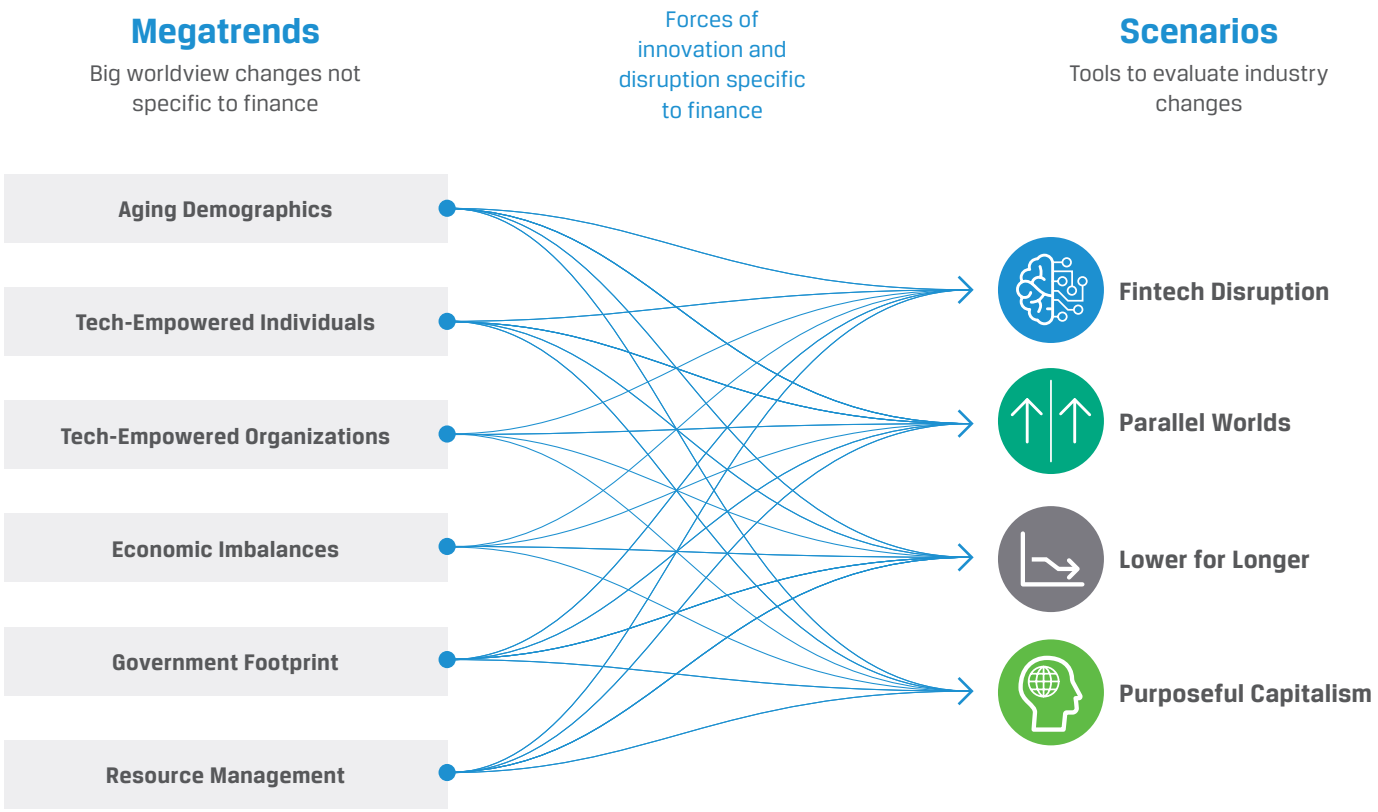
Even when forecasts are directionally correct in finance, they are usually specifically wrong. That is because the future of finance is created by a combination of many moving parts and legions of complex interactions; the result is inherently impossible to predict. Consequently, we use scenario planning to reveal insights about the future state of the investment profession, regardless of what future unfolds.

Our scenarios draw on a number of megatrends—large scale changes in circumstances that are omnipresent in all facets of our world—that are identified as virtually certain to disrupt the ecosystem regardless of how the future unfolds.

The megatrends are mixed with finance-specific forces in different combinations to create unique scenarios in the form of narratives about the future. These narratives are not forecasts; instead, each narrative strives to tell a unique story. With these stories in mind, decision makers are equipped to recognize the narratives as the future unfolds and act early.

Our time frame is 5–10 years, which is long enough to allow business models to substantively change in response to the disruptive megatrends and forces we identify, but not so long as to be overly futuristic.

MEGATRENDS COMBINE FOR POSSIBLE FUTURES



FOUR SCENARIOS FOR FUTURE STRATEGY

These scenarios represent four ways in which the future of investment management could unfold.

Fintech Disruption

New technologies promote new business models; disruption and creative destruction are endemic; challengers do better than incumbents; major disruptions to the world of work

Major Elements

- Quickening flow of disruptions from technological innovation in digitization and digitalization
- Fintech develops globally with a particularly strong Asia-Pacific element
- Regulatory infrastructure in finance gradually integrates technology-driven models
- Disruptions to investment organization business models; success with technological advancement is critical
- Traditional active management shrinks; some growth in alternatives, smart betas, and outcome-oriented solutions
- Smart machines and systems, data analysis, and inference play a disruptive role in finance's evolution
- Financial services becomes highly personalized and digitalized everywhere
- Robo-advice and its "cyborg" variants become preferred style or tool for delivering investment advice

Parallel Worlds

Different segments—by geography, generation and social group—engage in society differently; a higher baseline for financial services participation with wider dispersion; product preferences for personalization, simplicity and speed

Major Elements

- Better worldwide education, healthcare and telecoms increase societal engagement
- Social media carries potency to bring people together and to divide, legitimately and illegitimately
- Potential for mass disaffection; consequences in anti-globalization, populism, and authoritarian nationalism
- New-style financial institutions enabling personalized, simple, and speedy engagement; trust is also needed
- Big data serves customiza-tion of investment products to specific segments; more reflection of personal values
- Improvement in financial literacy and empowerment produce better financial participation
- The "have-nots" act on their disillusionment with the system
- The trustworthiness of the tech model with tangible products and immediate gratification is tested in investment contexts

Lower for Longer

New normal low interest rates and returns become embedded for the foreseeable future (5–10 years), accentuated by lower levels of global growth and higher levels of political instability

Major Elements

- Limited success with interest rate normalization; natural interest rates stay low
- Growth challenges: indebtedness, adverse demography, excess savings, China/EM, companies hoard cash
- Large gaps in pension coverage with longevity; pension poverty
- Moves to lower-cost, higher-tech investment solutions; premium on innovation; industry consolidates
- Private markets carry growing weight in capital raising; issues with opaque-ness, liquidity, agency, overcrowding
- Corporate and public pension costs rise to pay for increased longevity and reduced returns
- Disappointment with outcomes rubs off on trust; investment skill under pressure to demonstrate its value
- Geopolitical instability connects with social instability; inequality fissures; negative feelings deepen; job fears; immigration challenges

Purposeful Capitalism

Capitalism's way of working evolves; the investment industry raises its game with more professional, ethical, and client-centric organizations acting in aligned-to-purpose, lower-cost, and efficient ways

Major Elements

- Governments and firms work toward a more positive direction of travel for capitalism with more respect for wider stakeholders
- Markets for publicly listed equity and private equity are more fair, efficient, and deep over time and grow as a result
- Firms and investment organizations integrate their wider purpose alongside their profit motivations
- Asset owners are more influential; they add focus to longer-term value creation and sustainability
- There is an increased attention to fiduciary responsibility in investment with better alignment
- Fierce competition for leadership talent among investment organizations; diversity and culture are draws
- Investment providers need to have a "clean license to operate" including ESG principles

ORGANIZATIONAL GAME CHANGERS

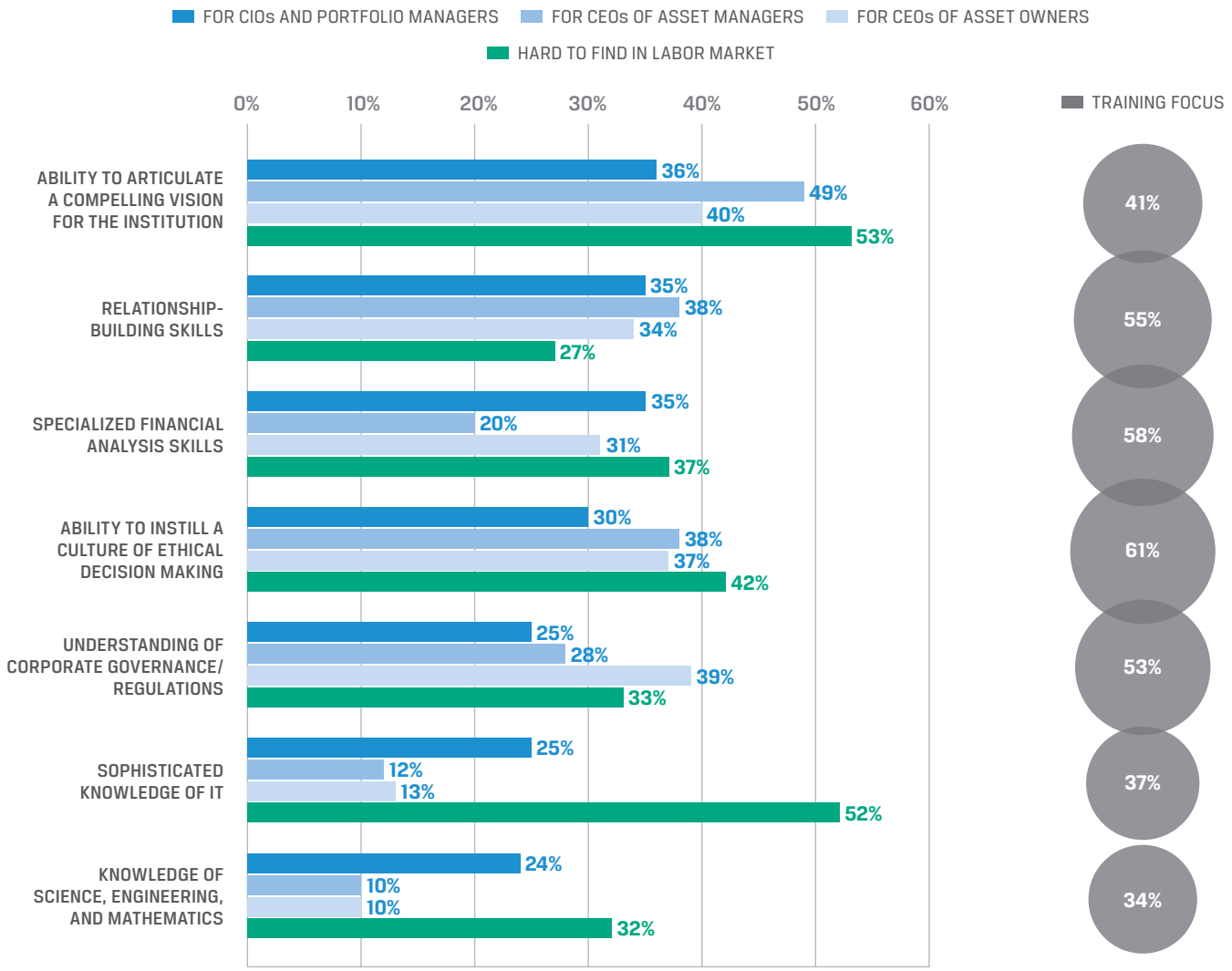
Investment organizations can be divided into asset owners, asset managers, and investment intermediaries. These are all "people businesses," dependent on talented leaders and staff to move forward. Our survey asked about the most important skills for leaders in the future, and the results indicate that investment organizations need to recruit and develop employees along new dimensions.

Investment organizations looking to retool for the future face some challenges. It is particularly difficult to find people with the ability to articulate a compelling vision for the institution and to instill an ethical culture—the two most-valued skills. But financial analysis skills and ethics rank at the top of the list for training.

Game Changer 1: New Skills for New Circumstances

- Increasing need for soft skills, like creative intelligence and influencing skills, given that technology will replace many straightforward human processes
- Adaptiveness to change is needed for increasingly disrupted situations, but this skill is in short supply
- Training requires attention to ethical and professional orientation
- Organizations need to increase their understanding of the interaction of skills in group settings
- There is a critical need for increased diversity both for a business case and improved cultural strength

MOST IMPORTANT SKILLS FOR THE FUTURE

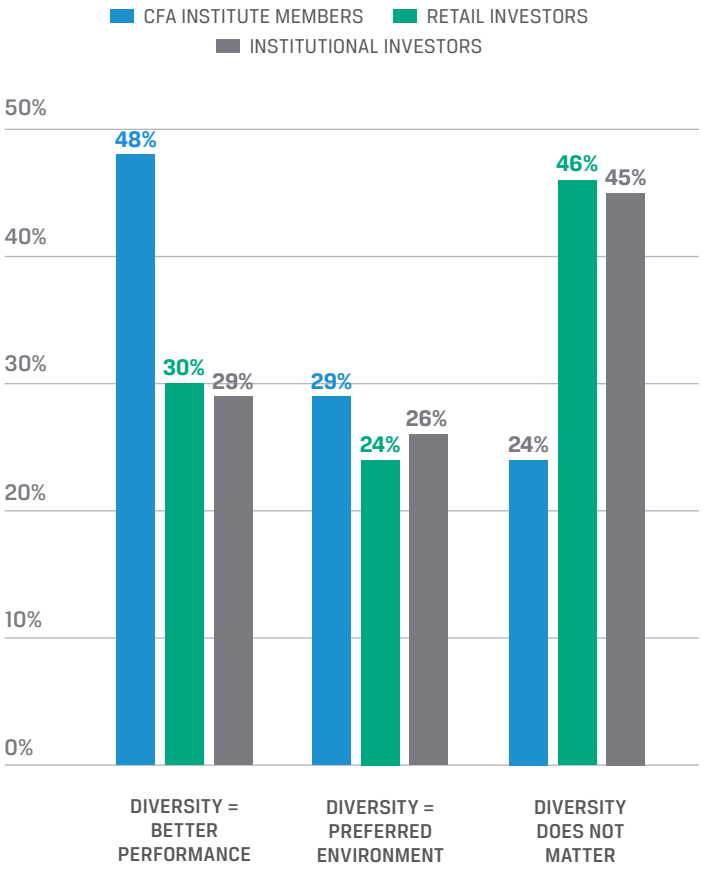


The Role of Diversity

The other major dimension of workforce capabilities will be the contribution of improved diversity. Diverse people are most often identified through surface characteristics (gender, race, national culture, education, sexual orientation, age, etc.), but the business case for diversity is linked to intrinsic individual characteristics, such as values, perspectives, experiences, knowledge, and way of thinking.

Diverse groups benefit from more and different ways of seeing complex problems and, thus, better ways of solving them. A growing body of research has shown the link to better performance and better culture that a gender diverse industry could have. There are opportunities to benefit from cognitive diversity and overcome the risks of groupthink.

WHEN IT COMES TO THE GENDER DIVERSITY OF A TEAM OF INVESTMENT PROFESSIONALS, WHICH ONE OF THE FOLLOWING BEST DESCRIBES YOUR VIEW?



The benefits of gender diversity to improve outcomes are beginning to be understood by the investment community, as indicated by the accompanying chart from earlier CFA Institute research.

These issues warrant increasing leadership attention given that the materiality of behavioral factors to decision outcomes has become clearer, and is in synch with the thinking and methodologies to make progress on this front.

Game Changer 2: New Pensions and Lifetime Savings Models

- Private pension markets are barbelled—adapting the mature markets and developing the immature ones
- Best practice is usually found where there are "win-win" alignments between far-sighted sponsoring firms and well-governed pension funds
- Pension engagement and advice is ripe for disruption—it needs new models that use technology more efficiently
- Increasingly, private pensions will follow the Defined Contribution (DC) model with a flexible investment platform and behaviorally smart design
- Low levels of contributions and low returns produce inadequate retirement income; working later in life is necessary

Game Changer 3: Evolving States of Trust

- Trust is mediated by the values, competencies, and transparency of our investment organizations
- Trust reflects a particular type of communication model: communicate early, fully, and often and to fill gaps in understanding
- To build trust, show some societal responsibility; deliver to expectations in competency and ethical practice; add consistent value
- Trust will rise in the industry if selection of future talent emphasizes strong values
- Trust will be influenced in future by innovation in technology—for example, blockchain technology distributes trust

THE TRUST EQUATION

Trust from an end investor is the *dependency on a service provider in a situation of risk over a prolonged period.*

The type of trust expected by an end investor is far more complex and tacit than the trust expected by the end user of most any other product, regardless of its type or business sector of origin. Its importance grows with the size of risks taken and the length of the term of the relationship—making it core to investment service delivery.

Trust and value in investment are interconnected. For the end investor, value will relate to perceptions of outcome relative to expectations. (In other words, do not think first of performance versus benchmarks as these do not represent particularly relevant expectations for most investors.) Value and trust are developed by an individual or an organization by building **credibility** and demonstrating **professionalism** as captured in the **"Trust Equation."**

The Trust Checklist for Organizations

At a simplistic level, a highly professional firm is filled with many highly professional individuals. To achieve this across an entire organization, however, a complex coordination challenge must be met, and its solutions require good culture and an appropriate business model to secure alignment to the necessary attributes of credibility and professionalism. Trust in the context of the investment organization spans a spectrum of critical attributes.

T Transparency

Organizations should display "glass door transparency" of all things, including business processes, limitations of the investment process, risks, performance reporting, fees and their impact on portfolios, and potential conflicts of interest. They should be candid about the mistakes they have made and explain what steps they are taking to correct them.

R Realistic Measures

Firms and their employees need to be realistically measured in relation to financial and non-financial goals over relevant time horizons. End investors are concerned with outcomes.

U United Values

Alignment of values between firms and all of their stakeholders is critical. Organizations build their strongest trust by being aligned in their purpose, objectives, and way of working with those they serve.

S Sustainable and Fair Rewards

Fees and rewards need to be fair and reflect the value clients receive. Trust will best be secured when there are incentives for agents to do their absolute best for their clients.

T Time-Tested Relationships

Good relationships develop over time and allow the client to develop confidence. Research shows that people are much more trusting when working with consistent partners—a situation which offers a chance to build a good reputation through repeated interactions.

TOWARD GREATER SOCIETAL BENEFITS

Aspirations Align with Need

Making a consistent and determined contribution to societal wealth and well-being is not just a nice goal for the investment management profession—it is quite possibly a matter of existential importance. The good news: the research shows that investment professionals aspire to a more positive social contribution.

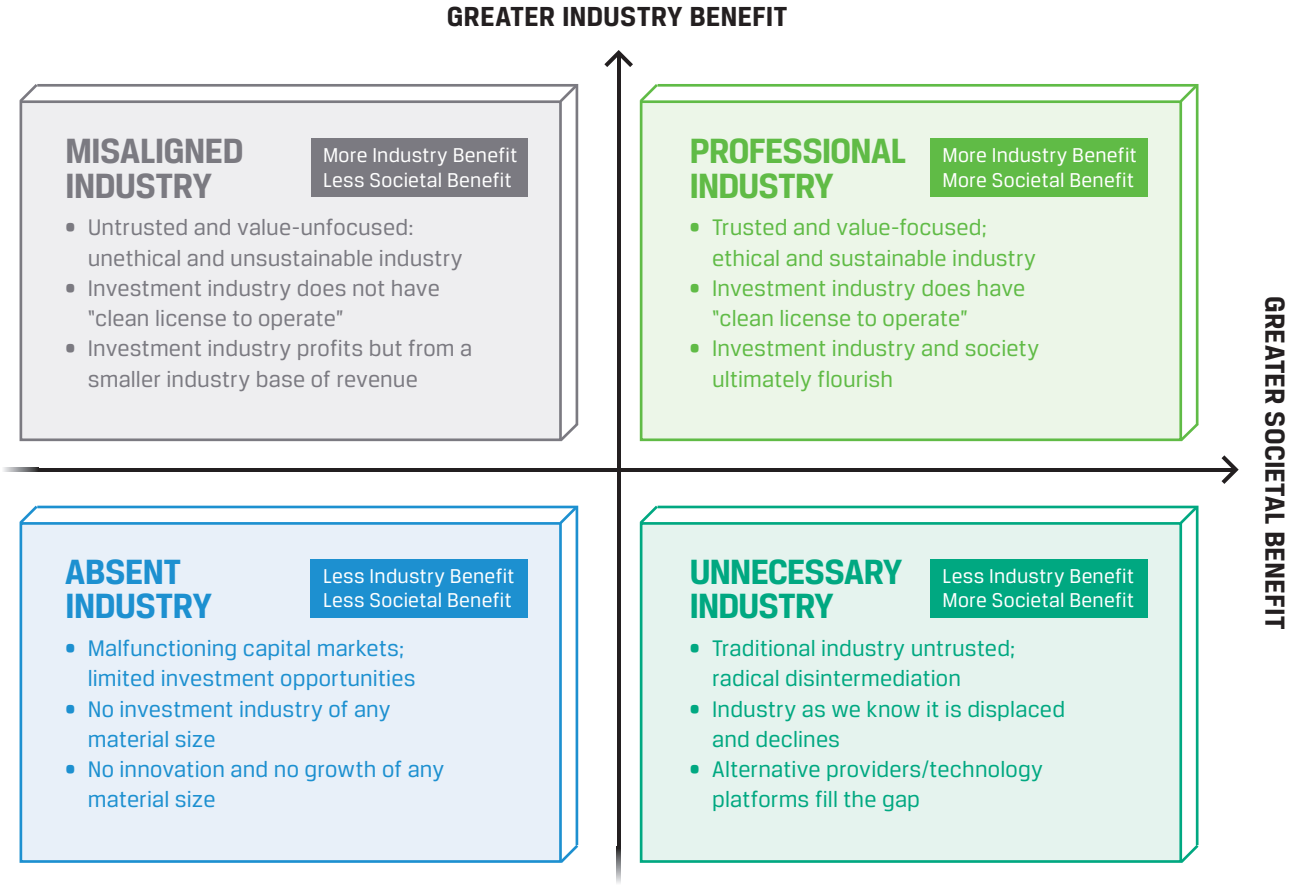
We offer below a model for creating a healthy investment industry by looking at the potential outcomes from the interaction of differing levels of industry versus societal benefit.

Potential for the Industry

11% **of investment leaders describe the impact of the investment industry as very positive for society today**

51% **of investment leaders expect the impact of the investment industry to be very positive for society *contingent on stronger principles***

FOUR STATES FOR THE RELATIONSHIP BETWEEN INVESTMENT MANAGEMENT AND SOCIETY



IDEAS TO MOVE THE NEEDLE TO A BETTER INDUSTRY

The research that underpins this report tells us that change is coming. We have anticipated how that change might play out in planning scenarios, and what state the industry could end up in depending on how well it adapts to change and to what degree it earns (or does not earn) widespread public trust.

We believe the following "to-do" list and a road map derived from it can be the first steps in the journey toward a future where a healthy investment management profession benefits societal wealth and well-being.

Professional Transformation: Identify What Is Needed to Go from Industry to Profession

An "industry" is defined by the circumstances of producing something of value to a consumer. A "profession" is different; it extends a license to operate to individuals and organizations that is granted and maintained by containing requirements for entry, standards of fair practice, disciplinary procedures, and continuing education for its professionals in conditions of an ongoing relationship. In doing this, the profession combines value and trust. The trust in this arrangement is of considerable value, not least because it creates the conditions for growth in wealth and well-being. The open questions are: How can the investment industry evolve so that it shares identifiable and key characteristics in the manner of medicine, law, and accounting? What is the current gap? What benefits could arise from filling this gap?

Fiduciary Implementation: Master the Meaning of "Fiduciary" in a Way That Can Be Effectively Implemented Even with Inherent Conflicts

Fiduciary responsibility in most jurisdictions means putting the interests of beneficiaries first when determining investment strategy, limiting conflicts of interest, and investing to the standard of care of a prudent expert. All investment organizations face the practical issue of balancing these requirements within the context of their own viability. CFA Institute will be conducting further engagement on how organizations should be dealing with fiduciary responsibilities and other issues where legal and regulatory frameworks are at potential conflict with the ambiguities and uncertainties endemic to the investment field.

Stronger Standards: Specify and Influence Culture and Practice with Regard to Values and Costs

CFA Institute successfully introduced standards for presentation of performance records in the form of the GIPS® standards. There may be other areas of practice that could benefit from such an approach. We cite standards for the structure and size of fees and costs as one possible idea. We also believe that the testing of new types of investment products could be the subject of standards in ways that draw on practices in other industries and professions.

Work toward Better Diversity

Diversity is desirable for a combination of cultural and financial values. Research suggests that diverse teams are better at complex decision making and that surface-level diversity issues, such as gender, have a first-level impact, but that cognitive diversity is more deeply impactful. CFA Institute is developing a mix of research, advocacy, and standards to support this developing field.

Leverage the Ecosystem

CFA Institute has more than 146,000 members worldwide, and this vast group is sometimes tapped to achieve a global view on a wide spectrum of issues. We are struck by the potential of networks empowered by new technologies to focus political and social capital in particular areas. Our membership can speak more powerfully for society's benefit through such a mechanism, particularly if it speaks with one bold voice.

A WAY FORWARD

We have put forward a number of steps and ideas by which the investment industry can realize its fullest potential, and we now encourage our members and industry leaders to act to make this a reality.

CFA Institute is committed to further consultation with leading industry figures on the following:

- Creating a road map for moving our industry to higher standards of professionalism, with its implications for fiduciary responsibility and for attaining the status of a profession
- How we can work together on the most pressing industry issues, particularly business models that capture purpose, trust, and value; cultural values that are inclusive; and technological competencies that streamline our industry
- How the CFA Program can maintain its edge in fast-moving industry conditions

The future is a choice to be taken by you applying foresight and coherent actions...not an outcome given to you reflecting uncertainty and compelled reactions

About the Report

In 2016, CFA Institute commissioned the Institutional Investor Thought Leadership Studio to survey members of the investment management profession for an overview of the current and future state of the profession. A questionnaire was distributed to two lists, one drawn from Institutional Investor's database, the other from CFA Institute. There were 1,145 responses (644 from CFA Institute) collected from 8–22 December 2016, with a margin of error of 2.9%. In addition, Institutional Investor conducted interviews with 19 executives in the investment management profession to obtain context and further details about the collected data.

Authors:
Rebecca Fender, CFA
Robert Stammers, CFA
Roger Urwin, FSIP
Jason Voss, CFA

Contributors:
Giuseppe Ballocchi, CFA
Richard Brandweiner, CFA
Anne Cabot-Alletzhauser
Margaret Franklin, CFA
Lutfey Siddiqi, CFA

Steering Committee: Gary Baker, CFA, John Bowman, CFA, Michael Collins, Bjorn Forfang, Stephen Horan, CFA, Nick Pollard, Nitin Mehta, CFA, Kurt Schacht, CFA, Paul Smith, CFA

Additional thanks go to Bristol Voss, Nicole Lee, Tara Smith, and Melissa Carroll of CFA Institute, and Sam Knox of Institutional Investor Thought Leadership Studio, as well as the many industry leaders who participated in the research.

CALL-TO-ACTION FOR THE INDUSTRY

Start with Purpose
Enlightened self-interest

Build the Talent
Values critical

Plan for Disruption
Foresee the scenarios

Produce the Value
Outcomes versus expectations

Make Trust Happen
The biggest edge of all

CFA Institute

CFA Institute is the global association of investment professionals that sets the standard for professional excellence and credentials.

The organization is a champion for ethical behavior in investment markets and a respected source of knowledge in the global financial community. The end goal: to create an environment where investors' interests come first, markets function at their best, and economies grow.

CFA Institute has more than 146,000 members in 160 countries and territories, including 140,000 CFA charterholders and 147 member societies.

The CFA Institute Future of Finance initiative is a long-term, global effort to shape a trustworthy, forward-thinking investment profession that better serves society.

For more information, visit **www.cfainstitute.org/futurefinance** or contact us at **FutureFinance@cfainstitute.org** to offer your ideas about how to shape the industry for the future. We encourage you to cite this report using the link **www.cfainstitute.org/futurestate**

Institutional Investor

Institutional Investor is among the world's leading investment information brands. Its highly regarded content reaches the world's most influential investors across an array of media platforms, conferences, capital markets databases and emerging markets information services. Institutional Investor's Thought Leadership Studio works closely with its clients to execute independent primary research, and to create relevant content to amplify the findings.

www.cfainstitute.org

Prediction of Hidden Liquidity in the Limit Order Book of GLOBEX Futures

HUGH L. CHRISTENSEN AND ROBERT WOODMANSEY

HUGH L. CHRISTENSEN
is a researcher in the Engineering Department at the University of Cambridge in Cambridge, UK.
hlc54@cam.ac.uk

ROBERT WOODMANSEY
is the chief executive of Onix Solutions in London, UK.
robert.woodmansey@onixs.biz

Modern securities exchanges have the concept of the open *limit order book* (LOB), where any market participant can see all the orders in the market. *Hidden orders* are a variation on this theme, where certain orders are not visible in the LOB. The LOB can be considered a store of participants' future intentions. The ability to hide information in this store is detrimental to traders, who use that information to decide upon their future actions. In contrast, the owner of the hidden information benefits by removing the informational disadvantage associated with having their intentions publicly known and avoiding being exploited by subsequent participants with more timely information.

Hidden volumes in the LOB are of great interest to both *liquidity providers* (market makers) and *liquidity takers*. Liquidity providers can use knowledge of hidden volumes both to generate alpha and manage risk. In terms of risk management, hidden orders can result in a liquidity provider's limit order being "picked off" when the liquidity provider did not intend to execute that volume or was unaware of some new information that had come to the market. If the liquidity provider knew of the hidden order, he or she might choose to withdraw his or her volume from the LOB. A liquidity provider's alpha generation can benefit from knowledge of a hidden order in the LOB, either directly or indirectly. First, there are a number of strategies for directly exploiting the hidden volume: for example, "front running," a type of hidden order called an *iceberg order.* An iceberg order is a limit order where only a fraction of the total order size is shown in the LOB at any one time (the *peak*), with the remainder of volume hidden. If the iceberg is an order to sell, the strategy is to short the market to the estimated size of the iceberg, causing the price to decrease. When the volume of the iceberg has been hit and the price has decreased, the position is closed at a profit (Durbin [2010]). Second, and indirectly, some alpha generation strategies use the information content of the LOB to forecast future returns, and hidden orders lead to a false picture of the LOB, giving an erroneous prediction of future returns (Cont et al. [2010]). Liquidity takers are interested in using hidden orders to reduce their exposure risk by minimizing the announcement of their intention to trade, thus decreasing their market impact. In this way, informed traders can conceal the fact that they know useful information from the rest of the market (De Winne and D'hondt [2007]).

The motivation behind this article is to present an online algorithm that can be used to detect iceberg orders for the major derivatives exchanges. The emphasis of the

article is on the applicability of the research for use both in academia and in industry.

This article is structured as follows: First, we introduce the market participants to clarify why hidden volume is of relevance. In the next section, an overview of the exchanges is given and the data used in the paper presented. We then review the LOB, the different sorts of hidden volume are introduced, and we carry out a literature review on detecting icebergs. For the article's main contribution, we present an algorithm for detecting iceberg-hidden volume on GLOBEX and outline a low-latency computational implementation of the algorithm. The performance of the algorithm on LOB data is then simulated and results presented. Next, examples of how the algorithm can be used by the investment community are given. Finally, we draw conclusions and make suggestions for further work.

MARKET PARTICIPANTS

The application of quantitative approaches to trading is now a well-established field; however, the majority of participants do not have the ability to apply quantitative methods at a micro-structure level. This is still limited to a small subgroup of ultra-high-frequency traders. In the futures market, this subgroup is made up of the Principal Traders Group.[1] Unlike many quantitative hedge funds, this group is largely self-financed, and as such has a different outlook on risks and regulations. The main aim of this group is to participate in "low-risk" trading, which is limited to market-making and arbitrage activities. Given that these activities take place at very high frequency and result in short holding periods, the participants are required to trade very large volumes to generate their required returns. One such company, RSJ,[2] makes public its trading volumes for EUREX, CME GLOBEX, and NYSE Liffe derivatives exchanges, stating that their total monthly trading volume "exceeds 20 million lots." The data equate to 1.1%, 3.8%, and 4.7%, respectively, as a percentage of the total electronic volume traded on these exchanges in 2010 and 2011. These percentages are similar to figures released by the Scandinavian equities exchange, NASDAQ OMX, in 2011, which show that 10 high-frequency firms alone are responsible for 16% of the exchange's volume (Cave [2011]). Given the dependence of these firms on executing large volumes, detecting hidden volume is of particular economic relevance, as this is volume which could potentially be traded against, leading to increased profits.

DERIVATIVES EXCHANGES AND DATA

Algorithmic traders tend to be interested only in the most liquid, vanilla securities. To this end, we begin by considering the four leading Western derivatives exchanges by volume traded: CME GLOBEX, ICE, EUREX, and NYSE Liffe. These exchanges trade two main products: futures and options. While the volumes are similar in both products, there are higher quoting rates and fewer trades in options relative to the futures (Bank for International Settlement [2010]). The mechanisms by which electronic trading platforms operate tend to vary widely between exchanges, and we summarize the relevant points from these four exchanges in Exhibit 1. Due to their dominant market position, we proceed considering just CME GLOBEX, while noting that the contents of this article are relevant to the other exchanges with hidden volume. Descriptive statistics associated with GLOBEX and high-frequency trading are shown in Exhibit 2.

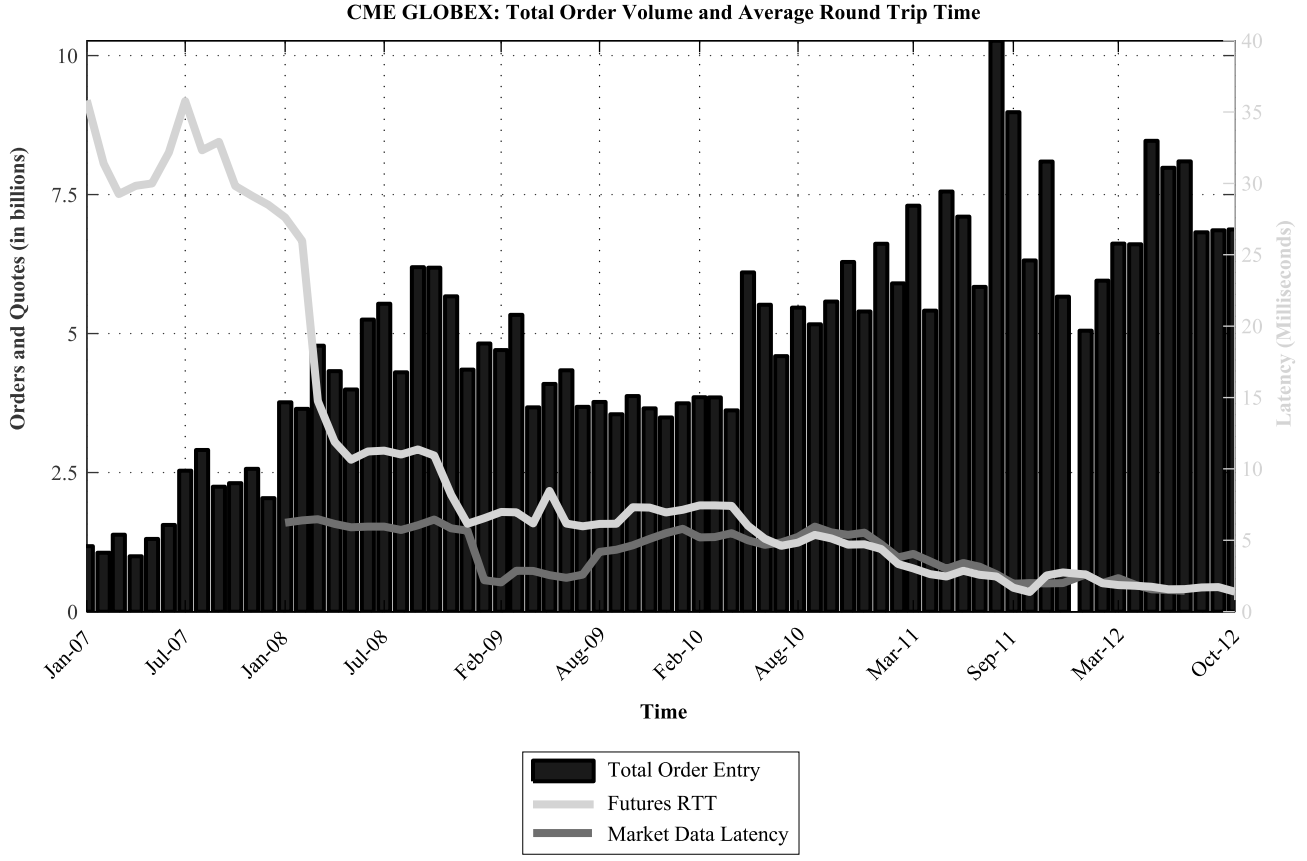
In the case of GLOBEX, options do not support implied pricing or, in some cases, iceberg orders,[3] and for this reason this article looks only at futures. Specifically, we consider the e-mini S&P 500 (ES) future for the 258 trading days of 2011. We use the CME datamine product, which is a recording of the real-time GLink session, is fully public, and can be accessed via the exchange (Chicago Mercantile Exchange [2012]). Unlike on many other exchanges, platforms, and dark pools, data

EXHIBIT 1
Derivatives Exchanges

	EUREX	CME	ICE	NYSE Liffe
Iceberg orders	✗	✓	✓	✗
Implied orders	✓	✓	✓	✓
Second-generation implieds	✓	✓	✓	✗
All implieds broadcast	✗	✗	✗	✓

EXHIBIT 2
CME GLOBEX Total Order Volume and Average Round Trip Time (RTT)

RTT is measured from the time the ilink gateway begins to process a message, processing by the match engine and subsequent outbound acknowledgement by the ilink gateway. Futures RTT and market data latency can be seen to be decreasing while total order volume is increasing.



access at CME is transparent and no parties have access to any special data that is restricted in any way, nor can "extra information" be purchased or subscribed to. Historical data were pre-processed from their raw FIX FAST structure into a structure suitable for analysis in MATLAB. This structure is shown in Exhibit 3 and constitutes one row in an ASCII file. For each day, for each futures contract, the data is parsed into a separate ASCII file. The final class of data used in this article is *static data*, which comprise system parameters that vary on a future-by-future or contract-by-contract basis, such as tick-size, implied quoting functionality, number of price levels in the LOB, and so on. (Chicago Mercantile Exchange [2013b]).

LOB Rebuild

Rebuilding the LOB is the process of taking the broadcast data and regenerating the multi-dimensional LOB. Fields 1-9 of Exhibit 3 are required for the basic rebuild of the LOB, and fields 7 and 10-15 are required to carry out trade matching. For GLOBEX, the LOB is reset weekly with the last data sent on Friday and the LOB being blank at Sunday start-up.

The deterministic rebuild process for GLOBEX is shown in Algorithm 1, as per Christensen et al. [2013]. The three dimensions of the LOB are side $\widetilde{S}$ (bid/ask), class (price/size/number of orders) and M price levels, such that $m = 1, \ldots, M$. dV is a vector size $1 \times T$ of the

EXHIBIT 3
GLOBEX Data Structure

#	FIX Tag	Name	Example Values	Description
1	1023	MDPriceLevel	1, 2, . . . , 10	Price level of the LOB
2	52	SendingTime	HH:MM:SS.FFF	Time of message transmission in UTC
3	269 & 270	BidPrice	1342.75	Bid price
4	269 & 271	BidSize	1	Bid size
5	269 & 270	AskPrice	1343.25	Ask price
6	269 & 271	AskSize	9	Ask size
7	279	MDUpdateAction	0, 1, 2	Type of market data update action. 0=new, 1=change, 2=delete.
8	346	NumberOfOrders	500	Number of orders in the LOB at that price level. Supported for explicit orders only.
9	276	QuoteCondition	K	K=implied, absent=explicit.
10	269 & 270	TradePrice	1343.00	Trade price
11	269 & 271	TradeSize	3	Trade size
12	277	TradeCondition	E,1	E=opening trade, 1=trade causes no price or volume change in the LOB. Broadcast price and volume are meaningless.
13	5797	AggressorSide	1, 2	Indicates which side is aggressor of the trade. 1=buy (trade volume comes out of the ask side of the LOB), 2=sell (trade volume comes out of the bid side of the LOB), absent=no aggressor (as per pre-open). Not sent in an implied spread or outright trade message.
14	336	TradingSessionId	0, 1, 2	Market state identifier. 0=pre-opening, 1=opening, 2=continuous trading.
15	5799	MatchEventIndicator	{SP H1-F1;H1;F1}	The legs of a strategy trade. Partially parsed from the FIX message structure.

changes in volume at each time step. $dV < 0$ relates to either trades, size-reducing order modifications, or order cancellations.

THE STRUCTURE OF THE GLOBEX LIMIT ORDER BOOK

On GLOBEX, two types of futures contract exist, each with their own LOB: single-leg (outright contracts) and multi-leg (strategy contracts). An example of a double-leg intra-product contract is a calendar spread. Additionally, on GLOBEX, two types of order exist: explicit orders and implied orders. Explicit orders are orders entered into the LOB by market participants. Implied orders are orders entered into the LOB by the trading system itself as a result of no-arbitrage arguments between single-leg contracts and multi-leg contracts. Implied orders exist to increase the market's liquidity and shift the high liquidity found at the front of the forward curve back down toward the less liquid contracts (Overdahl [2011], Blank [2007]). As implied pricing depends on an active futures curve, implied functionality is present on a futures-specific basis (CME [2013a]). For example, there is no implied functionality in the equity index or FX sectors, but there is in the interest rates and agricultural sectors.

The LOB on GLOBEX is a *combination* of two "sub-books"—an explicit order book and an implied order book. The implied book is limited to being up to two levels deep, while the explicit book is limited to being up to 10 levels deep, both on a security-specific basis. In the broadcast exchange data feed, implied and explicit orders can be distinguished. The process by which the two sub-books are combined is based on the premise that the explicit sub-book has higher priority than the implied sub-book. In the case of ES, there is no implied pricing, so the LOB is equal to the explicit sub-book.

The process by which the LOB is constructed has consequences for later analysis of the LOB. The simplest approach to rebuilding the LOB is a purely *deterministic* implementation of exchange rules. On GLOBEX, this rebuild approach results in volumes being aggregated at each price level, or the L2H model. Another possible approach to rebuilding is a *probabilistic* approach whereby the unaggregated volumes are inferred, or the L3 model (Christensen et al. [2013]). By unaggregating volumes, an extra dimension is introduced into the LOB, giving side, class, price level, and size level. The ability to see the detail of individual orders is especially beneficial to market makers. For example, volume allocation from pro-rata match engines is conditional on the relative sizes of the individual orders at a price level, so a liquidity provider can maximize an allocation by knowing the L3 structure (Janecek and Kabrhel [2007]). The hidden volume considered in this article can be thought of as occupying a new dimension, a hidden level, or the L2H model. L2H shows the hidden volume at the price level and the aggregated visible volume, with the exception of the peak of the hidden volume, which is unaggregated and visible.

Alg.1 GLOBEX LOB Rebuild Algorithm. $[LOB,\ dV] = \text{Rebuild(FIX)}$

```
LOB(side, class, priceLevel) ← 0 {Initialize the empty LOB}
dV ← 0 {Initialize the vector. dV = dV_1, ..., dV_T}
for t = 1 : T do
  {Only process orders, not trades}
  if order then
    {Extract price, volume, side, price level and number of orders}
    [P, V, S̃, m, n] = FIX_t
    if (MDUpdateAction == 0) then
      Insert a new price level
      dV_t = V
      for i = m : M − 1 do
        {Shift existing levels down by 1}
        LOB(S̃, 1, i + 1) = LOB(S̃, 1, i)
        LOB(S̃, 2, i + 1) = LOB(S̃, 2, i)
        LOB(S̃, 3, i + 1) = LOB(S̃, 3, i)
      end for
      LOB(S̃, 1, m) = P      {Assign price}
      LOB(S̃, 2, m) = V      {Assign size}
      LOB(S̃, 3, m) = n  {Assign nu. of orders}
    else if (MDUpdateAction == 1) then
      Change an existing price level
      dV_t = V − LOB(S̃, 2, m)
      LOB(S̃, 1, m) = P      {Assign price}
      LOB(S̃, 2, m) = V      {Assign size}
      LOB(S̃, 3, m) = n  {Assign nu. of orders}
    else if (MDUpdateAction == 2) then
      Delete an existing price level
      dV_t = −V
      for i = m : M − 1 do
        {Shift existing levels up by 1}
        LOB(S̃, 1, i) = LOB(S̃, 1, i + 1)
        LOB(S̃, 2, i) = LOB(S̃, 2, i + 1)
        LOB(S̃, 3, i) = LOB(S̃, 3, i + 1)
      end for
      LOB(S̃, 1, M) = 0 {Nullify last price level}
      LOB(S̃, 2, M) = 0
      LOB(S̃, 3, M) = 0
    end if
  end if
end for
```

These three representations of the same LOB are shown in Exhibit 4. The L2 view shows the aggregated volume at the price level, as broadcast by GLOBEX. The L3 view shows the inferred unaggregated volume at the price level. The L2H view shows the aggregated volume with an iceberg order present. In this example, the peak of the iceberg is equal to 25. In all three views the *visible* volume is equal to 50 lots. In the L2 and L3 views the *realizable* volume is equal to 50 lots, however in the L2H view the realizable volume is equal to 175 lots due to the presence of the hidden volume (one visible peak at 25 lots and six hidden peaks at 25 lots). The order at bottom of the bar with S = 25 has time-priority, however in the L2H case, all subsequent tranches of the iceberg order have time priority over the S = 15 order.

HIDDEN LIQUIDITY

There are three reasons why the LOB may not display the "true" liquidity state of the market: *iceberg* orders, *unbroadcast* orders, and *phantom* orders. All these order types cause the *realizable* volume of the LOB to differ from the displayed volume.

Iceberg Orders

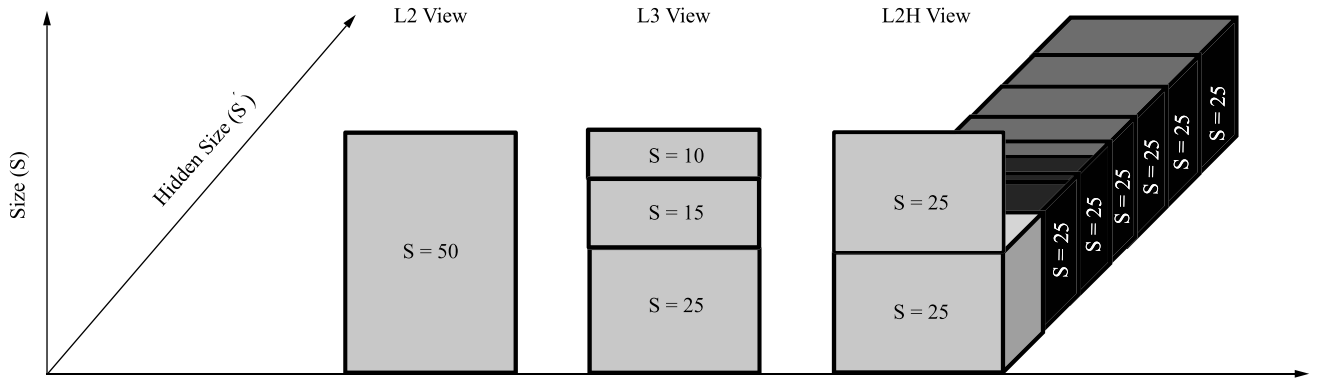
Iceberg orders are an order type supported by GLOBEX. An iceberg order is a special type of limit order, where in addition to a price, side, and size, the user is required to specify a *max show* value. Max show is the upper size limit of the fraction of the total order size that is shown to the market, while the remainder of the order volume remains hidden. When the displayed quantity has been filled, another portion *less than or equal to* the displaced quantity is then displayed, with the time priority of the *initial* peak and the remaining hidden volume reduced by the peak size. This is notably different from a trader constructing his or her own iceberg order system, as a sequential series of limit orders would not have the time priority of the first order. Hence iceberg orders are particularly important for latency, sensitive markets where there is a time component to the matching algorithm, such as the equity index and FX sectors (Chicago Mercantile Exchange [2013b]).

Iceberg orders are a way of limiting information flow, their prime reason for existing being to facilitate large trade execution. This is done by preventing market makers from noticing the large incoming order and changing the price in anticipation of it, thereby reducing the market impact of the trade. However, iceberg orders are controversial. They diminish the benefits of transparent, order-driven markets, including price efficiency, low costs of market monitoring, and less information asymmetries (Madhavan [2000]). If the iceberg allows a trader to avoid the informational disadvantage associated

EXHIBIT 4

Three Views of the LOB, Each Showing One Price Level

In all three views, the *visible* volume is equal to 50 lots. In the L2H view the *Realizable* volume is equal to 175 lots.



with limit orders over market orders, why are icebergs not always used? The answer might be that liquidity suppliers sometimes wish to influence the LOB in some way to their advantage. Approaches such as destabilizing the LOB, "moving the market," and phantom orders are legal gray areas.

GLOBEX refuses to quantify any statistics relating to iceberg orders, but has said they are "popular" and that due to icebergs, the "true liquidity in CME Group markets is generally much superior to displayed liquidity" CME [2011].

Unbroadcast Orders

Some types of implied orders *are* eligible to be filled but are *not* broadcast in the market data feed CME [2013b]. This is significant because it means that there is liquidity in the LOB which cannot be seen, meaning there is the potential for a fill at a price level in the LOB where no order was seen to sit. These unbroadcast orders occur only in the contracts with implied pricing and then only in the first two price levels of the LOB. While the implied book is just two levels deep, it has the potential to have a far greater update rate.

Phantom Orders

A phantom order is one to which a trader is not committed or indeed which a trader does not intend ever to execute (Burghardt et al. [2006]). Phantom orders are generally considered bad for the market, as they give the impression of more liquidity than there actually is. These orders can be hit only by being able to act faster than the trader who placed them, and so are termed "negative liquidity." Phantom orders are characterized by rapid-fire submission followed by cancellation, with quoting rates of up to 20KHz being observed in highly illiquid securities (Hunsader [2010]). Borkovec defines phantom volume as limit orders added and canceled from the LOB in a period of less than two seconds and finds that this is 10% of all orders for equities data from NYSE ARCA in 2005 Borkovec et al. [2012]. GLOBEX limits this behavior by regulating the message-to-volume ratio submitted by a particular trader, where a message includes an order, modification, or cancel. This ratio ranges from 4 (for ES) to 60 (for less liquid futures). The messaging policy is an aggregated average over a 24-hour window, and if a trader exceeds the ratio, he or she is fined (Chicago Mercantile Exchange [2013b]). The reasons for submitting phantom orders are unclear but may include manipulating/destabilizing the LOB or detecting hidden liquidity.

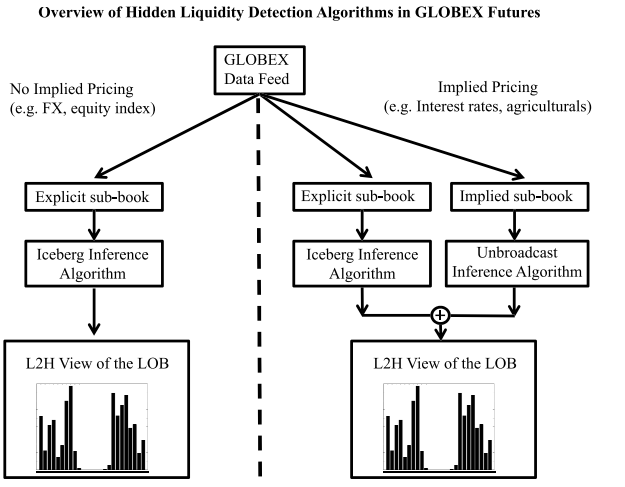
Summary of Hidden Volumes

Phantom orders are not *truly* realizable volume and so are not considered further. Realizable hidden volume on GLOBEX futures takes two forms—iceberg orders and unbroadcast orders. In order to detect all the hidden volume in the LOB of GLOBEX futures, the approach outlined in Exhibit 5 can be applied. As the LOB is built by combining the sub-books of implied and explicit

EXHIBIT 5

Schematic of Prediction Algorithms for GLOBEX Hidden Volume

Conditional on implied pricing, iceberg and unbroadcast algorithms are run independently or in parallel.



orders, the iceberg algorithm can be applied to the explicit sub-book to detect all the hidden liquidity present in futures which do not support implied orders. For futures which do support implied pricing, the iceberg algorithm needs to be run in parallel with an unbroadcast algorithm, which detects the hidden volume in the implied book. In this article we just consider the hidden volume resulting from iceberg orders and plan to publish research on the unbroadcast hidden volume in a subsequent article.

LITERATURE REVIEW OF ICEBERG DETECTION

An extensive literature on detection of iceberg orders exists, which falls into the three categories of active algorithms, model-based algorithms, and frequentist algorithms. These categories of detection algorithm are now reviewed.

Active Algorithms

An active algorithm seeks to detect iceberg orders by "pinging" the LOB with orders that the participant never intends to be filled. In Hasbrouck and Saar [2001] the authors note that hidden orders constitute 3% of all submitted limit orders but account for 12% of all executions, while more than 25% of all limit orders submitted are canceled within two seconds of submission. They propose that these fleeting orders are likely used by aggressive traders searching for hidden orders. For example, a buyer might submit an order priced just short of the ask quote, hoping to trade against any hidden sell orders. In this view, a fleeting limit order represents a liquidity demander, rather than a supplier.

Durbin [2010] suggests detecting icebergs by use of fill or kill (FOK) limit orders. On GLOBEX, FOK are canceled if not immediately filled for the specified minimum quantity at the specified price or better. By submitting small FOK orders over a range of prices, the presence of hidden orders can be detected by whether the order is filled or not.

Model-Based Algorithms

Both De Winne and D'hondt [2007] and Bessembinder et al. [2009] use regression models on 2002–2003 Euronext equities data. This data set allows hidden depth to be directly observed. De Winne finds that more than 45% of the depth at the top five levels of the LOB is hidden and that iceberg order size is six times greater than a normal order. Bessembinder finds that 18% of incoming orders include some hidden size, 44% of order volume is hidden, and hidden orders are more common in illiquid issues and for large trade sizes and when order arrival rates are low.

Hautsch and Huang [2009] build a Bayesian model with a Bernoulli likelihood function using logit multiple regression, where the probability of an order being executed with hidden liquidity can be predicted by eight predictors, including distance from the mid price, the size of the spread, and the lagged return.

Avellaneda et al. [2011] try to match the empirically observed LOB mechanics by including an implied hidden volume term in a stochastic diffusion model. The approach allows the implied hidden liquidity of different securities to be compared. For four securities on three exchanges, the results show differences of more than 220%.

Esser and Mönch [2007] consider the case for exchanges where the iceberg peak loses time priority after each execution and generate a stochastic model for the optimal peak size of an iceberg order. The authors model price by a jump-diffusion process, and when the diffusion process hits an iceberg order, a jump in price occurs, leading to a probabilistic model for the peak size. They conclude that 8% of the LOB volume is hidden.

Fleming and Mizrach [2009] consider U.S. Treasury data from the inter-dealer platform BrokerTec, using a model for the LOB which incorporates hidden volume Moinas [2006]. The authors observe that the percentage of executed hidden volume is low, at 2%, and that this low figure masks the fact that there is usually no hidden depth, but when there is hidden depth, it is substantial. The authors also observe that the pattern of hidden depth differs from that of visible depth, having the largest volume at the first price level for most maturities, while the visible volume is greatest a few levels out.

Frequentist Algorithms

In Burghardt et al. [2006], the authors consider the concept of "sweep to fill," whereby a large trader clears out all volume from the LOB. The authors compare the sweep to fill average price with the observed VWAP market impact and note that the impact of VWAP tends to be smaller than sweep to fill measures would suggest, meaning that the LOB must be more liquid than it seems, with impact factors differing between 4% and 10%, depending on the order size. The authors postulate that this extra liquidity exists in the form of iceberg orders in the LOB.

Borkovec [2012] takes an approach that is based on estimating the true liquidity environment by generating joint probability distributions of intraday volume profiles and various predictors (for example, spread, volatility, and depth). Hidden volumes are found by trying to match trades to quotes. Hidden orders are found to be 53% larger than the visible orders.

Frey and Sandas [2009] develop an empirical frequentist approach for detecting hidden volume using LOB data, which relies on the fact that the peak size is constant and that the time stamp for resupply is the same as the time stamp for the executed volume. On XETRA, time priority is *lost* between successive peaks, so each peak goes to the back of the queue. The authors apply their approach to XETRA, DAX 30 equities data from 2004 and find that iceberg orders make up 9% of all orders and 16% of all executed volume, and that the average number of tranches is five. In terms of size, iceberg orders are on average 12 to 20 times larger than visible limit orders and have a peak size that is 2.5 times larger than visible limit orders.

Summary of Literature Review

From the literature review, we find a lack of algorithms which could be applied online for hidden volume prediction on GLOBEX. The literature review is summarized in Exhibit 6.

PREDICTING ICEBERGS ON GLOBEX

In this section we present our model for detecting and predicting iceberg orders on GLOBEX. In summary, initially an iceberg order is indistinguishable from a limit order, with the same price and size. But over time, the execution of displayed peaks and subsequent display of new peaks allow market participants to learn about its existence and predict its size with increasing accuracy.

EXHIBIT 6
Literature Review on Modeling Iceberg Orders

Reference	Exchange/ Platform	Data	Period	Volume Hidden (%)
Burghardt et al.[2006]	GLOBEX	ES future	Jan–Apr 2006	4-10
Avellaneda et al.[2011]	NASDAQ, NYSE, BATS	4 ETF's/equities	Jan 2010	Relative
Bessembinder et al.[2009]	Euronext Paris	100 equities	Apr 2003	44
Esser and Mönch [2007]	XETRA	1 equity	Jan–Mar 2002	8
Moro et al. [2009] Vaglica et al.[2008]	SETS, SIBE	97 equities	Jan 2001–Dec 2004	52
Tuttle [2005]	NASDAQ	97 equities	4 weeks 2001, 2002	25
De Winne and D'hondt [2007]	Euronext Paris	CAC 40, equities	Oct–Dec 2002	45
Borkovec et al.[2012]	NYSE ARCA	329 equities	Jun–Aug 2005	3
Frey and Sandas [2009]	XETRA	DAX 30, equities	Jan–Mar 2004	16
Fleming and Mizrach [2009]	ICAP BrokerTec	US Treasuries	Jan 2001–Feb 2006	2
Hasbrouck and Saar [2001]	Island ECN	300 equities	Oct–Dec 1999	12
Hautsch and Huang [2009]	NASDAQ	7 equities	Jan 2009	14
Yao [2012]	NASDAQ	2,390 equities	Jan 2010–Nov 2011	16

Iceberg Mechanics on GLOBEX

The mechanics of how GLOBEX icebergs operate are now presented. On GLOBEX, when a trader enters an iceberg order, he or she is required to specify side $\widetilde{S}$, limit price P, the total order size V, and max show Ψ (a fraction of the total size).[4] A limit price means that the order must be filled at a price *at least as good* as the specified price. In the data feed, this means that an aggressor order that fills a resting limit order need *not* show a trade price equal to the resting limit order price. Depending on the side of the LOB, there will be a greater (less) than or equal to condition $\overset{>}{\underset{<}{\equiv}}$. When V is exactly divisible by Ψ, the size of the iceberg peak ψ is equal to the max show size. When V is not exactly divisible by Ψ, the final tranche is equal to the remainder. When the iceberg order enters the LOB, it displays only a portion of the order to the marketplace (the peak), while the rest of the iceberg remains hidden. When the displayed quantity has been filled, another peak is then displayed, with the time priority of the *initial* peak and the remaining hidden volume reduced by the peak size. While GLOBEX *retains* time priority between successive peaks, this is not the case for all exchanges; for example, XETRA loses time priority between peaks. Due to this conservation of time priority, icebergs are most popular on markets that have a time component in their match algorithm (such as the FIFO algorithm used by ES), as opposed to markets that do not have a time component (for example, the pro-rata algorithm used by eurodollar). When pro-rata matching does occur, the match algorithm considers only ψ and not V in making its allocation. Iceberg order execution is shown in Algorithm 2.

Alg. 2 GLOBEX Iceberg Execution Algorithm

```
1:  Inputs(V, Ψ)        {Total order size, max show}
2:  N = ⌈V/Ψ⌉                    {Number of tranches}
3:  for n = 1 to N do
4:     if (n == N) then
5:        ψ = V              {Assign the peak size}
6:     else
7:        ψ = Ψ
8:     end if
9:     v_n = ψ              {Place the order into the LOB}
10:    while v_n > 0 do
11:       v_n ← trade     {Match volume from trade(s)}
12:    end while
13:    V = V − ψ {Reduce the hidden volume by the
       peak}
14: end for
```

In the broadcast data feed, there is no *simple* way to distinguish an iceberg order from any other order. The only way to know if an order is part of an iceberg is if you had placed the order yourself or had access to private exchange information. However, a key feature of the data structure allows us to *infer* the presence of iceberg orders in the broadcast data feed. This feature is that LOB update messages are broadcast *post* trades happening, albeit with a time lag due to system latency.

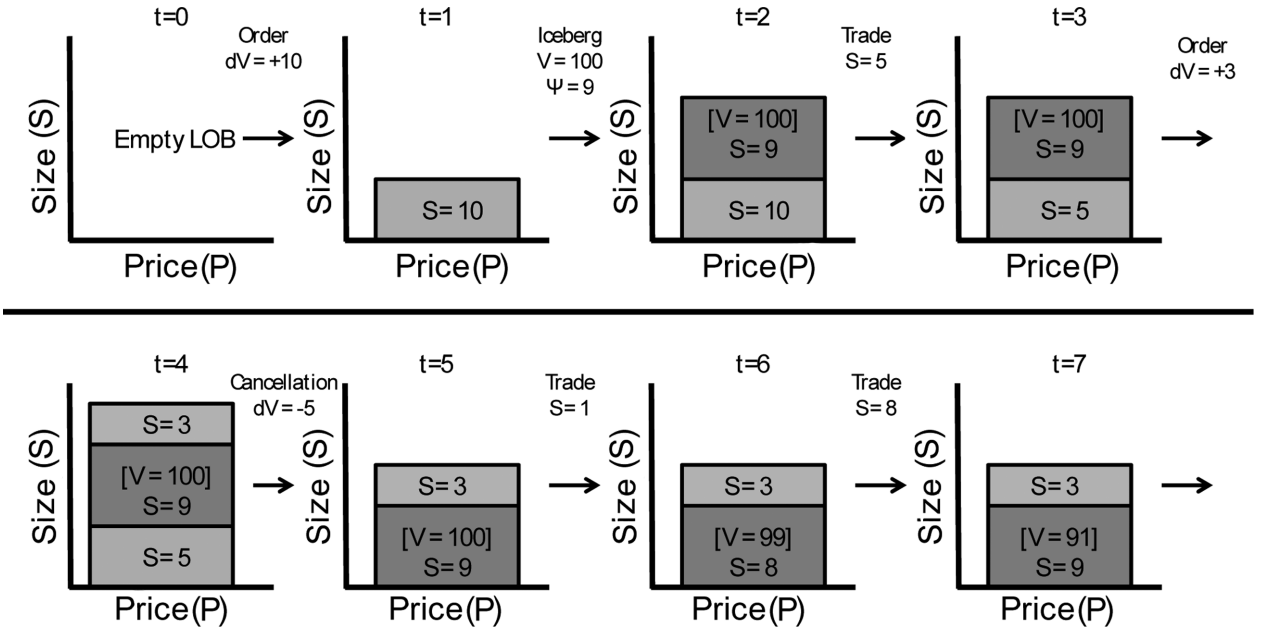
The mechanics associated with broadcast messages at the point of iceberg order execution are at the center of this article. In the simplest case, when the peak of an iceberg order is filled by a trade, three messages are seen in the data feed: first, a trade message, with associated size; second, an LOB update message, from which the decrease in volume dV can be inferred; third, an LOB update message replenishing the peak.

These mechanics are now illustrated with an example, as shown in Exhibit 7. An iceberg order is specified with a total size $V = 100$ and a max show $\Psi = 9$. The first 11 tranches of the iceberg are of size $\psi = 9$ and the final tranche is size $\overline{\psi} = 1$, so $N = 12$ (where N is the number of tranches). At time step $t = 6$, a trade message for $S = 8$ is seen in the data feed. This results in two further messages being sent by GLOBEX, firstly, a LOB update message $dV = -8$, and secondly a peak replenish message $dV = 9$. The first of these messages is the trade volume being removed from the LOB. The second of these messages is the next tranche of the iceberg order being placed into the LOB. It this peak replenish message time dt after a trade which allows the iceberg order to be detected. The rules associated with what messages are broadcast become more complex than in Exhibit 7 when the trade size is greater than Ψ. In summary, in the event of a large trade, not all the peak replenish messages are broadcast. For example, if there is an iceberg order to buy ($V = 100$, $\Psi = 10$) currently showing $\psi = 10$, and there is a sell side aggressor order of size 30, there will be three trade messages (3×10 lots), but *no* peak replenish message, because the displayed peak quantity will remain the same, $\psi = 10$. In a second example, if there is an iceberg order to buy ($V = 100$; $\Psi = 10$) currently showing $\psi = 10$, and there is a sell side aggressor order of size 32, there will be four trade messages (3×10 lots, 1×2 lots), and a peak replenish message of size 8, which is the displayed peak quantity

EXHIBIT 7

Schematic of Iceberg Order Mechanics.

A simplified LOB consisting of just one price level is shown progressing through time. At each step a FIX message is applied to the LOB. The bottom of stack has the highest time priority. Normal limit orders are shown in light greys, iceberg order in dark grey. Square brackets around *V* mean the value cannot be seen in the broadcast data.



ψ = 8. These two examples can be explained by the fact that the GLOBEX incremental LOB management rules send out update messages only when the external characteristics of the LOB change. This means that the presence of iceberg orders can result in the LOB volume not changing after trades have been executed.[5]

As for any limit order, iceberg orders can be modified after submission. If Ψ is modified by being increased in size after being submitted, then the order currently shown retains the old max show, and once that has been filled, thereafter shows the new value, while the time priority of the iceberg is maintained. However, if the modification is a decrease in Ψ, then the iceberg order loses its time priority.

Trade Size Descriptive Statistics

The variations in the message broadcast rules mean that our approach needs to be conditional on trade size. In Exhibit 8, descriptive statistics for trade size in the front month ES contract for the period January 1, 2011, through December 31, 2011, are presented. For this period, the mean number of trades per day was 0.5 million, while the mean volume per day was 2.2 million lots. Trade size approximately follows an exponential distribution, with most trades being small and a few being very large. This prior distribution of trade sizes leads to a way of classifying a trade as "normal" or "large," based on the 99th percentile. If an order at time $t-1$ meets the conditions to be a viable iceberg order, and at time t a large trade occurs, then the large trade may lead to more complex iceberg mechanics.

EXHIBIT 8

ES Trade Size Statistics (lots)

Statistic	Value	Statistic	Value
Mean	5	Maximum	1,069
Mode	1	95th Percentile	21
Median	2	97.5th Percentile	34
Minimum	1	99th Percentile	53

Prediction Algorithm Overview

Our algorithm has two phases, a learning phase and an online inference phase. While the learning phase

could be online, we set it to be offline for reasons of simplicity and computational latency.

1. Offline learning. The algorithm carries out a forward pass to identify likely icebergs. The resulting identified orders are then used to generate the distributional parameter $\boldsymbol{\Theta}$. This is done ex post, using the previous H days of historical data.
2. Online inference. This phase uses the output $\boldsymbol{\Theta}$ from the learning phase during a forward pass. The output $\boldsymbol{\Gamma}$ is an online estimate for the existence of icebergs.

The forward pass algorithm is so called as it proceeds forwards in time.

Variables Used in the Algorithm

In this section, the variables used in the algorithm are defined. The latent variables in the problem are max show Ψ and total order size V. The set of K icebergs is denoted by $\boldsymbol{\Phi} = \{\Phi_1, \ldots, \Phi_k, \ldots, \Phi_K\}$, where K is the total number of iceberg orders seen in one trading day. Φ_k denotes a single iceberg order such that $\Phi_k = \{\phi_{k,1}, \ldots, \phi_{k,n}, \ldots, \phi_{k,N}\}$. N is the number of tranches within a single iceberg order. The variables are summarized in Exhibit 9.

The inference stage outputs the variables shown in Exhibit 10. The variable $\boldsymbol{\Gamma}$ dynamically updates over time and can be examined at any point in time to see the parameters of the estimated icebergs in the LOB. $\boldsymbol{\Gamma}$ is the final output of the algorithm.

Detection Algorithm

In this section, the logic behind how the algorithm works is described. The algorithm applies a pattern recognition approach which seeks out a certain combination of events that allows an iceberg order to be identified and tracked, and thus predicted.

The only way an iceberg order can be distinguished from a normal limit order is when a *trade* causes the peak

EXHIBIT 9
Learning Phase Variables

Variable	Description
k	The number of iceberg orders seen in one trading day for a given security. $k = \{1, \ldots, K\}$.
Φ_k	Unique identifier for this iceberg order. $\Phi_k = \{\tilde{S}, P, \Psi, \alpha\}$.
$\tilde{S}$	The side of the LOB. Is the iceberg a buy or sell order?
m	Price level in the LOB. $m = 1, \ldots, M$.
P	Limit price of the iceberg order.
α	Is the iceberg currently active or not? 0=false, 1=true.
ω	If the iceberg is inactive, was it cancelled or filled? 0=cancelled, 1=filled.
Ψ	Max show. The upper size limit of the iceberg peak.
$\phi_{k,n}$	The n^{th} tranche of iceberg order Φ_k. $\phi_{k,n} = \{\psi, t, \lambda, \theta\}$.
n	The number of tranches of an iceberg order. $n = \{1, \ldots, N\}$.
ψ	Peak size of tranche. The final tranche peak size is a special case and denoted by $\bar{\psi}$.
t	Time of tranche (HH:MM:SS.FFF). $t = 1, \ldots, T$.
λ	Time between the trade and the publication of the refresh message. In milliseconds.
θ	Time between tranches. $\theta = \phi_{k,n}(t) - \phi_{k,n-1}(t)$. In milliseconds.

EXHIBIT 10
Inference Phase Outputs

Variable	Description
$\hat{V}_k$	The estimated total size of the k^{th} iceberg (in lots).
$\hat{\Psi}_k$	The set of peak sizes of the k^{th} iceberg (in lots). $\hat{\Psi} = \{\hat{\psi}_1, \ldots, \hat{\psi}_n, \ldots, \hat{\bar{\psi}}_N\}$.
$\boldsymbol{\Gamma}$	The set of the inference phase outputs. $\boldsymbol{\Gamma} = \{\hat{V}, \hat{\Psi}\}$.

of an iceberg order to be filled, and then GLOBEX automatically refreshes the peak in the LOB. This automatic refresh is indistinguishable from a new order submission. The *initial* detection signal is defined as an order placed after a trade that satisfies a list of conditions:

- The order must be submitted within *dt* of the trade occurring, due to internal latency of the GLOBEX system. *dt* is set equal to 300 milliseconds. If *dt* is too large, it is likely that non-iceberg orders will fall inside the window. If too small, iceberg orders may be missed.
- The trade price must satisfy the limit order price.
- The change in order size must be positive (i.e., not a size reducing modification or a cancellation).
- An implied order can never be an iceberg.
- As the trade volume must have been taken from the first price level $m = 1$, the order must add volume back to the first price level.
- The side of the trade must be equal to the side of the order update. For example, when the aggressor trade is a buy, the trade volume comes out of the ask side of the LOB.

The inference is that an order that meets these criteria might be the peak of an iceberg. This methodology cannot detect iceberg orders in the depth levels, as by definition trades occur only at the first price level $m = 1$. An exception to this is when an iceberg has been partially executed and the price moves away, leaving the iceberg in a depth position in the LOB.

The next step of the algorithm is to check whether this is the first tranche of a new iceberg, or an additional tranche of an existing iceberg. This is complicated by the ability of the refresh message to change in size, conditional on the trade size. Four cases can occur:

- If the trade size is less than Ψ, the refresh order size must be greater than or equal to the trade size.
- If the trade size is equal to Ψ, or multiples thereof, then no refresh order is seen.
- If the trade size is greater than Ψ, then the refresh order size is equal to the modulus after division. If a large trade occurs *and* the refresh message is the first peak $\phi_{k,1}$ seen for that iceberg, then a special case occurs. It is assumed that the iceberg order peak refresh message is not equal to the max show $\psi_1 \neq \Psi$, as would be the case for a normal trade size, as the large trade size will be greater than the max show. Instead, it is assumed that $\psi_1 \leq \Psi$. This is incorporated by not setting the max show for the iceberg equal to the first peak seen.
- The final tranche of the iceberg order is a special case, as it may be less than or equal to the max show, $\bar{\psi} \leq \Psi$. This occurs when V is not exactly divisible by Ψ.

The binary indicator variable α denotes whether an iceberg is currently active by tracking the cumulative volume traded at each specific side/price (i.e. Φ_k) since the last tranche was seen. If the cumulative volume at Φ_k exceeds the max show (plus *dt* to allow for system latency) and no refresh message has appeared, $\alpha = 0$. This allows the algorithm to detect multiple iceberg orders with the same price/side/max show combinations. The approach is made possible by the fact that on GLOBEX, time priority is retained between tranches. If the activity test comes back with $\alpha = 0$, there are three possibilities:

- The suspected iceberg was not really an iceberg to begin with (a false positive). This is defined as: if when $\alpha = 0$, $\phi_{k,n}$ has $n = 1$, then Φ_k is removed from $\mathbf{\Phi}$, i.e., if only one tranche of a possible iceberg order was seen and then the iceberg was not seen again, we say it was coincidence that an order was submitted in time window *dt* that met assorted criteria. If Φ_k was not deleted from $\mathbf{\Phi}$, then the parameter set would be distorted by the presence of non-iceberg limit orders.
- The iceberg order was canceled. As with any limit order, cancellation of an iceberg order can occur at any time before being fully filled. The cancellation message is *seen* to apply to just the displayed tranche, $|dV| \leq \Psi$; however, it actually *applies* to the remaining volume of the iceberg. Search the time between $\phi_{k,n}$ and the cumulative volume exceeding Ψ at the price/side of interest. Set $\omega = 0$.
- The iceberg order was filled in its entirety. If neither of the other possibilities are true, this is taken to be the case. Set $\omega = 1$.

Initial detection of an iceberg order occurs by first looking for an order within a window *dt* and with certain size constraints. The subsequent tranches are further constrained by P, $\widetilde{S}$, and $\mathbf{\Phi}$; however, there is still the

possibility that n sequential orders could have been seen which meet the classification requirements. The probability that $\phi_{k,1}$ is not really an iceberg order is reasonably high, while the probability that $\phi_{k,2}$ is not really an iceberg order is lower, and so on. The probability of sequential false detections is *mutually independent*. This compounding of probabilities is shown in Equation (1)

$$p(\Phi_k) = 1 - \left(\prod_n^N p(\phi_{k,n}) \right) \approx \left\{ 1 - e^{-(\beta \times n)} \right\}, \ \beta > 0 \quad (1)$$

The detection algorithm assumes that for Φ_k, once $n > 1$, the order is an iceberg. Hence, our criteria of $n > 1$ is an approximation. This is shown graphically in Exhibit 11. Subplot one shows the probability of detection conditional on the number of tranches seen, according to Equation (1). Subplot two shows this probability according to Algorithm 6. Both examples use an iceberg with ten tranches.

A summary of the forward pass is shown in Algorithm 6. This algorithm is run in offline (learning) and online (inference) modes. The variable "data" is the data structure from Exhibit 3, and dV is the change in volume output by Algorithm 1. Only for online mode is $\boldsymbol{\Theta}$ required.

We are now at the stage where we have generated $\boldsymbol{\Phi}$. In the next two sections, the details of generating $\boldsymbol{\Theta}$ from learning and $\boldsymbol{\Gamma}$ from inference are presented.

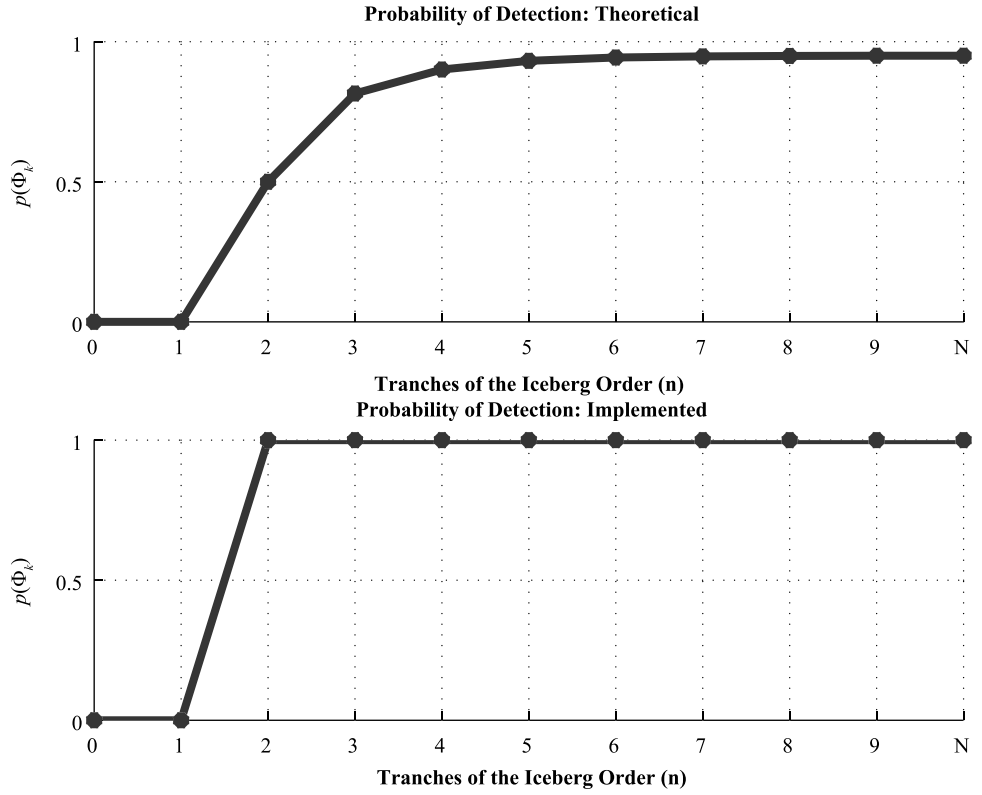
Learning Phase: Generating Θ

In this section, we describe the process of generating $\boldsymbol{\Theta}$ from $\boldsymbol{\Phi}$. Our main interest in iceberg orders is being able to *predict* the existence of hidden volume. To do this, information on the prior behavior of iceberg orders is used. It is hypothesized that participants

EXHIBIT 11

Graphical Representation of the Probability of Detection

$p(\Phi_k)$ is the probability that the order Φ_k is an iceberg. $p(\Phi_k) = 1$ [0] means it is [is not] an iceberg.



who submit icebergs do not do so randomly, but display repetitive behavior when they submit these orders, particularly in terms of the ratio V/Ψ. The learning phase captures these statistically significant relationships from historical data, allowing prediction of the remaining iceberg order once it has been detected.

In order to carry out learning, we have to solve a *multidimensional mass estimation* problem (Hastie et al. [2001]). In our problem, the dimensions are max show Ψ and total order size V. As both V and Ψ are discrete integers (as the minimum size that can be traded is one lot), we are interested in finding the bivariate *probability mass function*. A probability mass function is a function that describes a *discrete* probability distribution. Estimating probability mass functions with discrete variables can be straightforward. As there is only a finite number of values, the simple relative frequency of occurrence can be found. For the case of a bivariate joint distribution, a separate density could be found for each value of the first variable, while holding the second variable constant. However, this approach is practically awkward if the number of levels for the discrete variable is large compared to the number of samples. Moreover, the joint distribution problem has us estimating completely separate distributions for the second variable for every value of the first variable, without any sharing of information between them. A better solution is to *smooth* those distributions toward each other, allowing for interpolation over sparse (missing) data, while taking into account any nearby observations, or lack of them. Many methodologies exist that allow for density estimation including splines, wavelets, Fourier series, and parametric approaches. We opt to use nonparametric kernel estimation Scott [1992]. While kernels for discrete variables do exist, for example Aitchison and Aitken [1976]; Rajagopalan and Lall [1995], we use the *Gaussian kernel* on the basis of simplicity, and then, at the point of sampling the distribution, round the output to the nearest integer Haddad and Akansu [1991]. Carrying out this estimation on a large data set is computationally demanding, though as this is done off-line it does not affect trading.

Learning is carried out before the start of each trading day, using the previous H days worth of data, excluding any icebergs Φ_k that were not filled $\omega = 0$. The learning phase is summarized in Algorithm 3. Bivariate Gaussian kernel density estimation is shown in Algorithm 4, where n is the number of observations in each vector and α is a Gaussian confidence limit. The output distribution Θ is shown in Exhibit 12.

Alg. 3 Iceberg Off-line Learning Algorithm. Θ = offline_learning(Φ)

```
[V, Ψ] ← 0 {Initialize empty vector}
for h = 1 : H do
    Consider iceberg orders from past H days
    for k = 1 : K do
        Consider the K iceberg orders that day
        if Φ_k(ω) == 1 then
            Consider iceberg orders which were filled
            [V, Ψ] = extract(Φ_k)
        end if
    end for
end for
Θ = KDE(V, Ψ)
```

Alg. 4 Bivariate Gaussian Kernel Density Estimation. $f(x, y)$ = KDE(x, y)

$$\sigma_x = \frac{1}{\alpha n^{\frac{1}{6}}}\text{median}\,(|x - \hat{x}|)$$

$$\sigma_y = \frac{1}{\alpha n^{\frac{1}{6}}}\text{median}\,(|y - \hat{y}|)$$

$$\hat{f}(x, y) = \frac{1}{n}\sum \mathcal{N}(x; \mu_x, \sigma_x^2)\mathcal{N}(y; \mu_y, \sigma_y^2)$$

Inference Phase: Generating Γ

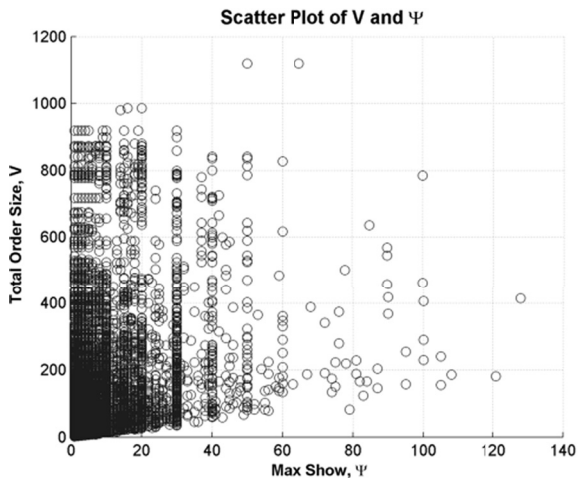
In this section, we describe the process of evaluating Θ to give the prediction Γ. For each iceberg Φ_k in the LOB, an update rule is used to update the estimated remaining volume, given the volume seen so far. This is done by a five-step process:

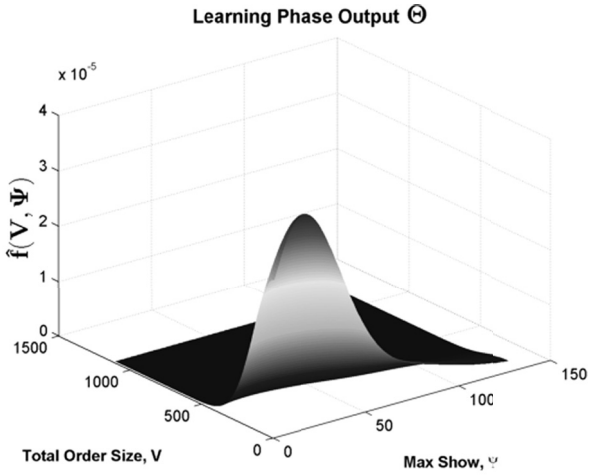
1. We wish to evaluate Θ for $\Phi_k(\Psi)$, the marginal distribution. This has the effect of reducing the dimensionality of the distribution by 1, such that $f = \mathcal{G}(V)$ where $\mathcal{G}$ is some non-linear function that captures the relationship.
2. We are interested in finding the most likely value for $\hat{V}$ for this order, given a known max show Ψ and the knowledge that some of the volume of the iceberg order has already been filled. The filled volume is denoted by $V_{0:n}$ and the unfilled volume by $V_{n+1:N}$, so $V = V_{0:n} + V_{n+1:N}$. Hence the search space is constrained in this manner.

EXHIBIT 12

The Output from the Learning Phase, Θ from One Trading Day

It is this distribution which is evaluated during the inference phase. The distribution is generated by nonparametric kernel mass estimation, which acts to smooth the data set and separate the latent signal from the background noise. Data shown is for ES, January 10–15, 2011.





3. $\hat{V}_k$ is found by maximizing the distribution $p(\hat{V} \mid \Psi, V_{0:n})$ which is equal to maximizing the function $f = \mathcal{G}(V_{n+1:N})$.
4. Knowing $\hat{V}$ allows us to calculate the volume of the subsequent tranches, such that $\hat{V}_k = \sum_{n=1}^{N-1} \Psi_n + \overline{\Psi}_N$, giving $\hat{\Psi}_k$, such that $\mathbf{\Gamma} = \{\hat{V}, \hat{\Psi}\}$.
5. The forecast is updated on one of two events:

   - every time a new tranche is received, $V_{0:n}$ changes, and so the maximization is reevaluated, or
   - if $\Phi_k(\alpha)$ changes to become inactive.

This is a discrete empirical probabilistic update scheme, where the prior distribution is estimated from the data. The conditional probability is updated in light of the volume filled so far $\mathcal{G}(V_{n+1:N})$. The scheme is summarized in Algorithm 5 An example output shown in Exhibit 13. In this example Ψ = 10, $V_{1:n}$ = 20, so only the distribution ≥ 21 is searched and volume which has already been executed is excluded from the search. The result of the search is $\hat{V}_k$ = 52, giving $\hat{\Psi}_k = \{\Psi_{1,\ldots 5} = 10, \overline{\Psi}_6 = 2\}$.

COMPUTATIONAL IMPLEMENTATION

A full discussion of the details of the computational implementation is beyond the scope of this article; however, we wish to give the reader a flavor of the issues involved. The CME matching engines are physically located at the CME data center (the "colo") in Aurora, Illinois.[6] At this data center, market participants can buy "rack space" to connect to GLOBEX via GLink, the order routing interface. What hardware is inserted into the rack space is up to the individual participant.

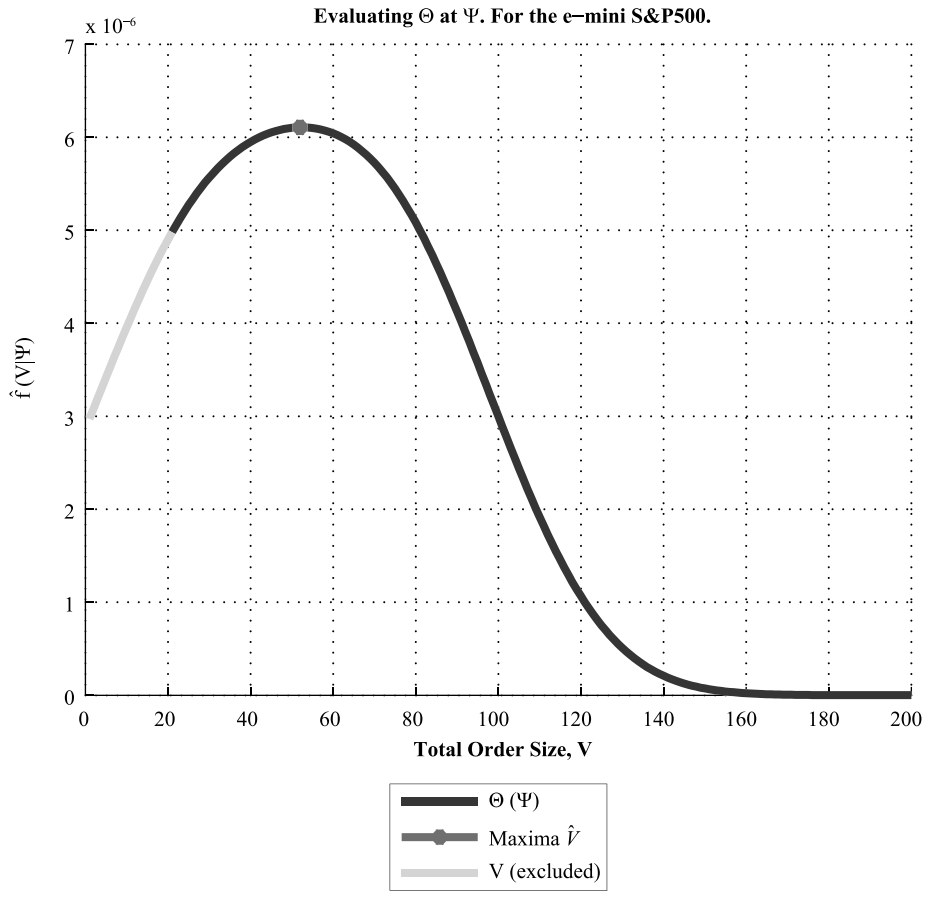
Computation of the algorithm can be split into two distinct phases: offline learning and online inference. Offline learning does not need to happen at the colo, where resources are expensive, but can take place at any server farm anywhere in the world. The learning phase is run as a daily batch process, and each day before the markets open, the latest parameter set is uploaded to the colo servers for use during inference.

Alg. 5 Iceberg On-line Inference Algorithm. $\mathbf{\Gamma}$ = online_inference($\mathbf{\Phi}, \mathbf{\Theta}$)

1: **for** $k = 1 : K$ **do**
2: $\mathcal{G} = \mathbf{\Theta}(\Psi)$ {Evaluate $\mathbf{\Theta}$ for $\Phi_k(\Psi)$}
3: $\hat{V} = max\,(\mathcal{G}(V_{n+1:N}))$ {Maximize constrained $\mathcal{G}$}
4: $\mathbf{\Gamma}_k = \left\{\hat{V}, \hat{\Psi}\right\}$
5: **end for**

EXHIBIT 13

Inference by Evaluating the Distribution Θ. The Distribution is Reduced in Dimensionality to 2D as Ψ Is Known. $\hat{V}$ Is then Found by Maximizing the Constrained Distribution.



Data Storage

Before any learning can take place, the raw FIX/FAST data must be processed, as per Exhibit 3, and added to a data depository. On GLOBEX, we estimate that there are approximately 70 futures over seven asset classes which trade on average once a minute and can be deemed "liquid." The data storage requirements from these are large but manageable, with the unprocessed, compressed data for the 70 futures at 12 GB per day and the processed uncompressed data at 36 GB per day. A cost-efficient way of managing such data is a cloud-based solution, which includes backups: for example, the Amazon Simple Storage Service (S3) file system.[7]

Learning

Once data processing is done, learning is then carried out in batch mode using the last H days worth of data to generate the following day's parameters. For each futures contract, the parameter $\boldsymbol{\Theta}$ is learned. To run the learning phase on these 70 futures within a 24-hour window, we estimate needing a server farm consisting of 25 nodes, each node being a quad core machine with 16GB RAM. Again, a cloud solution allows a high degree of flexibility with low costs: for example, the Amazon Elastic Compute Cloud (EC2).

Batch learning requires the use of a parallelized distributed framework, such as Condor. Condor is open-source software which optimally allocates CPU from

the cluster of worker nodes to process jobs as and when the nodes become free (Thain et al. [2005]). While, in theory, learning could be done with MATLAB, MATLAB would need to be licensed on every worker node in the cluster, and so for this reason Java executables are suggested.

Inference

The online inference code is run by hardware sitting in the rack space at the colo and is latency sensitive. Even during peak market events (approximately 15,000 orders/second), GLOBEX market data is disseminated *externally* within 15 milliseconds of being generated. This market data then takes 5 microseconds from the GLOBEX server to a colo rack, while a subsequent order then takes 5 microseconds from a colo rack to the FALCON match engine. The inference software must be quick enough to operate in this time space if exploitation of detected icebergs is to be achieved.

A Java-based architecture running on Unix blade servers (one per asset class) is suggested to carry out the online LOB rebuilds and subsequent inference. One possible development solution would be to use *content-addressable memory* (CAM). CAM is a special type of computer memory used in certain very high-speed searching applications. JavaSpaces are a Java implementation of this memory paradigm for parallel computing. JavaSpace is shared memory, which, in addition to simple object caching, can use a "master-worker" software pattern, where the master hands out jobs to generic workers Setzkorn and Paton [2004]. This pattern has analogies with the distributed Condor framework used in the learning phase; however, rather than being across different machines as in the case of Condor, we are suggesting implementing JavaSpaces on a single server with multiple cores. This framework negates the need for complex scheduling algorithms, while at the same time achieving low latency. The hidden order problem is particularly well suited to such parallelization, as the individual LOBs are independent and can each be treated as a Java object.

Summary of Implementation

The barriers to entry for this type of algorithmic trading are high: specifically, costs associated with rack space, server farms, hardware, and the specialized development expertise required to implement the research. A schematic of the implementation is shown in Exhibit 14. The schematic shows one physical location, the CME colo data center in Aurora, IL, where the GLOBEX matching engines are based and the inference phase of our algorithm occurs and two cloud locations, where data storage, processing and subsequent learning occurs. The server farm is shown as being Amazon EC2 and the persistent storage as Amazon S3, both popular solutions with financial developers. Learning at the server farm takes place on a daily basis and then pretrading the latest set of parameters are uploaded to the colo for the inference algorithm to use during that trading day.

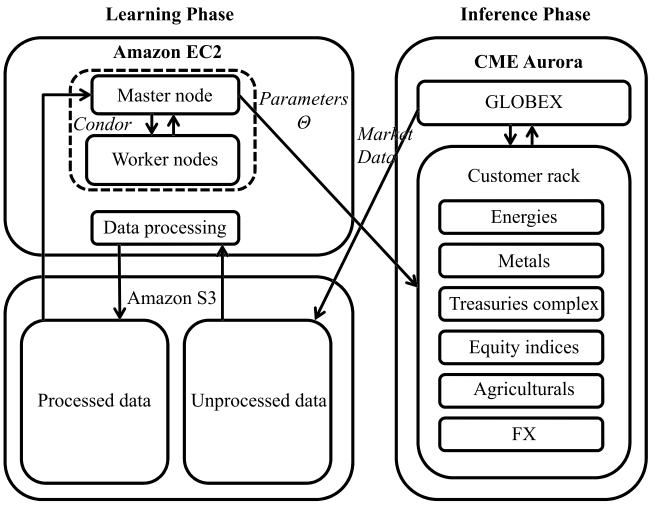
SIMULATION AND RESULTS

As the values of the latent variables V and Ψ are never known, the *true* performance of the algorithm cannot be directly tested. There are two exceptions to this, however:

1. Using a generative model to create synthetic data where some of the orders are iceberg orders.
2. Testing in a "live" setting by submitting iceberg orders using a small amount of capital, and then immediately closing the positions.

EXHIBIT 14

Computational Implementation of the Hidden Liquidity Detection Algorithm



Due to space and monetary constraints, neither of these approaches is implemented. The algorithm is evaluated on real data in the following section.

Real Data

The results from running our algorithm on the ES future are now presented. While the ES future does have strategy contracts, these are limited to calender spreads, and quoting and trading activity in them is extremely limited, to the extent that they can be ignored. Nearly 100% of activity is in the front month outright contract, so only these orders are considered. The mean number of limit orders per day over the period from January 1, 2011, to December 31, 2011, was 4.4 million. The mean number of iceberg orders per day, $\overline{K}$, was seen to be 0.14 million, meaning that 3.2% of all limit orders are iceberg orders.

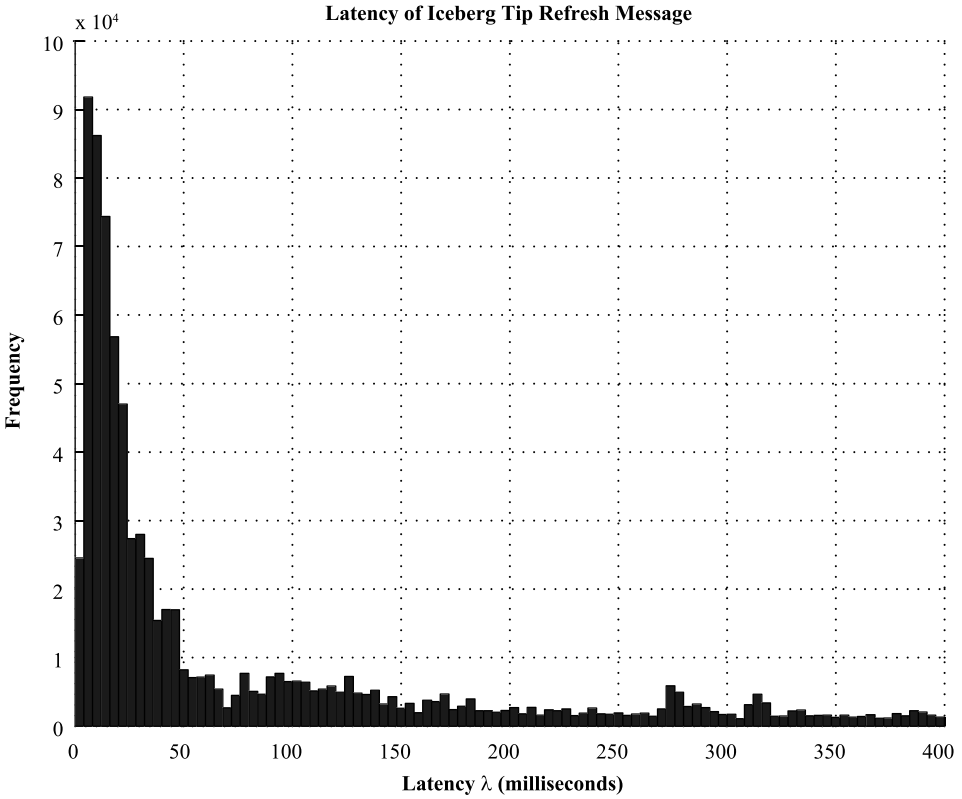
From the iceberg orders detected by Algorithm 6, the latency of inserting the iceberg refresh messages can be inspected. The insertion time delay is due to computational latency. The latency is not constant due to the varying load in the trading engine at any point in time. By setting $dt = \overline{\lambda} + 3\sigma_\lambda$ in the algorithm, unnecessary searching for refresh messages can be minimized. The distribution of this latency is shown in Exhibit 15. The data shows the refresh messages are broadcast according to an exponential distribution with a mean μ_λ of 76 milliseconds and standard deviation σ_λ of 98 milliseconds. This allows us to set dt = 369 in Algorithm 6 so that orders are searched up to 369 milliseconds after trade messages.

The joint distribution of total order volumes and peak sizes is shown in Exhibit 12. Related to this distribution is the number of tranches of iceberg orders $N = \hat{V}/\Psi$, as shown in Exhibit 16. It is noteworthy that

EXHIBIT 15

The Distribution of the Latency of Iceberg Refresh Messages λ

When a trade fills the tranche of an iceberg order, a new tranche is inserted into the LOB by GLOBEX.



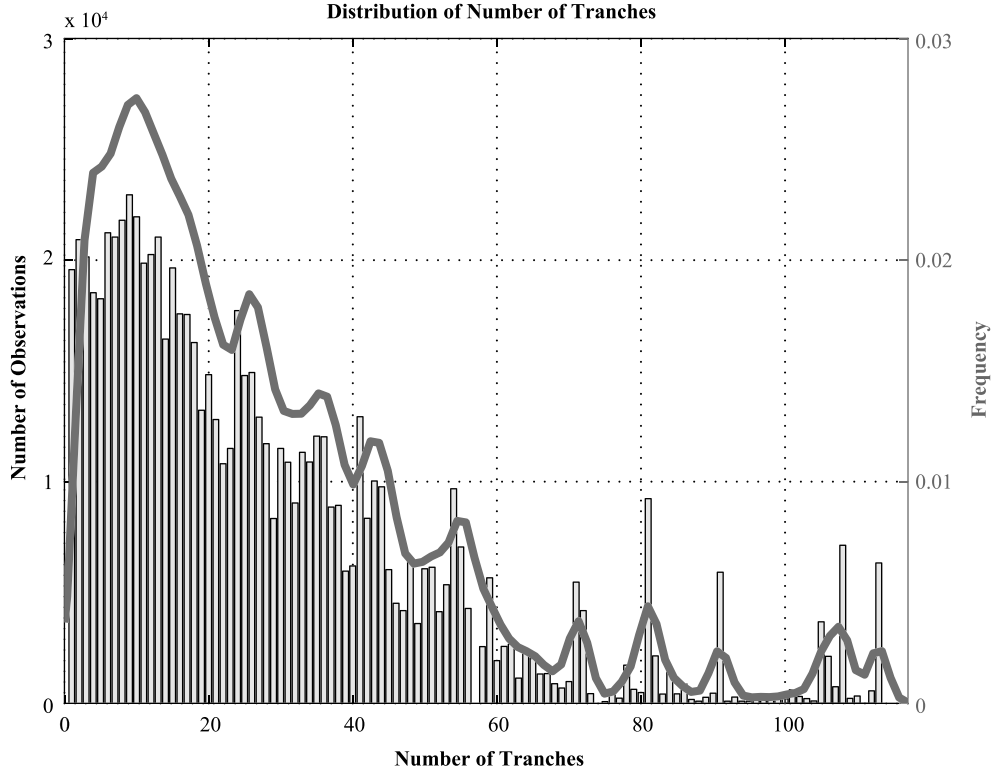
Alg.6 Iceberg Detection Forward Algorithm. $[\mathbf{\Gamma}, \mathbf{\Theta}] = \text{IDFA}(data, dV, dt, \mathbf{\Theta})$

for $t = 1 : T$ **do**
  if trade **then**
    $[tPrice, tSide, tSize] = trade_t$ {Extract trade price, side and size}
    for $i = t : t + dt$ **do**
      if order **then**
        $[P, \tilde{S}, m] = order_i$ {Extract order price, side and price level}
        if ($\tilde{S}$ == ask) **then**
          $pCond = P \leq tPrice$ {The limit price on the ask. The lowest price at which a seller is willing to sell.}
        else if ($\tilde{S}$ == bid) **then**
          $pCond = tPrice \leq P$
        end if
        if $\Big((m == 1)$ & $(pCond)$ & $(\tilde{S} == tSide)$ & $(dV_t > 0)$ & $(isImplied \neq true)\Big)$ **then**
          $\psi = dV_t$
          if Φ_k(largeTrade) **then**
            if $\psi > \Phi_k(\Psi)$ **then**
              $\Phi_k(\Psi) = \psi$
              Φ_k(largeTrade) = 0
              $sizeCond = 1$
            else
              if $tSize > largeOrder$ **then**
                $sizeCond = 1$
              else
                $sizeCond = 0$
              end if
            end if
          else
            if $tSize == 1$ **then**
              $expectedRefresh = \Phi_k(\Psi)$
              $sizeCond = (expectedRefresh == \psi)$
            else
              if $tSize \geq \psi$ **then**
                $expectedRefresh = rem(tSize, \Phi_k(\Psi))$
              else if $tSize < \psi$ **then**
                $expectedRefresh = \Phi_k(\Psi) - tSize$
              end if
              $sizeCond = (expectedRefresh \leq \psi)$
            end if
          end if
          if $sizeCond$ **then**
            Φ_k = testIfActive(Φ_k)
            if ((exists(Φ_k)) & ($\Phi_k(\alpha) == 1$)) **then**
              {Φ_k exists and is active}
              $\lambda = i - t$
              $\theta = i - \phi_{k,n}(t)$
              $\phi_{k,n+1} = \{\psi, t, \lambda, \theta\}$ {Add a new tranche to an existing iceberg order}
            else if ((exists(Φ_k)) & ($\Phi_k(\alpha)$ =!1)) **then**
              {Φ_k might have existed and is no longer active.}
              {How many tranches of Φ_k have been seen so far?}
              if ($n == 1$) **then**
                {Φ_k turned out not to be an iceberg order. Delete it.}
                $\Phi_k = [\,]$
              else if ($n > 1$) **then**
                {Φ_k existed and is no longer active.}
                {Search for cancelled orders at $t > \phi_{n,k}(t)$ which match Φ_k parameters.}
                if (Cancelled) **then**
                  $\Phi_k(\omega) = 0$
                else
                  {Φ_k was filled}
                  $\Phi_k(\omega) = 1$
                end if
              end if
            else
              {This is a new iceberg order}
              $\Psi = \psi$
              $\Phi_k = \left\{\tilde{S}, P, \Psi, \alpha\right\}$
              $\lambda = i - t$
              $\phi_{k+1,n} = \{\psi, t, \lambda\}$
            end if
            if (Inference phase) **then**
              $\mathbf{\Gamma}$ = online_inference($\mathbf{\Phi}, \mathbf{\Theta}$) {Use $\mathbf{\Theta}$ from the learning phase. Return $\mathbf{\Gamma}$.}
            end if
          end if
        end if
      end if
    end for
  end if
  if $rem(tradeSize, \Phi_k(\Psi)) == 0$ **then**
    {An existing iceberg active order has a peak size equal to integer multiples of the the max show.}
    $\phi_{k,n+1} = \{\psi, t, \lambda, \theta\}$ {Add a new tranche to an existing iceberg order. Set Φ_k(largeTrade) = 1}
  end if
end for
if (Learning phase) **then**
  {Run at the end of each trading day. offline_learning loads the previous H trading days worth of $\mathbf{\Phi}$ and concatenates with the current version.}
  $\mathbf{\Theta}$ = offline_learning($\mathbf{\Phi}$) {Return $\mathbf{\Theta}$.}
end if

EXHIBIT 16

Distribution of the Number of Tranches of an Iceberg Order

The kernel density plot shows visible spikes around round numbers, suggesting human bias.



local maxima can be seen in the distribution at multiples of 5 and 10, suggesting a human bias in selecting these numbers. In particular, the following values of N are popular: 10, 25, 40, 55, 70, 80, 90, and 110. This is in agreement with earlier studies where the icebergs are known from private data, which show that values of $N = 5$ and $N = 10$ account for 17% and 37%, respectively, of all the icebergs in the sample (Esser and Mönch [2007]). The mean value of the distribution is 29, with the minimum and maximum values of 2 and 113 respectively. Our mean is notably larger than the the findings of Frey, who finds that on average an iceberg consists of seven tranches, of which five are executed and the remaining two canceled (Frey and Sandas [2009]). By definition the algorithm does not consider an order with only one tranche to be an iceberg order, introducing a known bias to the results.

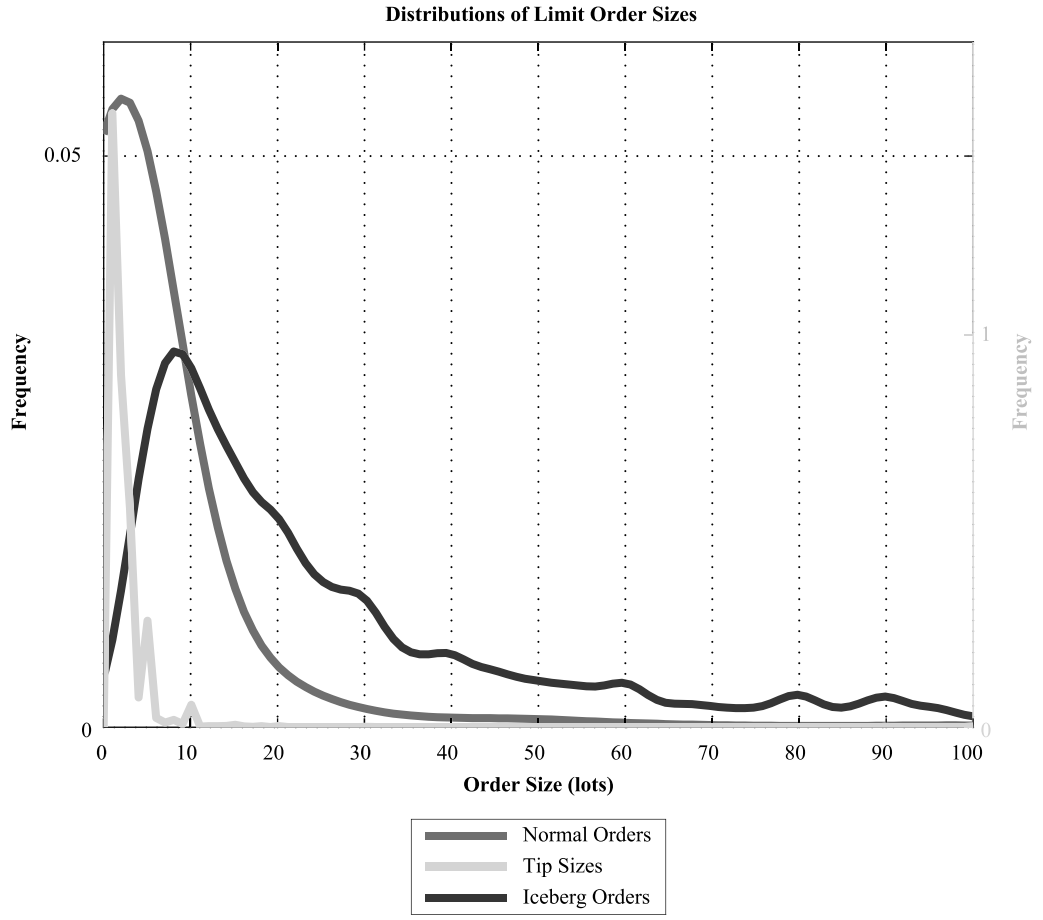
Distributions of total order size for normal limit orders, total order size for iceberg orders, and peak size for iceberg orders are shown in Exhibit 17. It can be seen that the mean iceberg order is 12 times bigger than the mean limit order (5 lots versus 62 lots), while the peak size is approximately equal to the mean limit order size. This is in close agreement with the findings of Frey, who finds that iceberg orders are 16 times bigger than average limit orders and that the peak size of these limit orders is 2.5 times bigger than the average limit order. This statistically significant increase in size suggests that the requirement to use iceberg orders comes from wishing to minimize market impact. The large size of iceberg orders is also good reason for liquidity providers to attempt to detect to detect this hidden volume.

The cumulative traded volume is shown against the cumulative traded volume from iceberg orders for a single trading day in Exhibit 18. It can be seen that iceberg orders are filled as per the trading profile of the regular market. When the whole data set is used, iceberg orders account for 12% of the total volume executed.

The number of orders present at each price is broadcast by GLOBEX. In Exhibit 19, this figure is

EXHIBIT 17
Distribution of Order Sizes

Kernel density estimates are shown for three classes of order–non-iceberg orders, iceberg orders and iceberg peak sizes.



compared against the number of iceberg orders at each price level for a single trading day. Data is bucketed into one-hour samples and the number of orders averaged. As iceberg orders in depth levels can only ever be detected if the price moves away from the inside price level, it is likely that the number of iceberg orders is underestimated for the depth price levels. The results show that on this trading day, iceberg orders comprise of 1.7% of limit orders by number and 5.6% of limit orders by volume, with nearly 100% of these occurring at the best price level. Both figures are below the data set averages of 3.2% and 9% respectively.

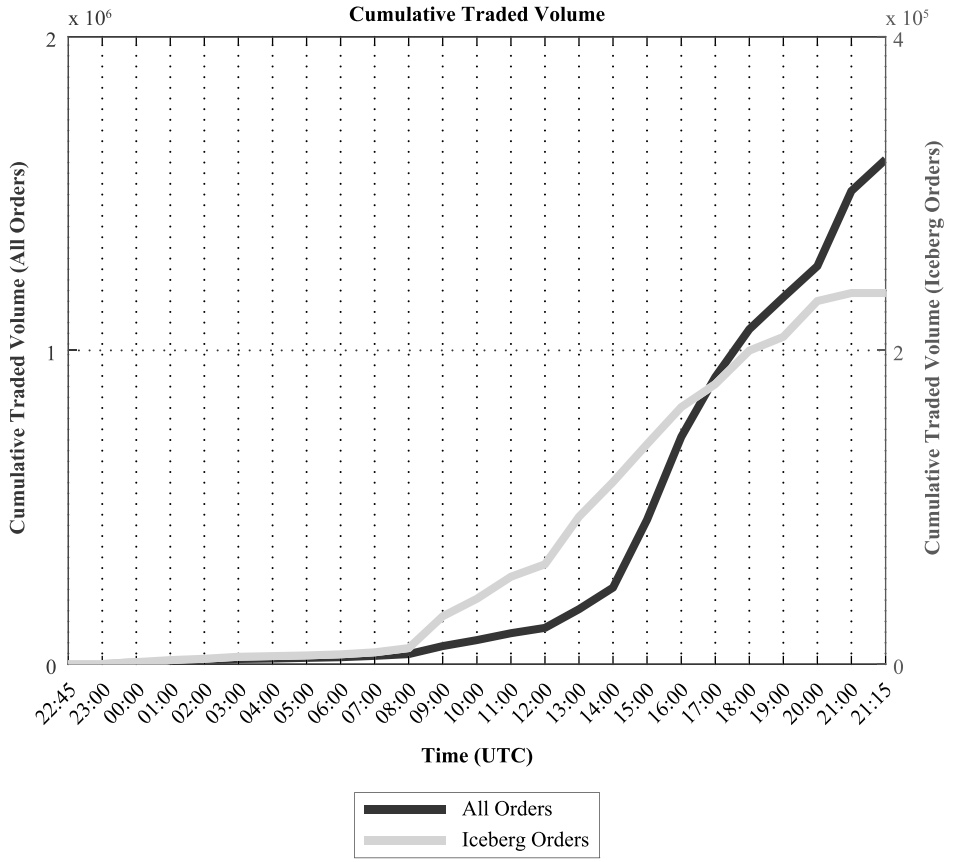
The predictive ability of Algorithm 5 is inspected by tracking the error term in the prediction of the total iceberg volume $\hat{V}$ over the course of an iceberg order being filled. Three experiments are carried out, each of which looks at $\hat{V}$ being tracked for each iceberg on the trading day of January 13-14, 2011, for ESH1, using $\boldsymbol{\Theta}$ generated from the previous trading day. Each experiment uses a different way of generating $\hat{V}_{k,n}$, allowing the performance of our approach to be benchmarked.

1. The iceberg total volume $\hat{V}_{k,n}$ is generated by the online inference Algorithm 5.
2. The iceberg total volume is randomly selected from a uniform distribution. Constraints are $\hat{V}_{k,n}$ larger than the current volume and smaller than the largest iceberg seen.
3. The iceberg total volume is equal to the current volume. The iceberg is finished or canceled.

EXHIBIT 18

Cumulative Traded Volume from Displayed and Hidden Orders

The intraday profile of hidden order fills appear visually different from that of displayed volume.



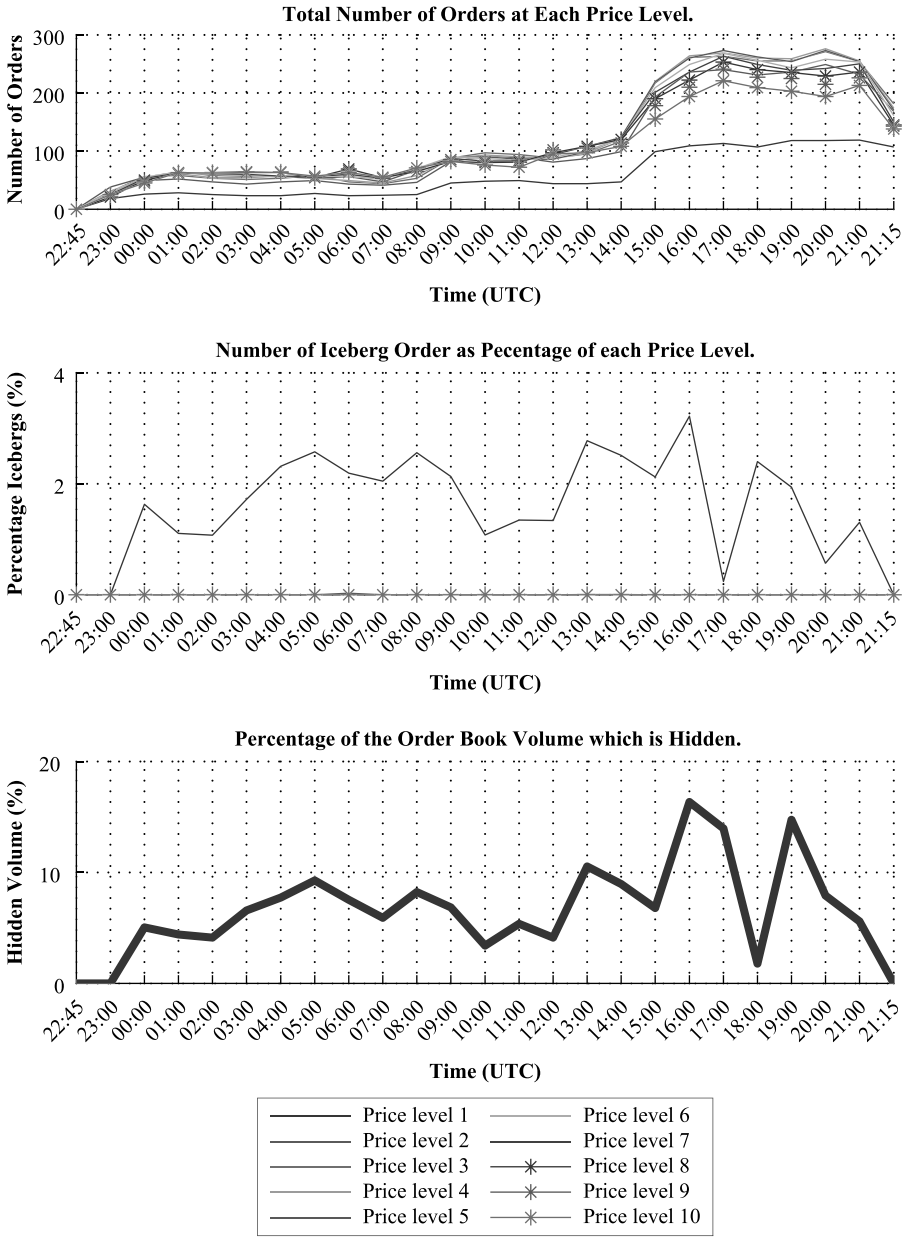
The results are expressed as root-mean-square deviation (RMSD) *prediction errors* by comparing $\hat{V}_{k,n}$ to the realized iceberg order volume $V_{\mathrm{k,R}}$ using $\mathrm{RMSD}_k = \frac{\sqrt{\sum_n^N (V_{\mathrm{k,R}} - \hat{V}_{k,n})}}{N}$. For each experiment, the progression of each iceberg order is normalized by linear interpolation, so that each iceberg order has an equal number of tranches, $N = 10$. Bootstrapping is a resampling methodology that allows a sampling distribution to be estimated from limited data. Bootstrapping enables the differences between the performance of the three experiments to be inspected for statistical significance. Using bootstrapping, for each experiment, the mean μ of the population of the means and standard deviation σ of the means are calculated. The results are shown in Exhibit 20 and Exhibit 21. Exhibit 20 shows Experiment I is seen to have the lowest μ and σ, suggesting that Algorithm 5 gives the best predictions. Experiment II is the worst performing, as expected, with both experiment I and III showing several factors of improvement. While the performance of experiment I and III is similar, the difference can be seen to statistically significant at the 95% confidence level. Exhibit 21 shows the sampling distribution of the means of each experiment, along with the mean of means and the 2.5% tail confidence intervals. For the case of experiments I and III which gave similar results, the distribution of means can be seen to be statistically significant as the mass of the distributions do not overlap.

Using ESH1 data for 13-14 January 2011 Exhibit 22 inspects the degree of sequential updates. A flat curve means that the first prediction was accurate, while a steep or non-linear curve means that the estimated volume

EXHIBIT 19

Number of Iceberg Orders at Price Levels

Price levels 1-10 are shown for the ESH1 future. Bid and ask are combined. January 13–14, 2011.



changes dramatically throughout the trading day. The information from this gradient $d\hat{V}/dn$ is captured by the RMSD. In subplot one, all the trajectories are shown, with some have a flat gradient and a few being non-linear and having a steep gradient. In subplot two, the mean curve is shown and this can be seen to be reasonably flat, suggesting that once an iceberg has been detected the predicted volume is accurate.

EXHIBIT 20
Quality of Iceberg Predictions

Experiment	μRMSD	σ_μ
I	23.57	0.017
II	1,502.34	0.144
III	24.61	0.018

Applications for the Investment Community

Hidden liquidity detection and prediction has a range of applications, from smart order routing (Almgren and Harts [2008]) to front-running (Harris [1996]). In this section, we present specific examples of how our algorithm can be used for liquidity providers and for liquidity takers.

Market makers can use a weighted bid-ask ratio as a proxy for supply and demand (see, for example, Kim et al. [2007]). An example of such a proxy is the distance of a quote from the mid, weighted by its order size, $r = \frac{|mid-bid|\times bidSize}{|mid-ask|\times askSize}$. The presence of hidden volume can cause this proxy to be wrongly estimated. Exhibit 23 shows the ratio of this proxy using hidden volume, by Algorithm 5, to the displayed volume. ESH1 data for 13-14 January 2011 was used. Only the best price levels of the LOB were used in the calculation. The ratio of the metrics was calculated as a percentage error $r = \frac{|r_I - r_{\hat{I}}|}{r_I}$ where r_I is the ratio including the iceberg volume and

EXHIBIT 21
Bootstrap Sampling of RMSDs from Iceberg Predictions

For each of the three experiments, this graphic shows the sampling distribution of the population of the means.

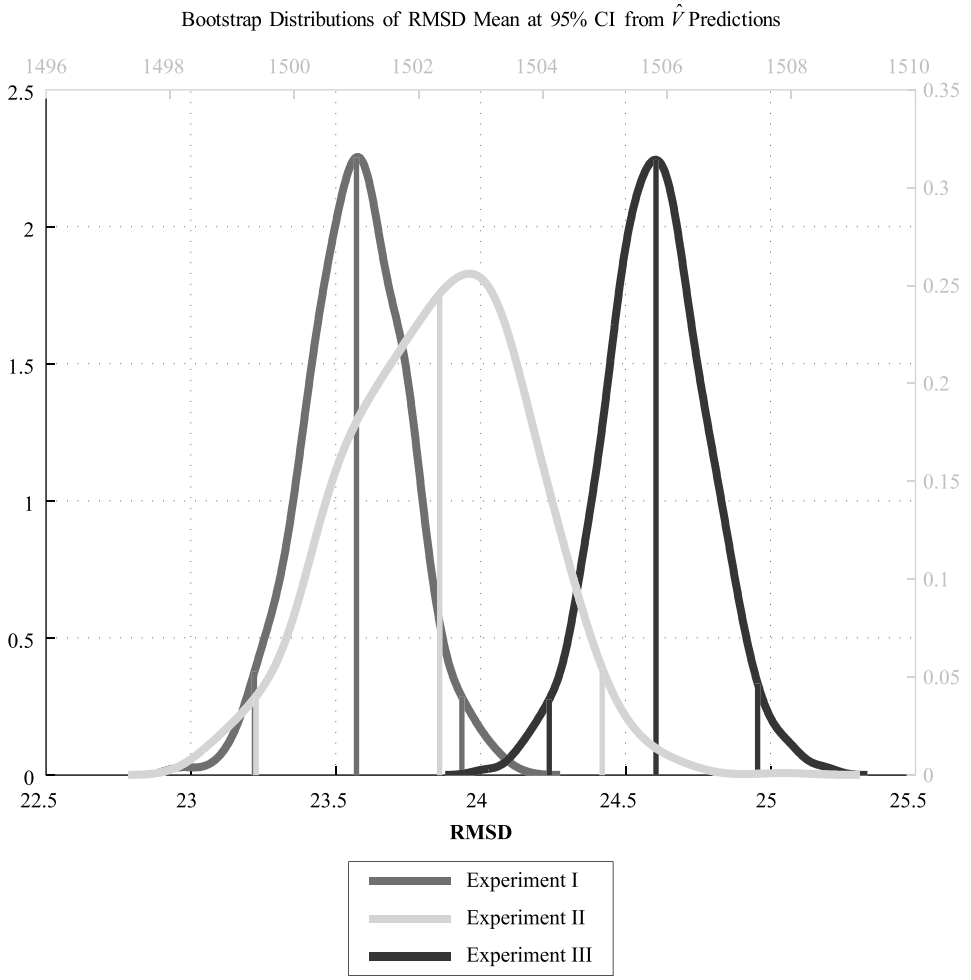


EXHIBIT 22

Quality of Iceberg Predictions from Experiment I

For all the iceberg orders on one trading day, the value of $\hat{V}$ was tracked and normalized to $N = 10$.

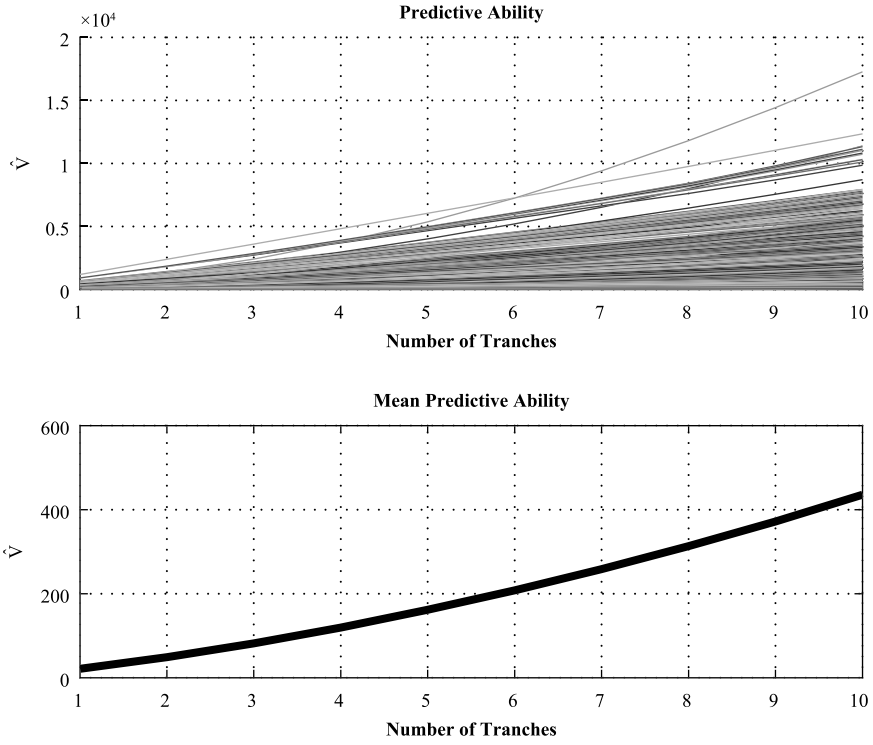
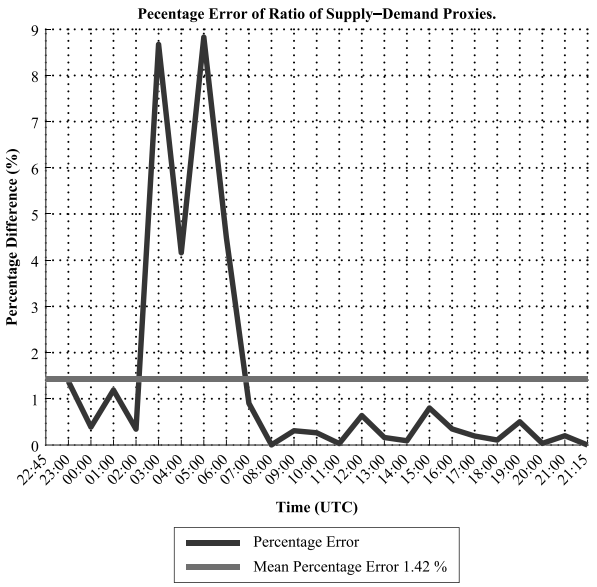


EXHIBIT 23

Time-Varying Percentage Difference for a Weighted Bid-Ask Metric, Using the LOB Volume Predicted by the Algorithm Against the Displayed LOB Volume



r_E is the ratio excluding the iceberg volume. The results show that an error greater than 1% exists when hidden volume is considered, which could be enough to affect liquidity provision based on r.

In a second liquidity supplier example, an order could be placed on the opposite side of the LOB to the detected iceberg in size equal to $\hat{V}$, to try and gain the bid-ask spread. The time until $\hat{V}$ is filled could be estimated using an order arrival rate model (see, for example, Easley et al. [2008], Wolff [1982]). The higher the arrival rate, the shorter the waiting time. The practical benefit of such a prediction would be to let the user know how long he had left to "act" on the iceberg information. In a time-priority market such as ES, joining the bottom of the queue on the other side of the LOB from Φ_k could mean that the time delay causes the iceberg to be missed and hence the liquidity supplier would be "penalized" via the exchanges quote-to-trade ratio. Such volume "targeting" strategies would increase market makers' executed volume in proportion to the volume in the LOB.

A liquidity taker uses iceberg orders to minimize market impact. When an iceberg order submitted by a liquidity taker is detected by the market, the market will stop filling it and move the price away, causing the liquidity taker to incur slippage, behavior that can be explained by the market realizing that the order imbalance has changed (Esser and Mönch [2007]). It is suggested that market impact resulting from iceberg orders being detected on GLOBEX can be minimized by canceling an iceberg after the second tranche of the iceberg order has been filled. This strategy would make the prediction algorithm presented in this article redundant, because the first tranche of an iceberg is used for the initial detection, and only after the second tranche is the order positively confirmed as an iceberg, as per Exhibit 11. By applying this strategy to a series of iceberg orders, the disadvantages associated with detection could be avoided.

CONCLUSIONS AND FURTHER WORK

Conclusions

Iceberg orders on GLOBEX are detectable and predictable. The presented approach can be used by liquidity suppliers to exploit iceberg orders, while liquidity takers can use it to submit iceberg orders in a way that inferring the true hidden volume is most difficult. The average iceberg order for the ES has total size 62 with a peak size of 5, meaning that its order size is 12 times larger than the average order size. In our ES data sample, iceberg order peaks account for 3% of all limit orders submitted by number, and iceberg orders match against 12% of all traded volume. Iceberg orders constitute 9% by volume and 1.5% by number of all resting orders in the LOB.

The gaming behavior presented in this article could be countered by GLOBEX supporting an optional FIX tag to allow randomization of certain parts of the iceberg order, such as Ψ, while constraining $V = \sum_n^N \psi_n$. Not to support such functionality may be unfair to less sophisticated investors, who may not realize the potential consequences of using iceberg orders. Having markets which are fair and transparent is central to the integrity of the global financial system.

Further Work

By applying the algorithm across the seven major asset sectors (equity index, STIRs, bonds, FX, agriculturals, energies, metals) traded on GLOBEX, iceberg orders may be seen to play a more important role in some sectors than others it is speculated that this might be related to the matching algorithm used in the sector, specifically if that algorithm has a time component.

An additional modification to the learning step presented in this article could be to carry out estimation of Θ conditional on various factors which could affect either V or Ψ: for example, the distance between the limit price and the mid-price, the size of the bid-ask spread, or market volatility.

The empirical model presented in this article can be reformulated as a probabilistic Bayesian model, and we plan to publish on such a model in the future. By defining the state variables as the peak size and the total size, the process can be represented as a Markov chain with some deterministic transition probabilities.

A variety of patterns exists in the LOB. Some of these patterns are generated by system effects, such as order types and latencies, and some patterns are generated by repetitive human behavior, such as order sizes and support levels. An exciting area of future work will be further automating pattern recognition in the LOB using expectation maximization algorithms, such as Baum-Welch, to learn latent structure (Baum et al. [1970]).

ENDNOTES

We would like to thank www.cmegroup.com for allowing us to use their data in this research. We would also like to thank www.onixs.biz for allowing us to use their C# FAST decoder.

[1]www. futuresindustry.org/ptg/membership.asp

[2]www.rsj.com/en/algorithmic-trading/current-volumes

[3]Only interest rate options do not support icebergs on GLOBEX.

[4]FIX tag 210

[5]When FIX tag 277=1, trade volume is also not removed from the book.

[6]www.cmegroup.com/colo

[7]http://aws.amazon.com

REFERENCES

Aitchison, J., and C. Aitken. "Multivariate Binary Discrimination by the Kernel Method." *Biometrika*, Vol. 63, No. 3 (1976), pp. 413-420.

Almgren, R., and B. Harts. "A Dynamic Algorithm for Smart Order Routing." White paper, StreamBase, 2008.

Avellaneda, M., J. Reed, and S. Stoikov. "Forecasting Prices from Level-I Quotes in the Presence of Hidden Liquidity." *Algorithmic Finance*, Vol. 1, No. 1 (2011), pp. 35-43.

Bank for International Settlement. "*Triennial Central Bank Survey of Foreign Exchange and Derivatives Market Activity in 2010*." 2010.

Baum, L., T. Petrie, G. Soules, and N. Weiss. "A Maximization Technique Occurring in the Statistical Analysis of Probabilistic Functions of Markov Chains." *The Annals of Mathematical Statistics*, Vol. 41, No. 1 (1970), pp. 164-171.

Bessembinder, H., M. Panayides, and K. Venkataraman. "Hidden Liquidity: An Analysis of Order Exposure Strategies in Electronic Stock Markets." *Journal of Financial Economics*, Vol. 94, No. 3 (2009), pp. 361-383.

Blank, J. "Implied Trading in Energy Futures." *The Journal of Trading*, Vol. 2, No. 3 (2007), pp. 45-48.

Borkovec, M. "Algorithmic Trading System and Method." U.S. Patent 8,140,416, 2012.

Burghardt, G., J. Hanweck, and L. Lei. "Measuring Market Impact and Liquidity." *The Journal of Trading*, Vol. 1, No. 4 (2006), pp. 70-84.

Cave, T. "Europe's Top 10 High-Frequency Kingmakers (in Scandinavia, at Least)." Dow Jones, October 2011.

Christensen, H., R. Turner, S. Hill, and S. Godsill. "Rebuilding the Limit Order Book: Sequential Bayesian Inference on Hidden States." *Quantitative Finance* (2013), submitted.

Chicago Mercantile Exchange. First Quarter Liquidity Monitor. www.cmegroup.com/education/files/Liquidity-Monitor-Q1-2011.pdf, 2011.

——. CME Data Mine. Market Data Platform FIX/FAST. Core Functionality. Version 3.2, 2012.

——. GCC Product Reference Sheet. www.cmegroup.com/confluence/display/EPICSANDBOX/GCC+Product+Reference+Sheet, 2013a.

——. Product Specific Information. www.cmegroup.com/confluence/display/EPICSANDBOX/Product+Specific+Information, 2013b.

Cont, R., S. Stoikov, and R. Talreja. "A Stochastic Model for Order Book Dynamics." *Operations Research*, Vol. 58, No. 3 (2010), pp. 549-563.

De Winne, R., and C. D'hondt. "Hide-and-Seek in the Market: Placing and Detecting Hidden Orders." *Review of Finance*, Vol. 11, No. 4 (2007), pp. 663-692.

Durbin, M. *All About High Frequency Trading*, 1st ed. McGraw-Hill Professional, 2010.

Easley, D., R. Engle, M. O'Hara, and L. Wu. "Time-Varying Arrival Rates of Informed and Uninformed Trades." *Journal of Financial Econometrics*, Vol. 6, No. 2 (2008), pp. 171-207.

Esser, A., and B. Mönch. "The Navigation of an Iceberg: The Optimal Use of Hidden Orders." *Finance Research Letters*, Vol. 4, No. 2 (2007), pp. 68-81.

Fleming, M., and B. Mizrach. "The Microstructure of a U.S. Treasury ECN: The Brokertec Platform." Federal Reserve Bank of New York, 2009.

Frey, S., and P. Sandas. "The Impact of Iceberg Orders in Limit Order Books." American Finance Association San Francisco Meetings Papers, 2009.

Haddad, R., and A. Akansu. "A Class of Fast Gaussian Binomial Filters for Speech and Image Processing." *Signal Processing, IEEE Transactions*, Vol. 39, No. 3 (1991), pp. 723-727.

Harris, L."Does a Large Minimum Price Variation Encourage Order Exposure?" New York Stock Exchange, 1996.

Hasbrouck, J., and G. Saar. "Limit Orders and Volatility in a Hybrid Market: The Island ECN." New York University, Working Paper, No. FIN-01-025, 2001.

Hastie, T., R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*, 2nd ed. Springer, 2001.

Hautsch, N., and R. Huang. "A Statistical Model for Detecting Hidden Liquidity." Quantitative Products Laboratory, Berlin, 2009.

Hunsader, S. "Analysis of the Flash Crash." NANEX, 2010.

Janecek, K., and M. Kabrhel. "Matching Algorithms of International Exchanges." 2007.

Kim, A., J.D. Farmer, and A. Lo. "Market Making by Learning Liquidity Imbalance." MIT, 2007.

Madhavan, A. "Market Microstructure: A Survey." *Journal of Financial Markets*, Vol. 3, No. 3 (2000), pp. 205-258.

Moinas, S. "Hidden Limit Orders and Liquidity in Limit Order Markets." Toulouse Business School, 2006.

Moro, E., J. Vicente, L. Moyano, A. Gerig, J. Farmer, G. Vaglica, F. Lillo, and R. Mantegna. "Market Impact and Trading Profile of Hidden Orders in Stock Markets." *Physical Review E*, Vol. 80, No 6 (2009), pp. 0661021-0661028.

Overdahl, J. "Implied Matching Functionality in Futures Markets." *Futures Industry* (November 2011), pp. 37-41.

Rajagopalan, B., and U. Lall. "A Kernel Estimator for Discrete Distributions." *Journal of Nonparametric Statistics*, Vol. 4, No. 4 (1995), pp. 409-426.

Scott, D. *Multivariate Density Estimation*. Wiley, 1992.

Setzkorn, C., and R. Paton. "Javaspaces: An Affordable Technology for the Simple Implementation of Reusable Parallel Evolutionary Algorithms." In *Knowledge Exploration in Life Science Informatics*, edited by W. Dubitzky, Springer, 2004, pp. 151-160.

Thain, D., T. Tannenbaum, and M. Livny. "Distributed Computing in Practice: The Condor Experience." *Concurrency and Computation: Practice and Experience*, 17 (2005), pp. 323-356.

Tuttle, L. "Hidden Orders, Trading Costs and Information." 2005.

Vaglica, G., F. Lillo, E. Moro, and R. Mantegna. "Scaling Laws of Strategic Behavior and Size Heterogeneity in Agent Dynamics." *Physical Review E*, Vol. 77, No. 3 (2008).

Wolff, R. "Poisson Arrivals See Time Averages." *Operations Research*, Vol. 30, No. 2 (1982), pp. 223-231.

Yao, C. "Hidden Agendas: A Study of the Impact of Concealed Orders." University of Illinois, 2012.

Erratum: Prediction of Hidden Liquidity in the Limit Order Book of GLOBEX Futures

Hugh L. Christensen[a,*], Robert Woodmansey[b]

[a]*Signal Processing and Communications Laboratory, Engineering Department, Cambridge University, CB2 1PZ, UK*
[b]*Onix Solutions, Alpha House, 100 Borough High Street, London, SE1 1LB, UK*

In our recent article we describe an iceberg prediction algorithm for limit order books (LOB) on CME GLOBEX (Christensen and Woodmansey, 2013). In the article successive peaks of an iceberg orders are described as retaining time priority. This is incorrect. Successive peaks of an iceberg order do *not* retain time priority. Each time a peak is filled, the refresh peak is inserted into the LOB at the back of the queue. CME Group have updated their public documentation to reflect this (CME, 2013).

These mechanics are now illustrated with an example, as shown in a revised Exhibit 7. An iceberg order is specified with a total size $V = 100$ and a max show $\Psi = 9$. The first 11 tranches of the iceberg are of size $\psi = 9$ and the final tranche is size $\bar{\psi} = 1$, so $N = 12$ (where N is the number of tranches). At time $t = 2$ the first peak of the iceberg can be seen in the LOB, behind a normal limit order for $S = 10$ (S is size). At this point in time, the hidden volume in the LOB is $V_H = 91$. At time step $t = 6$, a trade message for $S = 8$ is seen in the data feed. This results in two further messages being sent by GLOBEX, firstly, a LOB update message $dV = -8$, and secondly a peak replenish message $dV = 9$. The first of these messages is the trade volume being removed from the LOB. The second of these messages is the next tranche of the iceberg order being placed into the LOB *behind* the normal limit order of $S = 3$ (i.e. the normal limit order has higher time priority than the peak replenish order). It this peak replenish message time dt after a trade which allows the iceberg order to be detected.

As time-priority is lost between each peak, in essence an iceberg order is equivalent to a series of sequential limit orders entered by the trader. In reality, given the time it takes for a fill to be relayed back to the customer, iceberg orders are significantly quicker to be inserted into the LOB and also more convenient.

How does this change in system mechanics affect the prediction algorithm? The loss of time priority between tranches does *not* significantly affect the prediction algorithm. In the algorithm, orders which enter the LOB within dt seconds of a trade are considered viable candidates for peak refresh orders. We had initially reported that these peak refresh orders went to the front of the queue (having retained time priority). It is now known they go to the back of the queue having lost time priority. The only change to the algorithm requires a modification to the way in which the cumulative volume is tracked (page 79). Originally $\alpha = 0$ is set when the tracked cumulative volume exceeds the max show, plus dt to allow for system latency effects. Setting $\alpha = 0$ means that the iceberg order is no longer active (i.e. it has either been filled or cancelled). Now, in light of the loss of time priority between peaks, the α is set to zero when the tracked cumulative volume exceeds the max show plus the sum observable volume at the price level plus dt.

Does this change to the algorithm affect the presented results? Having re-run the algorithm using the updated methodology, the answer is no, hardly at all. The inside price level of ES contains significantly less volume than the depth price levels. For the year 2011, the volume of the inside price level (averaged over bid and ask) is approximately 750 lots. The differences between the old and new implementation of the algorithm were inspected. It was seen that the previous choice of dt, used to allow for system latency, was essentially acting as a buffer term. This meant significantly more volume was received than was present in the max show Ψ. Searching over this extended period allowed us to detect the refresh peak entering the LOB. In other words, by setting

*Corresponding author.
Email addresses: hlc54@cam.ac.uk (Hugh L. Christensen), robert.woodmansey@onixs.biz (Robert Woodmansey)

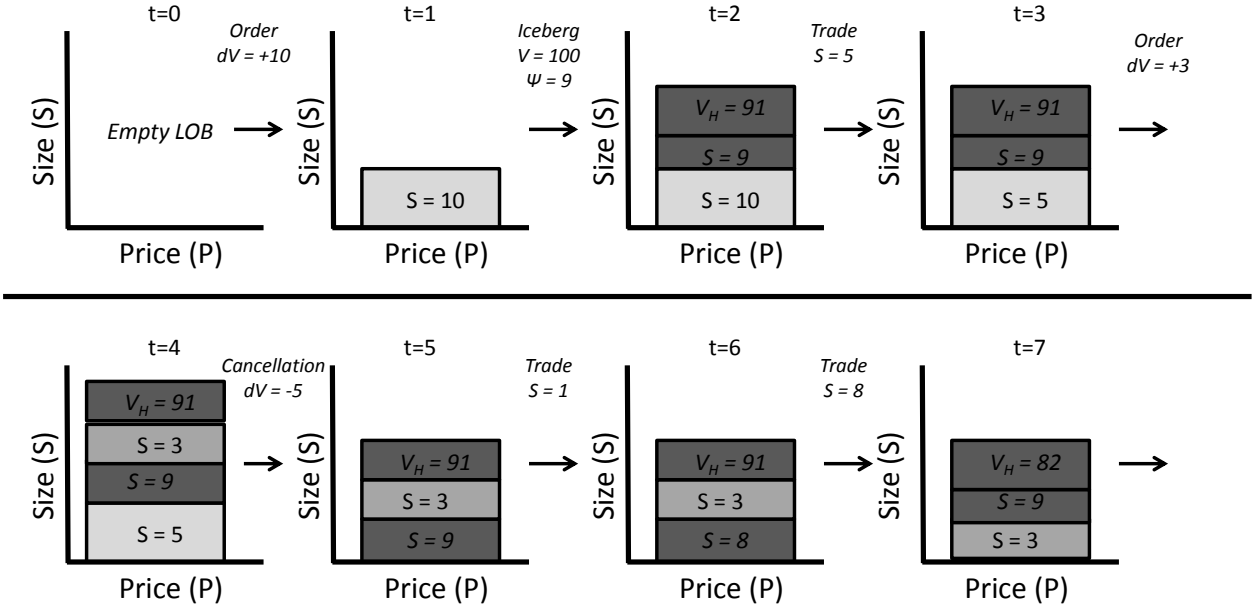


Exhibit 7: Schematic of iceberg order mechanics. A simplified LOB consisting of just one price level is shown progressing through time. At each step a FIX message is applied to the LOB. The bottom of stack has the highest time priority. Normal limit orders are shown in light greys, iceberg order in dark grey. V_H denotes hidden volume which can *not* be seen in the LOB.

dt to be large, it had the same effect as summing volume over the price level. This is in agreement with Exhibit 15, the distribution of the latency of iceberg refresh messages, which shows a maximum at 8 milliseconds and a tail extending out to 400 milliseconds. The maximum corresponds to either high number of lots per second being traded, or to when the price level contained little volume. The long tail corresponds to the opposite case, when it took some time to trade through the whole price level and the new peak replenish message appear.

References

Christensen, Hugh, R.Woodmansey. "Prediction of Hidden Liquidity in the Limit Order Book of GLOBEX Futures." *The Journal of Trading*, 8 (3) (1976), pp. 68–95.

Chicago Mecantile Exchange. "Order Qualifiers." `www.cmegroup.com/confluence/display/EPICSANDBOX/Order+Qualifiers`.

Rules of Machine Learning: Best Practices for ML Engineering

Martin Zinkevich

This document is intended to help those with a basic knowledge of machine learning get the benefit of best practices in machine learning from around Google. It presents a style for machine learning, similar to the Google C++ Style Guide and other popular guides to practical programming. If you have taken a class in machine learning, or built or worked on a machine-learned model, then you have the necessary background to read this document.

Terminology

The following terms will come up repeatedly in our discussion of effective machine learning:

Instance: The thing about which you want to make a prediction. For example, the instance might be a web page that you want to classify as either "about cats" or "not about cats".
Label: An answer for a prediction task -- either the answer produced by a machine learning system, or the right answer supplied in training data. For example, the label for a web page might be "about cats".
Feature: A property of an instance used in a prediction task. For example, a web page might have a feature "contains the word 'cat'".
Feature Column[1]: A set of related features, such as the set of all possible countries in which users might live. An example may have one or more features present in a feature column. A feature column is referred to as a “namespace” in the VW system (at Yahoo/Microsoft), or a field.
Example: An instance (with its features) and a label.
Model: A statistical representation of a prediction task. You *train* a model on examples then use the model to make predictions.
Metric: A number that you care about. May or may not be directly optimized.
Objective: A metric that your algorithm is trying to optimize.
Pipeline: The infrastructure surrounding a machine learning algorithm. Includes gathering the data from the front end, putting it into training data files, training one or more models, and exporting the models to production.

Overview

To make great products:
do machine learning like the great engineer you are, not like the great machine learning expert you aren’t.

[1] Google-specific terminology.

Most of the problems you will face are, in fact, engineering problems. Even with all the resources of a great machine learning expert, most of the gains come from great features, not great machine learning algorithms. So, the basic approach is:

1. make sure your pipeline is solid end to end
2. start with a reasonable objective
3. add common-sense features in a simple way
4. make sure that your pipeline stays solid.

This approach will make lots of money and/or make lots of people happy for a long period of time. Diverge from this approach only when there are no more simple tricks to get you any farther. Adding complexity slows future releases.

Once you've exhausted the simple tricks, cutting-edge machine learning might indeed be in your future. See the section on Phase III machine learning projects.

This document is arranged in four parts:

1. The first part should help you understand whether the time is right for building a machine learning system.
2. The second part is about deploying your first pipeline.
3. The third part is about launching and iterating while adding new features to your pipeline, how to evaluate models and training-serving skew.
4. The final part is about what to do when you reach a plateau.
5. Afterwards, there is a list of related work and an appendix with some background on the systems commonly used as examples in this document.

Before Machine Learning

Rule #1: Don't be afraid to launch a product without machine learning.
Machine learning is cool, but it requires data. Theoretically, you can take data from a different problem and then tweak the model for a new product, but this will likely underperform basic heuristics. If you think that machine learning will give you a 100% boost, then a heuristic will get you 50% of the way there.

For instance, if you are ranking apps in an app marketplace, you could use the install rate or number of installs. If you are detecting spam, filter out publishers that have sent spam before. Don't be afraid to use human editing either. If you need to rank contacts, rank the most recently used highest (or even rank alphabetically). If machine learning is not absolutely required for your product, don't use it until you have data.

Rule #2: First, design and implement metrics.
Before formalizing what your machine learning system will do, track as much as possible in your current system. Do this for the following reasons:

1. It is easier to gain permission from the system's users earlier on.
2. If you think that something might be a concern in the future, it is better to get historical data now.
3. If you design your system with metric instrumentation in mind, things will go better for you in the future. Specifically, you don't want to find yourself grepping for strings in logs to instrument your metrics!
4. You will notice what things change and what stays the same. For instance, suppose you want to directly optimize one-day active users. However, during your early manipulations of the system, you may notice that dramatic alterations of the user experience don't noticeably change this metric.

Google Plus team measures expands per read, reshares per read, plus-ones per read, comments/read, comments per user, reshares per user, etc. which they use in computing the goodness of a post at serving time. **Also, note that an experiment framework, where you can group users into buckets and aggregate statistics by experiment, is important**. See **Rule #12**.

By being more liberal about gathering metrics, you can gain a broader picture of your system. Notice a problem? Add a metric to track it! Excited about some quantitative change on the last release? Add a metric to track it!

Rule #3: Choose machine learning over a complex heuristic.
A simple heuristic can get your product out the door. A complex heuristic is unmaintainable. Once you have data and a basic idea of what you are trying to accomplish, move on to machine learning. As in most software engineering tasks, you will want to be constantly updating your approach, whether it is a heuristic or a machine-learned model, and you will find that the machine-learned model is easier to update and maintain (see Rule #16).

ML Phase I: Your First Pipeline

Focus on your system infrastructure for your first pipeline. While it is fun to think about all the imaginative machine learning you are going to do, it will be hard to figure out what is happening if you don't first trust your pipeline.

Rule #4: Keep the first model simple and get the infrastructure right.
The first model provides the biggest boost to your product, so it doesn't need to be fancy. But you will run into many more infrastructure issues than you expect. Before anyone can use your fancy new machine learning system, you have to determine:

1. How to get examples to your learning algorithm.
2. A first cut as to what “good” and “bad” mean to your system.
3. How to integrate your model into your application. You can either apply the model live, or pre-compute the model on examples offline and store the results in a table. For example, you might want to pre-classify web pages and store the results in a table, but you might want to classify chat messages live.

Choosing simple features makes it easier to ensure that:

1. The features reach your learning algorithm correctly.
2. The model learns reasonable weights.
3. The features reach your model in the server correctly.

Once you have a system that does these three things reliably, you have done most of the work. Your simple model provides you with baseline metrics and a baseline behavior that you can use to test more complex models. Some teams aim for a “neutral” first launch: a first launch that explicitly de-prioritizes machine learning gains, to avoid getting distracted.

Rule #5: Test the infrastructure independently from the machine learning.

Make sure that the infrastructure is testable, and that the learning parts of the system are encapsulated so that you can test everything around it. Specifically:

1. Test getting data into the algorithm. Check that feature columns that should be populated are populated. Where privacy permits, manually inspect the input to your training algorithm. If possible, check statistics in your pipeline in comparison to elsewhere, such as RASTA.
2. Test getting models out of the training algorithm. Make sure that the model in your training environment gives the same score as the model in your serving environment (see Rule #37).

Machine learning has an element of unpredictability, so make sure that you have tests for the code for creating examples in training and serving, and that you can load and use a fixed model during serving. Also, it is important to understand your data: see Practical Advice for Analysis of Large, Complex Data Sets.

Rule #6: Be careful about dropped data when copying pipelines.

Often we create a pipeline by copying an existing pipeline (i.e. cargo cult programming), and the old pipeline drops data that we need for the new pipeline. For example, the pipeline for Google Plus What’s Hot drops older posts (because it is trying to rank fresh posts). This pipeline was copied to use for Google Plus Stream, where older posts are still meaningful, but the pipeline was still dropping old posts. Another common pattern is to only log data that was seen by the user. Thus, this data is useless if we want to model why a particular post was not seen by the user, because all the negative examples have been dropped. A similar issue occurred in Play. While working on Play Apps Home, a new pipeline was created that also contained examples from two other landing pages (Play Games Home and Play Home Home) without any feature to disambiguate where each example came from.

Rule #7: Turn heuristics into features, or handle them externally.
Usually the problems that machine learning is trying to solve are not completely new. There is an existing system for ranking, or classifying, or whatever problem you are trying to solve. This means that there are a bunch of rules and heuristics. **These same heuristics can give you a lift when tweaked with machine learning**. Your heuristics should be mined for whatever information they have, for two reasons. First, the transition to a machine learned system will be smoother. Second, usually those rules contain a lot of the intuition about the system you don't want to throw away. There are four ways you can use an existing heuristic:

1. **Preprocess using the heuristic**. If the feature is incredibly awesome, then this is an option. For example, if, in a spam filter, the sender has already been blacklisted, don't try to relearn what "blacklisted" means. Block the message. This approach makes the most sense in binary classification tasks.
2. **Create a feature**. Directly creating a feature from the heuristic is great. For example, if you use a heuristic to compute a relevance score for a query result, you can include the score as the value of a feature. Later on you may want to use machine learning techniques to massage the value (for example, converting the value into one of a finite set of discrete values, or combining it with other features) but start by using the raw value produced by the heuristic.
3. **Mine the raw inputs of the heuristic**. If there is a heuristic for apps that combines the number of installs, the number of characters in the text, and the day of the week, then consider pulling these pieces apart, and feeding these inputs into the learning separately. Some techniques that apply to ensembles apply here (see Rule #40).
4. **Modify the label.** This is an option when you feel that the heuristic captures information not currently contained in the label. For example, if you are trying to maximize the number of downloads, but you also want quality content, then maybe the solution is to multiply the label by the average number of stars the app received. There is a lot of space here for leeway. See the section on "Your First Objective".

Do be mindful of the added complexity when using heuristics in an ML system. Using old heuristics in your new machine learning algorithm can help to create a smooth transition, but think about whether there is a simpler way to accomplish the same effect.

Monitoring

In general, practice good alerting hygiene, such as making alerts actionable and having a dashboard page.

Rule #8: Know the freshness requirements of your system.
How much does performance degrade if you have a model that is a day old? A week old? A quarter old? This information can help you to understand the priorities of your monitoring. If you lose 10% of your revenue if the model is not updated for a day, it makes sense to have an engineer watching it continuously. Most ad serving systems have new advertisements to handle

every day, and must update daily. For instance, if the ML model for Google Play Search is not updated, it can have an impact on revenue in under a month. Some models for What's Hot in Google Plus have no post identifier in their model so they can export these models infrequently. Other models that have post identifiers are updated much more frequently. Also notice that freshness can change over time, especially when feature columns are added or removed from your model.

Rule #9: Detect problems before exporting models.
Many machine learning systems have a stage where you export the model to serving. If there is an issue with an exported model, it is a user-facing issue. If there is an issue before, then it is a training issue, and users will not notice.

Do sanity checks right before you export the model. Specifically, make sure that the model's performance is reasonable on held out data. Or, if you have lingering concerns with the data, don't export a model. Many teams continuously deploying models check the area under the ROC curve (or AUC) before exporting. **Issues about models that haven't been exported require an e-mail alert, but issues on a user-facing model may require a page.** So better to wait and be sure before impacting users.

Rule #10: Watch for silent failures.
This is a problem that occurs more for machine learning systems than for other kinds of systems. Suppose that a particular table that is being joined is no longer being updated. The machine learning system will adjust, and behavior will continue to be reasonably good, decaying gradually. Sometimes tables are found that were months out of date, and a simple refresh improved performance more than any other launch that quarter! For example, the coverage of a feature may change due to implementation changes: for example a feature column could be populated in 90% of the examples, and suddenly drop to 60% of the examples. Play once had a table that was stale for 6 months, and refreshing the table alone gave a boost of 2% in install rate. If you track statistics of the data, as well as manually inspect the data on occassion, you can reduce these kinds of failures.

Rule #11: Give feature column owners and documentation.
If the system is large, and there are many feature columns, know who created or is maintaining each feature column. If you find that the person who understands a feature column is leaving, make sure that someone has the information. Although many feature columns have descriptive names, it's good to have a more detailed description of what the feature is, where it came from, and how it is expected to help.

Your First Objective

You have many metrics, or measurements about the system that you care about, but your machine learning algorithm will often require a single **objective, a number that your algorithm**

is "trying" to optimize. I distinguish here between objectives and metrics: **a metric is any number that your system reports**, which may or may not be important. See also Rule #2.

Rule #12: Don't overthink which objective you choose to directly optimize.
You want to make money, make your users happy, and make the world a better place. There are tons of metrics that you care about, and you should measure them all (see **Rule #2**). However, early in the machine learning process, you will notice them all going up, even those that you do not directly optimize. For instance, suppose you care about number of clicks, time spent on the site, and daily active users. If you optimize for number of clicks, you are likely to see the time spent increase.
So, keep it simple and don't think too hard about balancing different metrics when you can still easily increase all the metrics. Don't take this rule too far though: do not confuse your objective with the ultimate health of the system (see Rule #39). And, **if you find yourself increasing the directly optimized metric, but deciding not to launch, some objective revision may be required.**

Rule #13: Choose a simple, observable and attributable metric for your first objective.
Often you don't know what the true objective is. You think you do but then you as you stare at the data and side-by-side analysis of your old system and new ML system, you realize you want to tweak it. Further, different team members often can't agree on the true objective. **The ML objective should be something that is easy to measure and is a proxy for the "true" objective**[2]. So train on the simple ML objective, and consider having a "policy layer" on top that allows you to add additional logic (hopefully very simple logic) to do the final ranking.

The easiest thing to model is a user behavior that is directly observed and attributable to an action of the system:

1. Was this ranked link clicked?
2. Was this ranked object downloaded?
3. Was this ranked object forwarded/replied to/e-mailed?
4. Was this ranked object rated?
5. Was this shown object marked as spam/pornography/offensive?

Avoid modeling indirect effects at first:

1. Did the user visit the next day?
2. How long did the user visit the site?
3. What were the daily active users?

Indirect effects make great metrics, and can be used during A/B testing and during launch decisions.
Finally, don't try to get the machine learning to figure out:

1. Is the user happy using the product?
2. Is the user satisfied with the experience?
3. Is the product improving the user's overall well-being?

---

[2] There is often no "true" objective. See Rule #39.

4. How will this affect the company's overall health?

These are all important, but also incredibly hard. Instead, use proxies: if the user is happy, they will stay on the site longer. If the user is satisfied, they will visit again tomorrow. Insofar as well-being and company health is concerned, human judgement is required to connect any machine learned objective to the nature of the product you are selling and your business plan, so we don't end up here.

Rule #14: Starting with an interpretable model makes debugging easier.

Linear regression, logistic regression, and Poisson regression are directly motivated by a probabilistic model. Each prediction is interpretable as a probability or an expected value. This makes them easier to debug than models that use objectives (zero-one loss, various hinge losses, et cetera) that try to directly optimize classification accuracy or ranking performance. For example, if probabilities in training deviate from probabilities predicted in side-by-sides or by inspecting the production system, this deviation could reveal a problem.

For example, in linear, logistic, or Poisson regression, **there are subsets of the data where the average predicted expectation equals the average label (1-moment calibrated, or just calibrated)**[3]. If you have a feature which is either 1 or 0 for each example, then the set of examples where that feature is 1 is calibrated. Also, if you have a feature that is 1 for every example, then the set of all examples is calibrated.

With simple models, it is easier to deal with feedback loops (see Rule #36).

Often, we use these probabilistic predictions to make a decision: e.g. rank posts in decreasing expected value (i.e. probability of click/download/etc.). **However, remember when it comes time to choose which model to use, the decision matters more than the likelihood of the data given the model (see Rule #27).**

Rule #15: Separate Spam Filtering and Quality Ranking in a Policy Layer.

Quality ranking is a fine art, but spam filtering is a war. The signals that you use to determine high quality posts will become obvious to those who use your system, and they will tweak their posts to have these properties. Thus, your quality ranking should focus on ranking content that is posted in good faith. You should not discount the quality ranking learner for ranking spam highly. **Similarly, "racy" content should be handled separately from Quality Ranking**.

Spam filtering is a different story. You have to expect that the features that you need to generate will be constantly changing. Often, there will be obvious rules that you put into the system (if a post has more than three spam votes, don't retrieve it, et cetera). Any learned model will have to be updated daily, if not faster. The reputation of the creator of the content will play a great role.

At some level, the output of these two systems will have to be integrated. Keep in mind, filtering spam in search results should probably be more aggressive than filtering spam in email

[3] This is true assuming that you have no regularization and that your algorithm has converged. It is approximately true in general.

messages. Also, it is a standard practice to remove spam from the training data for the quality classifier.

ML Phase II: Feature Engineering

In the first phase of the lifecycle of a machine learning system, the important issue is to get the training data into the learning system, get any metrics of interest instrumented, and create a serving infrastructure. **After you have a working end to end system with unit and system tests instrumented, Phase II begins**.

In the second phase, there is a lot of low-hanging fruit. There are a variety of obvious features that could be pulled into the system. Thus, the second phase of machine learning involves pulling in as many features as possible and combining them in intuitive ways. During this phase, all of the metrics should still be rising. There will be lots of launches, and it is a great time to pull in lots of engineers that can join up all the data that you need to create a truly awesome learning system.

Rule #16: Plan to launch and iterate.
Don't expect that the model you are working on now will be the last one that you will launch, or even that you will ever stop launching models. Thus consider whether the complexity you are adding with this launch will slow down future launches. Many teams have launched a model per quarter or more for years. There are three basic reasons to launch new models:

1. you are coming up with new features,
2. you are tuning regularization and combining old features in new ways, and/or
3. you are tuning the objective.

Regardless, giving a model a bit of love can be good: looking over the data feeding into the example can help find new signals as well as old, broken ones. So, as you build your model, think about how easy it is to add or remove or recombine features. Think about how easy it is to create a fresh copy of the pipeline and verify its correctness. Think about whether it is possible to have two or three copies running in parallel. Finally, don't worry about whether feature 16 of 35 makes it into this version of the pipeline. You'll get it next quarter.

Rule #17: Start with directly observed and reported features as opposed to learned features.
This might be a controversial point, but it avoids a lot of pitfalls. First of all, let's describe what a learned feature is. A learned feature is a feature generated either by an external system (such as an unsupervised clustering system) or by the learner itself (e.g. via a factored model or deep

learning). Both of these can be useful, but they can have a lot of issues, so they should not be in the first model.

If you use an external system to create a feature, remember that the system has its own objective. The external system's objective may be only weakly correlated with your current objective. If you grab a snapshot of the external system, then it can become out of date. If you update the features from the external system, then the meanings may change. If you use an external system to provide a feature, be aware that they require a great deal of care.

The primary issue with factored models and deep models is that they are non-convex. Thus, there is no guarantee that an optimal solution can be approximated or found, and the local minima found on each iteration can be different. This variation makes it hard to judge whether the impact of a change to your system is meaningful or random. By creating a model without deep features, you can get an excellent baseline performance. After this baseline is achieved, you can try more esoteric approaches.

Rule #18: Explore with features of content that generalize across contexts.
Often a machine learning system is a small part of a much bigger picture. For example, if you imagine a post that might be used in What's Hot, many people will plus-one, re-share, or comment on a post before it is ever shown in What's Hot. If you provide those statistics to the learner, it can promote new posts that it has no data for in the context it is optimizing. YouTube Watch Next could use number of watches, or co-watches (counts of how many times one video was watched after another was watched) from YouTube search. You can also use explicit user ratings. Finally, if you have a user action that you are using as a label, seeing that action on the document in a different context can be a great feature. All of these features allow you to bring new content into the context. Note that this is not about personalization: figure out if someone likes the content in this context first, then figure out who likes it more or less.

Rule #19: Use very specific features when you can.
With tons of data, it is simpler to learn millions of simple features than a few complex features. Identifiers of documents being retrieved and canonicalized queries do not provide much generalization, but align your ranking with your labels on head queries.. Thus, don't be afraid of groups of features where each feature applies to a very small fraction of your data, but overall coverage is above 90%. You can use regularization to eliminate the features that apply to too few examples.

Rule #20: Combine and modify existing features to create new features in human-understandable ways.
There are a variety of ways to combine and modify features. Machine learning systems such as TensorFlow allow you to pre-process your data through transformations. The two most standard approaches are "discretizations" and "crosses" .

Discretization consists of taking a continuous feature and creating many discrete features from it. Consider a continuous feature such as age. You can create a feature which is 1 when age is less than 18, another feature which is 1 when age is between 18 and 35, et cetera. Don't overthink the boundaries of these histograms: basic quantiles will give you most of the impact.

Crosses combine two or more feature columns. A feature column, in TensorFlow's terminology, is a set of homogenous features, (e.g. {male, female}, {US, Canada, Mexico}, et cetera). A cross is a new feature column with features in, for example, $\{male, female\} \times \{US, Canada, Mexico\}$. This new feature column will contain the feature (male, Canada). If you are using TensorFlow and you tell TensorFlow to create this cross for you, this (male, Canada) feature will be present in examples representing male Canadians. Note that it takes massive amounts of data to learn models with crosses of three, four, or more base feature columns.

Crosses that produce very large feature columns may overfit. For instance, imagine that you are doing some sort of search, and you have a feature column with words in the query, and you have a feature column with words in the document. You can combine these with a cross, but you will end up with a lot of features (see Rule #21). When working with text there are two alternatives. The most draconian is a dot product. A dot product in its simplest form simply counts the number of common words between the query and the document. This feature can then be discretized. Another approach is an intersection: thus, we will have a feature which is present if and only if the word "pony" is in the document and the query, and another feature which is present if and only if the word "the" is in the document and the query.

Rule #21: The number of feature weights you can learn in a linear model is roughly proportional to the amount of data you have.

There are fascinating statistical learning theory results concerning the appropriate level of complexity for a model, but this rule is basically all you need to know. I have had conversations in which people were doubtful that anything can be learned from one thousand examples, or that you would ever need more than 1 million examples, because they get stuck in a certain method of learning. The key is to scale your learning to the size of your data:

1. If you are working on a search ranking system, and there are millions of different words in the documents and the query and you have 1000 labeled examples, then you should use a dot product between document and query features, TF-IDF, and a half-dozen other highly human-engineered features. 1000 examples, a dozen features.
2. If you have a million examples, then intersect the document and query feature columns, using regularization and possibly feature selection. This will give you millions of features, but with regularization you will have fewer. Ten million examples, maybe a hundred thousand features.
3. If you have billions or hundreds of billions of examples, you can cross the feature columns with document and query tokens, using feature selection and regularization. You will have a billion examples, and 10 million features.

Statistical learning theory rarely gives tight bounds, but gives great guidance for a starting point. In the end, use **Rule #28** to decide what features to use.

Rule #22: Clean up features you are no longer using.
Unused features create technical debt. If you find that you are not using a feature, and that combining it with other features is not working, then drop it out of your infrastructure. You want to keep your infrastructure clean so that the most promising features can be tried as fast as possible. If necessary, someone can always add back your feature.

Keep coverage in mind when considering what features to add or keep. How many examples are covered by the feature? For example, if you have some personalization features, but only 8% of your users have any personalization features, it is not going to be very effective.

At the same time, some features may punch above their weight. For example, if you have a feature which covers only 1% of the data, but 90% of the examples that have the feature are positive, then it will be a great feature to add.

Human Analysis of the System

Before going on to the third phase of machine learning, it is important to focus on something that is not taught in any machine learning class: how to look at an existing model, and improve it. This is more of an art than a science, and yet there are several anti-patterns that it helps to avoid.

Rule #23: You are not a typical end user.
This is perhaps the easiest way for a team to get bogged down. While there are a lot of benefits to fishfooding (using a prototype within your team) and dogfooding (using a prototype within your company), employees should look at whether the performance is correct. While a change which is obviously bad should not be used, anything that looks reasonably near production should be tested further, either by paying laypeople to answer questions on a crowdsourcing platform, or through a live experiment on real users.

There are two reasons for this. The first is that you are too close to the code. You may be looking for a particular aspect of the posts, or you are simply too emotionally involved (e.g. confirmation bias). The second is that your time is too valuable. Consider the cost of 9 engineers sitting in a one hour meeting, and think of how many contracted human labels that buys on a crowdsourcing platform.

If you really want to have user feedback, **use user experience methodologies**. Create user personas (one description is in Bill Buxton's *Designing User Experiences*) early in a process and do usability testing (one description is in Steve Krug's *Don't Make Me Think*) later. User personas involve creating a hypothetical user. For instance, if your team is all male, it might help to design a 35-year old female user persona (complete with user features), and look at the results it generates rather than 10 results for 25-40 year old males. Bringing in actual people to

watch their reaction to your site (locally or remotely) in usability testing can also get you a fresh perspective.

Rule #24: Measure the delta between models.
One of the easiest, and sometimes most useful measurements you can make before any users have looked at your new model is to calculate just how different the new results are from production. For instance, if you have a ranking problem, run both models on a sample of queries through the entire system, and look at the size of the symmetric difference of the results (weighted by ranking position). If the difference is very small, then you can tell without running an experiment that there will be little change. If the difference is very large, then you want to make sure that the change is good. Looking over queries where the symmetric difference is high can help you to understand qualitatively what the change was like. Make sure, however, that the system is stable. Make sure that a model when compared with itself has a low (ideally zero) symmetric difference.

Rule #25: When choosing models, utilitarian performance trumps predictive power.
Your model may try to predict click-through-rate. However, in the end, the key question is what you do with that prediction. If you are using it to rank documents, then the quality of the final ranking matters more than the prediction itself. If you predict the probability that a document is spam and then have a cutoff on what is blocked, then the precision of what is allowed through matters more. Most of the time, these two things should be in agreement: when they do not agree, it will likely be on a small gain. Thus, if there is some change that improves log loss but degrades the performance of the system, look for another feature. When this starts happening more often, it is time to revisit the objective of your model.

Rule #26: Look for patterns in the measured errors, and create new features.
Suppose that you see a training example that the model got “wrong”. In a classification task, this could be a false positive or a false negative. In a ranking task, it could be a pair where a positive was ranked lower than a negative. The most important point is that this is an example that the machine learning system *knows it got wrong* and would like to fix if given the opportunity. If you give the model a feature that allows it to fix the error, the model will try to use it.

On the other hand, if you try to create a feature based upon examples the system doesn’t see as mistakes, the feature will be ignored. For instance, suppose that in Play Apps Search, someone searches for “free games”. Suppose one of the top results is a less relevant gag app. So you create a feature for “gag apps”. However, if you are maximizing number of installs, and people install a gag app when they search for free games, the “gag apps” feature won’t have the effect you want.

Once you have examples that the model got wrong, look for trends that are outside your current feature set. For instance, if the system seems to be demoting longer posts, then add post length. Don’t be too specific about the features you add. If you are going to add post length,

don't try to guess what long means, just add a dozen features and the let model figure out what to do with them (see Rule #21). That is the easiest way to get what you want.

Rule #27: Try to quantify observed undesirable behavior.
Some members of your team will start to be frustrated with properties of the system they don't like which aren't captured by the existing loss function. At this point, they should do whatever it takes to turn their gripes into solid numbers. For example, if they think that too many "gag apps" are being shown in Play Search, they could have human raters identify gag apps. (You can feasibly use human-labelled data in this case because a relatively small fraction of the queries account for a large fraction of the traffic.) If your issues are measurable, then you can start using them as features, objectives, or metrics. The general rule is **"measure first, optimize second".**

Rule #28: Be aware that identical short-term behavior does not imply identical long-term behavior.
Imagine that you have a new system that looks at every doc_id and exact_query, and then calculates the probability of click for every doc for every query. You find that its behavior is nearly identical to your current system in both side by sides and A/B testing, so given its simplicity, you launch it. However, you notice that no new apps are being shown. Why? Well, since your system only shows a doc based on its own history with that query, there is no way to learn that a new doc should be shown.

The only way to understand how such a system would work long-term is to have it train only on data acquired when the model was live. This is very difficult.

Training-Serving Skew

Training-serving skew is a difference between performance during training and performance during serving. This skew can be caused by:

- a discrepancy between how you handle data in the training and serving pipelines, or
- a change in the data between when you train and when you serve, or
- a feedback loop between your model and your algorithm.

We have observed production machine learning systems at Google with training-serving skew that negatively impacts performance. The best solution is to explicitly monitor it so that system and data changes don't introduce skew unnoticed.

Rule #29: The best way to make sure that you train like you serve is to save the set of features used at serving time, and then pipe those features to a log to use them at training time.

Even if you can't do this for every example, do it for a small fraction, such that you can verify the consistency between serving and training (see **Rule #37**). Teams that have made this measurement at Google were sometimes surprised by the results. YouTube home page

switched to logging features at serving time with significant quality improvements and a reduction in code complexity, and many teams are switching their infrastructure as we speak.

Rule #30: Importance weight sampled data, don't arbitrarily drop it!
When you have too much data, there is a temptation to take files 1-12, and ignore files 13-99. This is a mistake: dropping data in training has caused issues in the past for several teams (see Rule #6). Although data that was never shown to the user can be dropped, importance weighting is best for the rest. Importance weighting means that if you decide that you are going to sample example X with a 30% probability, then give it a weight of 10/3. **With importance weighting, all of the calibration properties discussed in Rule #14 still hold**.

Rule #31: Beware that if you join data from a table at training and serving time, the data in the table may change.
Say you join doc ids with a table containing features for those docs (such as number of comments or clicks). Between training and serving time, features in the table may be changed. Your model's prediction for the same document may then differ between training and serving. The easiest way to avoid this sort of problem is to log features at serving time (see Rule #32). If the table is changing only slowly, you can also snapshot the table hourly or daily to get reasonably close data. Note that this still doesn't completely resolve the issue.

Rule #32: Re-use code between your training pipeline and your serving pipeline whenever possible.
Batch processing is different than online processing. In online processing, you must handle each request as it arrives (e.g. you must do a separate lookup for each query), whereas in batch processing, you can combine tasks (e.g. making a join). At serving time, you are doing online processing, whereas training is a batch processing task. However, there are some things that you can do to re-use code. For example, you can create an object that is particular to your system where the result of any queries or joins can be stored in a very human readable way, and errors can be tested easily. Then, once you have gathered all the information, during serving or training, you run a common method to bridge between the human-readable object that is specific to your system, and whatever format the machine learning system expects. **This eliminates a source of training-serving skew**. As a corollary, try not to use two different programming languages between training and serving - that decision will make it nearly impossible for you to share code.

Rule #33: If you produce a model based on the data until January 5th, test the model on the data from January 6th and after.
In general, measure performance of a model on the data gathered after the data you trained the model on, as this better reflects what your system will do in production. If you produce a model based on the data until January 5th, test the model on the data from January 6th. You will expect that the performance will not be as good on the new data, but it shouldn't be radically worse. Since there might be daily effects, you might not predict the average click rate or

conversion rate, but the area under the curve, which represents the likelihood of giving the positive example a score higher than a negative example, should be reasonably close.

Rule #34: In binary classification for filtering (such as spam detection or determining interesting e-mails), make small short-term sacrifices in performance for very clean data.
In a filtering task, examples which are marked as negative are not shown to the user. Suppose you have a filter that blocks 75% of the negative examples at serving. You might be tempted to draw additional training data from the instances shown to users. For example, if a user marks an email as spam that your filter let through, you might want to learn from that.

But this approach introduces sampling bias. You can gather cleaner data if instead during serving you label 1% of all traffic as "held out", and send all held out examples to the user. Now your filter is blocking at least 74% of the negative examples. These held out examples can become your training data.

Note that if your filter is blocking 95% of the negative examples or more, this becomes less viable. Even so, if you wish to measure serving performance, you can make an even tinier sample (say 0.1% or 0.001%). Ten thousand examples is enough to estimate performance quite accurately.

Rule #35: Beware of the inherent skew in ranking problems.
When you switch your ranking algorithm radically enough that different results show up, you have effectively changed the data that your algorithm is going to see in the future. This kind of skew will show up, and you should design your model around it. There are multiple different approaches. These approaches are all ways to favor data that your model has already seen.

1. Have higher regularization on features that cover more queries as opposed to those features that are on for only one query. This way, the model will favor features that are specific to one or a few queries over features that generalize to all queries. This approach can help prevent very popular results from leaking into irrelevant queries. Note that this is opposite the more conventional advice of having more regularization on feature columns with more unique values.
2. Only allow features to have positive weights. Thus, any good feature will be better than a feature that is "unknown".
3. Don't have document-only features. This is an extreme version of #1. For example, even if a given app is a popular download regardless of what the query was, you don't want to show it everywhere[4]. Not having document-only features keeps that simple.

[4] The reason you don't want to show a specific popular app everywhere has to do with the importance of making all the desired apps *reachable*. For instance, if someone searches for "bird watching app", they might download "angry birds", but that certainly wasn't their intent. Showing such an app might improve download rate, but leave the user's needs ultimately unsatisfied.

Rule #36: Avoid feedback loops with positional features.
The position of content dramatically affects how likely the user is to interact with it. If you put an app in the first position it will be clicked more often, and you will be convinced it is more likely to be clicked. One way to deal with this is to add positional features, i.e. features about the position of the content in the page. You train your model with positional features, and it learns to weight, for example, the feature "1st-position" heavily. Your model thus gives less weight to other factors for examples with "1st-position=true". Then at serving you don't give any instances the positional feature, or you give them all the same default feature, because you are scoring candidates *before* you have decided the order in which to display them.

Note that it is important to keep any positional features somewhat separate from the rest of the model because of this asymmetry between training and testing. Having the model be the sum of a function of the positional features and a function of the rest of the features is ideal. For example, don't cross the positional features with any document feature.

Rule #37: Measure Training/Serving Skew.
There are several things that can cause skew in the most general sense. Moreover, you can divide it into several parts:

1. The difference between the performance on the training data and the holdout data. In general, this will always exist, and it is not always bad.
2. The difference between the performance on the holdout data and the "next-day" data. Again, this will always exist. **You should tune your regularization to maximize the next-day performance**. However, large drops in performance between holdout and next-day data may indicate that some features are time-sensitive and possibly degrading model performance.
3. The difference between the performance on the "next-day" data and the live data. If you apply a model to an example in the training data and the same example at serving, it should give you exactly the same result (see Rule #5). Thus, a discrepancy here probably indicates an engineering error.

ML Phase III: Slowed Growth, Optimization Refinement, and Complex Models

There will be certain indications that the second phase is reaching a close. First of all, your monthly gains will start to diminish. You will start to have tradeoffs between metrics: you will see some rise and others fall in some experiments. This is where it gets interesting. Since the gains are harder to achieve, the machine learning has to get more sophisticated.

A caveat: this section has more blue-sky rules than earlier sections. We have seen many teams go through the happy times of Phase I and Phase II machine learning. Once Phase III has been reached, teams have to find their own path.

Rule #38: Don't waste time on new features if unaligned objectives have become the issue.

As your measurements plateau, your team will start to look at issues that are outside the scope of the objectives of your current machine learning system. As stated before, if the product goals are not covered by the existing algorithmic objective, you need to change either your objective or your product goals. For instance, you may optimize clicks, plus-ones, or downloads, but make launch decisions based in part on human raters.

Rule #39: Launch decisions are a proxy for long-term product goals.

Alice has an idea about reducing the logistic loss of predicting installs. She adds a feature. The logistic loss drops. When she does a live experiment, she sees the install rate increase. However, when she goes to a launch review meeting, someone points out that the number of daily active users drops by 5%. The team decides not to launch the model. Alice is disappointed, but now realizes that launch decisions depend on multiple criteria, only some of which can be directly optimized using ML.

The truth is that the real world is not dungeons and dragons: there are no "hit points" identifying the health of your product. The team has to use the statistics it gathers to try to effectively predict how good the system will be in the future. They need to care about engagement, 1 day active users (DAU), 30 DAU, revenue, and advertiser's return on investment. These metrics that are measureable in A/B tests in themselves are only a proxy for more long-term goals: satisfying users, increasing users, satisfying partners, and profit, which even then you could consider proxies for having a useful, high quality product and a thriving company five years from now.

The only easy launch decisions are when all metrics get better (or at least do not get worse). If the team has a choice between a sophisticated machine learning algorithm, and a simple heuristic, if the simple heuristic does a better job on all these metrics, it should choose the heuristic. Moreover, there is no explicit ranking of all possible metric values. Specifically, consider the following two scenarios:

Experiment	Daily Active Users	Revenue/Day
A	1 million	$4 million
B	2 million	$2 million

If the current system is A, then the team would be unlikely to switch to B. If the current system is B, then the team would be unlikely to switch to A. This seems in conflict with rational behavior: however, predictions of changing metrics may or may not pan out, and thus there is a large risk involved with either change. Each metric covers some risk with which the team is concerned.

Moreover, no metric covers the team's ultimate concern, "where is my product going to be five years from now"?

Individuals, on the other hand, tend to favor one objective that they can directly optimize. Most machine learning tools favor such an environment. An engineer banging out new features can get a steady stream of launches in such an environment. There is a type of machine learning, multi-objective learning, which starts to address this problem. For instance, one can formulate a constraint satisfaction problem that has lower bounds on each metric, and optimizes some linear combination of metrics. However, even then, not all metrics are easily framed as machine learning objectives: if a document is clicked on or an app is installed, it is because that the content was shown. But it is far harder to figure out why a user visits your site. How to predict the future success of a site as a whole is AI-complete, as hard as computer vision or natural language processing.

Rule #40: Keep ensembles simple.

Unified models that take in raw features and directly rank content are the easiest models to debug and understand. However, an ensemble of models (a "model" which combines the scores of other models) can work better. **To keep things simple, each model should either be an ensemble only taking the input of other models, or a base model taking many features, but not both.** If you have models on top of other models that are trained separately, then combining them can result in bad behavior.

Use a simple model for ensembling that takes only the output of your "base" models as inputs. You also want to enforce properties on these ensemble models. For example, an increase in the score produced by a base model should not decrease the score of the ensemble. Also, it is best if the incoming models are semantically interpretable (for example, calibrated) so that changes of the underlying models do not confuse the ensemble model. Also, **enforce that an increase in the predicted probability of an underlying classifier does not decrease the predicted probability of the ensemble**.

Rule #41: When performance plateaus, look for qualitatively new sources of information to add rather than refining existing signals.

You've added some demographic information about the user. You've added some information about the words in the document. You have gone through template exploration, and tuned the regularization. You haven't seen a launch with more than a 1% improvement in your key metrics in a few quarters. Now what?

It is time to start building the infrastructure for radically different features, such as the history of documents that this user has accessed in the last day, week, or year, or data from a different property. Use wikidata entities or something internal to your company (such as Google's knowledge graph). Use deep learning. Start to adjust your expectations on how much return you

expect on investment, and expand your efforts accordingly. As in any engineering project, you have to weigh the benefit of adding new features against the cost of increased complexity.

Rule #42: Don't expect diversity, personalization, or relevance to be as correlated with popularity as you think they are.
Diversity in a set of content can mean many things, with the diversity of the source of the content being one of the most common. Personalization implies each user gets their own results. Relevance implies that the results for a particular query are more appropriate for that query than any other. Thus all three of these properties are defined as being different from the ordinary.

The problem is that the ordinary tends to be hard to beat.

Note that if your system is measuring clicks, time spent, watches, +1s, reshares, et cetera, you are measuring the **popularity** of the content. Teams sometimes try to learn a personal model with diversity. To personalize, they add features that would allow the system to personalize (some features representing the user's interest) or diversify (features indicating if this document has any features in common with other documents returned, such as author or content), and find that those features get less weight (or sometimes a different sign) than they expect.

This doesn't mean that diversity, personalization, or relevance aren't valuable. As pointed out in the previous rule, you can do post-processing to increase diversity or relevance. If you see longer term objectives increase, then you can declare that diversity/relevance is valuable, aside from popularity. You can then either continue to use your post-processing, or directly modify the objective based upon diversity or relevance.

Rule #43: Your friends tend to be the same across different products. Your interests tend not to be.
Teams at Google have gotten a lot of traction from taking a model predicting the closeness of a connection in one product, and having it work well on another. Your friends are who they are. On the other hand, I have watched several teams struggle with personalization features across product divides. Yes, it seems like it should work. For now, it doesn't seem like it does. What has sometimes worked is using raw data from one property to predict behavior on another. Also, keep in mind that even knowing that a user has a history on another property can help. For instance, the presence of user activity on two products may be indicative in and of itself.

Related Work

There are many documents on machine learning at Google as well as externally.

- Machine Learning Crash Course: an introduction to applied machine learning

- Machine Learning: A Probabilistic Approach by Kevin Murphy for an understanding of the field of machine learning
- Practical Advice for the Analysis of Large, Complex Data Sets: a data science approach to thinking about data sets.

- Deep Learning by Ian Goodfellow et al for learning nonlinear models
- Google paper on technical debt, which has a lot of general advice.
- Tensorflow Documentation

Acknowledgements

Thanks to David Westbrook, Peter Brandt, Samuel Ieong, Chenyu Zhao, Li Wei, Michalis Potamias, Evan Rosen, Barry Rosenberg, Christine Robson, James Pine, Tal Shaked, Tushar Chandra, Mustafa Ispir, Jeremiah Harmsen, Konstantinos Katsiapis, Glen Anderson, Dan Duckworth, Shishir Birmiwal, Gal Elidan, Su Lin Wu, Jaihui Liu, Fernando Pereira, and Hrishikesh Aradhye for many corrections, suggestions, and helpful examples for this document. Also, thanks to Kristen Lefevre, Suddha Basu, and Chris Berg who helped with an earlier version. Any errors, omissions, or (gasp!) unpopular opinions are my own.

Appendix

There are a variety of references to Google products in this document. To provide more context, I give a short description of the most common examples below.

YouTube Overview

YouTube is a streaming video service. Both YouTube Watch Next and YouTube Home Page teams use ML models to rank video recommendations. Watch Next recommends videos to watch after the currently playing one, while Home Page recommends videos to users browsing the home page.

Google Play Overview

Google Play has many models solving a variety of problems. Play Search, Play Home Page Personalized Recommendations, and ‘Users Also Installed’ apps all use machine learning.

Google Plus Overview

Google Plus uses machine learning in a variety of situations: ranking posts in the “stream” of posts being seen by the user, ranking “What’s Hot” posts (posts that are very popular now), ranking people you know, et cetera.

Paper SAS2360-2016

Best Practices for Machine Learning Applications

Brett Wujek, Patrick Hall, and Funda Güneș

SAS Institute Inc.

ABSTRACT

Building representative machine learning models that generalize well on future data requires careful consideration both of the data at hand and of assumptions about the various available training algorithms. Data are rarely in an ideal form that enables algorithms to train effectively. Some algorithms are designed to account for important considerations such as variable selection and handling of missing values, whereas other algorithms require additional preprocessing of the data or appropriate tweaking of the algorithm options. Ultimate evaluation of a model's quality requires appropriate selection and interpretation of an assessment criterion that is meaningful for the given problem. This paper discusses many of the most common issues faced by machine learning practitioners and provides guidance for using these powerful algorithms to build effective models.

INTRODUCTION

The study of machine learning algorithms often focuses on minimizing the error over an unknown data-generating distribution (a significant departure from classical statistics), quantifying the complexity of such algorithms, and establishing the bounds of their error (Vapnik 1996). This paper seeks to make no comment on the mathematical and statistical study of algorithms, but rather to guide the machine learning practitioner through the common steps of data preparation, model training, and model deployment.

When applied judiciously, machine learning solutions deliver significant value to a business by extracting previously hidden knowledge from stored or streaming data. Consider the data to be a stockpile of building material and supplies, and machine learning algorithms to be the powerful tools that can help construct a valuable structure from that stockpile. When you extend this analogy to incorporate skilled craftsmen operating these tools, the final product still depends heavily on the existence of a well-defined plan, adherence to sound building practices, and avoidance of mistakes at critical junctures in the process.

Using machine learning effectively and successfully boils down to a combination of knowledge, awareness, and ultimately taking a scientific approach to the overall process. Machine learning is a fundamental element of data science. In that regard, there exists an implied commitment to taking a scientific approach when establishing and executing a machine learning solution (Donoho 2015). Blindly supplying data to these powerful algorithms with little forethought given to the nature of the data, the strengths/weaknesses and training options of the algorithms, appropriate assessment, and deployment desires will likely result in models that do not adequately address your business problem. Quite simply, better modeling practices lead to better business decisions.

This paper examines best practices and highlights common mistakes and oversights that are often witnessed in the various phases of devising a machine learning application: data preparation, training, and deployment. Table 1 outlines some of the most common challenges faced during implementation of a machine learning application along with corresponding suggested best practices. The remainder of this paper provides detailed descriptions and explanations for each of these issues.

Topic	Common Challenges	Suggested Best Practice
Data Preparation		
Data collection	• Biased data • Incomplete data • The curse of dimensionality • Sparsity	• Take time to understand the business problem and its context • Enrich the data • Dimension-reduction techniques • Change representation of data (e.g., COO)
"Untidy" data	• Value ranges as columns • Multiple variables in the same column • Variables in both rows and columns	Restructure the data to be "tidy" by using the melt and cast process
Outliers	• Out-of-range numeric values and unknown categorical values in score data • Undue influence on squared loss functions (e.g., regression, GBM, *k*-means)	• Robust methods (e.g., Huber loss function) • Discretization (binning) • Winsorizing
Sparse target variables	• Low primary event occurrence rate • Overwhelming preponderance of zero or missing values in target	• Proportional oversampling • Inverse prior probabilities • Mixture models
Variables of disparate magnitudes	• Misleading variable importance • Distance measure imbalance • Gradient dominance	Standardization
High-cardinality variables	• Overfitting • Unknown categorical values in holdout data	• Discretization (binning) • Weight of evidence • Leave-one-out event rate
Missing data	• Information loss • Bias	• Discretization (binning) • Imputation • Tree-based modeling techniques
Strong multicollinearity	Unstable parameter estimates	• Regularization • Dimension reduction
Training		
Overfitting	High-variance and low-bias models that fail to generalize well	• Regularization • Noise injection • Partitioning or cross validation
Hyperparameter tuning	Combinatorial explosion of hyperparameters in conventional algorithms (e.g., deep neural networks, super learners)	• Local search optimization, including genetic algorithms • Grid search, random search
Ensemble models	• Single models that fail to provide adequate accuracy • High-variance and low-bias models that fail to generalize well	• Established ensemble methods (e.g., bagging, boosting, stacking) • Custom or manual combinations of predictions
Model Interpretation	Large number of parameters, rules, or other complexity obscures model interpretation	• Variable selection by regularization (e.g., L1) • Surrogate models • Partial dependency plots, variable importance measures
Computational resource exploitation	• Single-threaded algorithm implementations • Heavy reliance on interpreted languages	• Train many single-threaded models in parallel • Hardware acceleration (e.g. SSD, GPU) • Low-level, native libraries • Distributed computing, when appropriate
Deployment		
Model deployment	Trained model logic must be transferred from a development environment to an operational computing system to assist in organizational decision-making processes	• Portable scoring code or scoring executables • In-database scoring • Web service scoring
Model decay	• Business problem or market conditions have changed since the model was created • New observations fall outside domain of training data	• Monitor models for decreasing accuracy • Update/retrain models regularly • Champion-challenger tests • Online updates

Table 1. Best Practices for Common Machine Learning Challenges

PREPARATION

Effective machine learning models are built on a foundation of well-prepared data. The importance of this phase cannot be overstated. In fact, it is commonly proclaimed that 80% of the time spent in devising a successful machine learning application is spent in data preparation (Dasu and Johnson 2003). Data preparation is not strictly about appropriately transforming and cleaning existing data; it also includes a good understanding of the features that need to be considered and ensuring that the data at hand are appropriate in the first place. Shortcuts in data preparation will shortchange your models. As they say, "garbage in, garbage out." Take the time to cultivate your data, and be wary of the common challenges described in this section.

ENSURING SUFFICIENT AND APPROPRIATE DATA

Do You Have the Right Data?

Before you simply throw your data into a modeling algorithm, before you even start to perform transformations to clean and shape your data to a form more suitable for modeling, start by asking yourself "do I have the *right* data to answer the business question being asked?" Just because you have a lot of data doesn't mean you have the *right* data. Ensure that the data are representative of the entire domain of interest—that the observations cover the anticipated range of values when this model is used in production. Beware of the perils of extrapolation, and understand that machine learning algorithms build models that are representative of the available training samples. The algorithms can be very inaccurate outside of that subspace, as shown in the example of various neural networks that are trained to the data shown in Figure 1 from Lohninger (1999). If you can collect more data to account for the domain of anticipated application of your model, your resulting model will probably be more effective. In addition to understanding the domain of the input variables, make certain that the target values you observe in the data set to be used for training include values representative of what you expect when you deploy the model. In particular, if your target is nominal, then the trained model will only be able to predict the specific values in your training set.

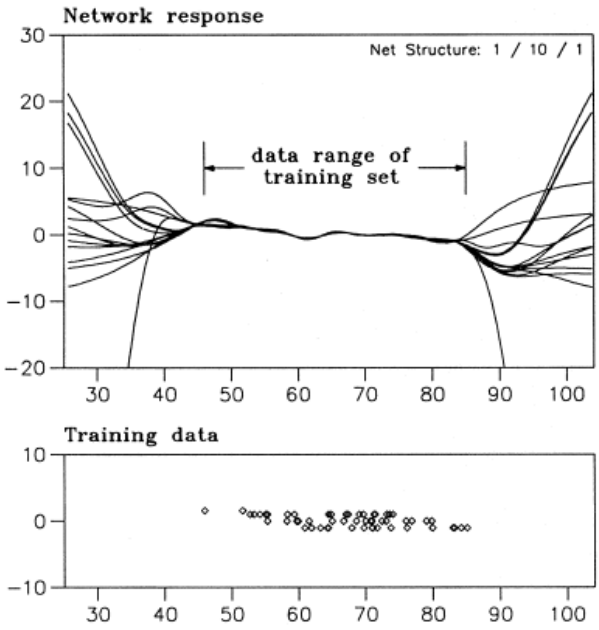


Figure 1. Highly Inaccurate Model Predictions from Extrapolation (Lohninger 1999)

Data Partitioning and Leakage

In typical machine learning tasks, data are divided into different sets (partitions): some data for training the model and some data for evaluating the model. It is critical that all transformations that are used to prepare the training data are applied to any validation, test, or other holdout data. It is also critically important that information from holdout data does not leak into the training data. Information leakage can occur in many ways and can potentially lead to overfitting or overly optimistic error measurements. For example, think of taking a mean or median across your data before partitioning and then later using this mean or median to impute missing values across all partitions of your data. In this case, your training data would be aware of information from your validation, test, or holdout partitions. To avoid this type of leakage, values for imputation and other basic transformations should be generated from only the training data or within each partition independently. Feature extraction can also lead to leakage. Consider principal component analysis (PCA), where features are created by decomposing a covariance or correlation matrix. If that covariance or correlation matrix is generated from all your data and then the derived principal component features or scores are used across all partitions of your data, your training data will be contaminated by information from other partitions. For more complex transformations such as feature extraction and binning, it is best to develop formulas or rules from the training data that can be applied to other partitions of data in order to generate the required features.

Accounting for Rare Events

It is also important to be mindful of your target of interest and understand whether it can be characterized as a rare event relative to your total number of samples. Applications such as detecting fraudulent activity must take special steps to ensure that the data used to train the model include a representative number of fraudulent samples in in order to capture the event sufficiently (for example, 1 out of every 1,000 credit card transactions is fraudulent). Fitting a model to such data without accounting for the extreme imbalance in the occurrence of the event will provide you with a model that is extremely accurate at telling you absolutely nothing of value. Special sampling methods that modify an imbalanced data set are commonly used in order to provide a more balanced distribution when modeling rare events (He and Garcia 2009). These techniques include oversampling and undersampling methods.

In oversampling, the instances of the rare event class are increased by adding randomly selected instances from the existing rare events to achieve a more balanced overall distribution (usually close to 50%). This enables your model to more easily detect and express the relationship between features and the target of interest. Just realize that doing so requires you to adjust your model predictions to account for the unnatural bias you have introduced to the rare event, and that the predicted event probabilities and the false positive and false negative rates will not be accurate. In SAS® Enterprise Miner™ you can adjust your model predictions by defining a target profile and using weighting or offset methods that incorporate the probability of the event as observed in the complete population (the prior probabilities) versus the proportion of the event actually witnessed in the training set—that is, the posterior probabilities in the oversampled data set (Wielenga 2007). Oversampling has the additional benefit of enabling you to train on a more reasonably sized sample rather than requiring a very large sample simply to ensure that more rare event observations are included. A more reasonably sized sample results in faster training, allowing more time to experiment with different models. One problem with oversampling is that it can lead to overfitting because an excessive amount of replicated data for the rare class is used (Mease, Wyner, and Buja 2007).

In contrast to oversampling, undersampling removes instances from the majority class in order to adjust the balance of the original data set. For undersampling, the problem is relatively obvious—removing examples from the majority class might cause the classifier to miss important information from that class. To overcome this problem, you can use informed undersampling algorithms such as EasyEnsemble and BalanceCascade. Both of these techniques are very straightforward. The EasyEnsemble method independently samples several subsets from the majority class and develops multiple classifiers that are trained on the combination of each subset with the minority class data. In this way, EasyEnsemble can be considered to be an unsupervised learning algorithm that explores the majority class data by using independent random sampling with replacement. On the other hand, the BalanceCascade algorithm takes a supervised learning approach that develops an ensemble of classifiers to systematically select which majority class examples to undersample (Zhang and Mani 2003).

Another way to deal with rare events is to fit zero-inflated regression models, which are special cases of finite mixture models. Zero-inflated models view data as a mixture of a constant distribution (which always generates zero counts) and another distribution (which always generates nonzero counts), such as a Poisson, binomial, or multinomial distribution. Using zero-inflated models to model rare events enables you to generate a mixing probability that tells what percentage of data comes from a constant distribution and what percentage comes from the other distribution. You can also use these models for clustering rare events by assigning each observation to a component that has the highest posterior probability. In addition, handling rare events in the regression modeling framework is advantageous because it generates interpretable models.

TENDING TO THE DATA STRUCTURE AND FEATURE CONTENT

Assuming you have the *right* data, it is important to examine and understand the values of the features, both within each individual feature and across the set of features. After you ensure that the structure of your data is appropriate for your machine learning application, you should look for large discrepancies in magnitude among interval variables, high cardinality among nominal variables, and outliers and missing values within the values of each feature.

Data Structure

The original data set presented to you might be in a form that is ill-suited for applying machine learning algorithms to build models or identify patterns. If the data are unstructured or semi-structured (such as text from logs or information in XML format), you need to transform the data set in some way to produce a structured data set that is more suitable for modeling. But even if you have structured data, you should ensure that the rows represent what you consider to be observations and that the columns represent feature or target variables. This type of representation is referred to as "tidy data" in (Wickham 2014).

Take the time to transform your data set into a tidy data set so that you start the machine learning application with "a standardized way to link the structure of a dataset (its physical layout) with its semantics (its meaning)" (Wickham 2014). Wickham defines the tidying process to include the following techniques:

- **Melting:** When column headers contain values rather than true variables, melt (stack) those columns into two columns that are represented across multiple rows: one column for the original column header (which is really a value) and the other for the value from the original column (often a frequency). For example, if a data set has columns for income ranges such as <$10k, $10k–$30k, and so on, with the values in those columns being the frequency of that income range for each observation (such as incomes within states), melt all these columns into two columns for Income Range and Frequency (such that you have multiple rows for each state).
- **String-splitting:** When a column actually contains multiple pieces of information, split it into multiple columns. For example, a column that contains M-20, F-35, M-42, and so on should be split into gender and age variables in two separate columns. Note that the melting process might result in such a column, so you might need to split after melting.
- **Casting:** When a column actually contains names of variables instead of values, cast (unstack) the column into multiple columns for each of the values (which are actually variables). Casting is the inverse of the melting process.

The ability of a machine learning algorithm to train an effective model for your business problem is highly dependent on the structure of your data set and on ensuring that features and observations are properly defined. Tidying your data set by melting, string-splitting, and casting can help you obtain more informative features and therefore increase model accuracy.

Standardization

Your data set most likely has features whose values are significantly different in magnitude and range. This disparity can degrade the performance of many machine learning algorithms, particularly those that distinguish observations and feature effectiveness based on a distance measurement (such as *k*-means clustering or nearest neighbor approaches), those that use numerical gradient information in their solution (such as neural networks and support vector machines), those that depend on a measure of the variance (such as principal component analysis), and those that penalize variables based on the size of their corresponding parameters (such as penalized regression techniques). The objective functions used for solving many of these machine learning algorithms can be dominated by the features that have large variance relative to other input features, preventing the model from being able to learn the relationship with the other features. For example, penalized regression techniques (which penalize the objective based on the magnitudes of the coefficients) disproportionately penalize the higher magnitude terms. However, some algorithms are scale invariant; for example, decision trees might bin inputs before splitting or make splitting decisions based on feature values independently of other features. But in general you need to account for the disparity in the magnitudes and ranges of values.

To mitigate the detrimental effect of widely varying magnitudes and ranges, you should transform your interval feature values to be on a similar scale, commonly standardizing to have mean 0 and variance 1 (z-scoring). Note that if your data set is sparse (that is, it contains a high percentage of 0 values), standardizing in this manner would destroy this sparseness and thus lose the ability to use methods that process sparse data very efficiently. In that case, a good alternative would be to simply scale the data based on the range or maximum absolute value of each feature. Also, if you have extreme outliers (in the far tails of the distribution) or nonnormally distributed features, z-scoring might not bring all values into scale. For extreme outliers, range standardization could compact all the other values into a very small percentage of the range and render them ineffectual in modeling; you might need to address outliers before standardizing. However you choose to scale your data, realize that you need to apply this same scaling process to new observations that are scored with any subsequently generated model in the future.

Standardization serves to put all variables on a level playing field, so to speak. However, it does not account for other issues within a set of individual variable values that must be considered. Some of the more common issues are covered in the following sections.

Managing Outliers

When you build models to predict future behavior or identify patterns, ideally you have a data set whose feature values are representative of typical observations. However, you will often find outliers in your data—observations that are very distinct from the others in one or more of the feature values. This can be true of interval variables that have values that occur in the extreme tails or as abnormal spikes in the distribution, and of nominal variables that have values that occur only in an extremely small percentage of the observations. Although outliers can certainly be very informative and can identify anomalies that deserve special attention, they can be quite detrimental to training an effective generalizable model. Some algorithms are more robust at dealing with outliers, as described in the following list:

- Supervised learning algorithms that use a squared-loss function to determine the parameters that best fit the training data are heavily influenced by outliers. For example, gradient boosting algorithms add large weights to observations that are considered to be hard cases.
- If you are clustering your data, the effect of outliers depends on the clustering method. For example:
    - *k*-means clustering can be quite sensitive to outliers because it tries to minimize the sum of squared distances from cluster member points to the cluster means; a large deviation caused by an outlier receives a lot of weight.
    - Density-based clustering (such as DBSCAN) tends to be less sensitive to outliers, usually identifying them as individual outlying clusters.
    - Hierarchical clustering simply assigns outliers to their own clusters.

    In general, outliers drag clusters artificially toward the outside of your feature space, either as individual clusters or by morphing clusters to incorporate the outliers.

Fortunately, outliers can usually be detected by some simple initial data exploration. You should make outlier management a standard part of your data exploration and preparation process before modeling. First, determine whether an outlying value is simply an invalid or erroneous entry that can be disregarded. If you have determined that an outlier provides no valuable information, it is acceptable to simply filter it out. However, if you determine that outliers might represent some real but rare relationship or if you think that the information from the other features in those observations is too valuable to discard, then include them in your model only after dealing with them appropriately by using one of the following techniques:

- For categorical variables, you can bin the values into an "Other" category. For more information, see the next section.
- For interval variables, you can winsorize the values, setting them to the lowest or highest non-outlier value (depending on which side of the distribution the outlier lies) or forcing them to be no greater than three standard deviations from the mean. Either of these approaches retains the observation, which might contain other valuable information from other features.
- For algorithms that incorporate a loss function to direct the training process, you can use a Huber loss function (Huber 1964), which greatly reduces the impact of outliers on the calculation of the loss.

Binning

Binning is the process of discretizing numerical variables into fewer categorical counterparts. For example, "age" variables are often binned into categories such as 20–39, 40–59, and 60–79. Building a model against each individual age probably does not provide any more information for a model than building it against age groups; binning reduces the complexity of mapping the feature values to the response. Binning tends to generate a more effective predictive model and can make the model easier to interpret. On the other hand, binning can also cause issues such as loss of power, because binning increases the number of model parameters to estimate. However, if you have big data, binning can be very beneficial, especially in difficult predictive modeling problems where many algorithms fail.

In particular, binning can simplify and improve accuracy of predictive models by doing the following:

- decreasing the impact of outliers
- enabling you to incorporate missing values
- managing high-cardinality variables (those with too many overall levels)
- reducing the noise or nonlinearity

Binning is known to work well to reduce noise (increase signal-to-noise ratio) and hence helps produce better predictive models for "messy" data that have many missing values, outliers, high-cardinality variables, nonlinearities between the input variables and the target, and skewed distributions of numeric input variables.

Missing Values

Missing values can be theoretically and practically problematic for many machine learning tasks, especially when missing values are present in the target variable. This section addresses only the more common scenario of missing values in input variables. When faced with missing values in input variables, practitioners must consider whether missing values are distributed randomly or whether missingness is somehow predictive of the target. If missing values appear at random in the input data, the input rows that contain missing values can be dropped from the analysis without introducing bias into the model. However, such a *complete case analysis* can remove a tremendous amount of information from the training data and reduce the predictive accuracy of the model. Missingness can actually be predictive: retaining information that is associated with missing values, including the missing values themselves, can actually increase the predictive accuracy of a model. The following list describes practices for accounting for missingness in training a machine learning model and describes how missing values must also be handled when scoring new data.

- **Naïve Bayes:** Naïve Bayes models elegantly handle missing values for training and scoring by computing the likelihood based on the observed features. Because of conditional independence between the features, naïve Bayes ignores a feature only when its value is missing. Thus, you don't need to handle missing values before fitting a naïve Bayes model unless you believe the missingness is not at random. For efficiency reasons, some implementations of naïve Bayes remove entire rows from the training process whenever a missing value is encountered. When missing is treated as a categorical level, infrequent missing values in new data can be problematic when they are not present in training data, because the missing level will have had no probability associated with it during training. You can solve this problem by ignoring the offending feature in the likelihood computation when scoring.

- **Decision trees:** In general**,** imputation, missing markers, binning, and special scoring considerations are not required for missing values when you use a decision tree. Decision trees allow for the elegant and direct use of missing values in two common ways:
  - When a splitting rule is determined, missing can be considered to be a valid input value, and missing values can either be placed on the side of the splitting rule that makes the best training prediction or be assigned to a separate branch in a split.
  - Surrogate rules can be defined to allow the tree to split on a surrogate variable when a missing value is encountered. For example, a surrogate rule could be defined that allows a decision tree to split on the state variable when the zip code variable is missing.

- **Missing markers:** Missing markers are binary variables that record whether the value of another variable is missing. They are used to preserve information about missingness so that missingness can be modeled. Missing markers can be used in a model to replace the original corresponding variable with missing values, or they can be used in a model alongside an imputed version of the original variable.

- **Imputation:** Imputation refers to replacing a missing value with information that is derived from nonmissing values in the training data. Simple imputation schemes include replacing a missing value in a particular input variable with the mean or mode of that variable's nonmissing values. For nonnormally distributed variables or variables that have a high proportion of missing values, simple mean or mode imputation can drastically alter a variable's distribution and negatively impact predictive accuracy. Even when variables are normally distributed and contain a low proportion of missing values, creating missing markers and using them in the model alongside the new, imputed variables is a suggested practice. Decision trees can also be used to derive imputed values. A decision tree can be trained using a variable that has missing values as its target and all the other variables in the data set as inputs. In this way, the decision tree can learn plausible replacement values for the missing values in the temporary target variable. This approach requires one decision tree for every input variable that has missing values, so it can become computationally expensive for large, dirty training sets. More sophisticated imputation approaches, including multiple imputation (MI), should be considered for small data sets (Rubin 1987).

- **Binning:** Interval input variables that have missing values can be discretized into a number of bins according to their original numeric values in order to create new categorical, nominal variables. Missing values in the original variable can simply be added to an additional bin in the new variable. Categorical input variables that have missing values can be assigned to new categorical nominal variables that have the same categorical levels as the corresponding original variables plus one new level for missing values. Because binning introduces additional nonlinearity into a predictive model and can be less damaging to an input variable's original distribution than imputation, binning is generally considered acceptable, if not beneficial, until the binning process begins to contribute to overfitting. However, you might not want to use binning if the ordering of the values in a particular input variable is important, because the ordering information is changed or erased by introducing a missing bin into the otherwise ordered values.

- **Scoring missing data:** If a decision tree or decision tree ensemble is used in training, missing values in new data will probably be scored automatically according to the splitting rules or the surrogate rules of the trained tree (or trees). If another type of algorithm was trained, then missing values in new data must be processed in the exact manner in which they were processed in the training data before the model was trained.

DIMENSIONALITY

Falling Prey to the Curse of Dimensionality

A common sentiment is that having more information will enable you to make better decisions. However, this belief is based on the assumption that you will be able to easily discern the meaningful information from the trivial and efficiently process the information. In reality, the more items of independent information you have, the more complex and costly your decision-making process becomes.

Although high dimensionality can result from the seemingly innocent desire to incorporate more features into your model, sometimes high dimensionality is simply the nature of the problem, such as in bioinformatics (DNA microarrays that are used to analyze tens of thousands of genes), multimedia data analysis with millions of pixels, and text documents that are represented by high-dimensional frequency count vectors.

The challenges that arise as you attempt to incorporate more features into your model are best described through a phenomenon known as the *curse of dimensionality* (Bellman 1957). Increasing the number of features increases the volume of the feature space exponentially; in turn, this higher-dimensional space requires exponentially more data points to sufficiently fill that space in order to ensure that combinations of feature values are accounted for. Figure 2 depicts the increased sparsity of the data with a very simple example of eight observations in one, two, and three dimensions. In one dimension, the observations sufficiently cover the domain, in two dimensions they still cover it fairly reasonably, but in three dimensions you start to see the sparsity of the data in the overall domain.

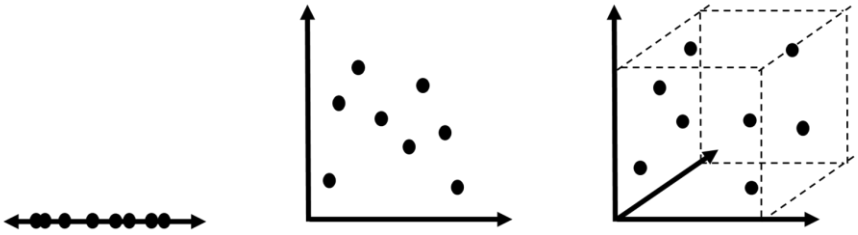


Figure 2. Increased Data Sparsity as Dimensions (Features) Are Added

It is difficult to imagine beyond three dimensions, but as the dimensionality further increases, a greater percentage of the available data reside in the “corners” of the feature space, which are much more difficult to classify than observations in the ever-shrinking “center” of the feature space. An illustrative example of this is provided in Spruyt (2014).

Consider *k*-means clustering as an example of the adverse effect of higher dimensionality. High dimensionality makes clustering difficult because having many dimensions means that all the observations are relatively "far away" from each other. It is difficult to know what a cluster is when all of the observations within a cluster are distant from one another. Generally, observations within a cluster are assumed to be close to one another; but in a high-dimensional space a cluster might be more like a dispersed cloud of loosely related points.

In general, higher dimensionality tends to stimulate overfitting, can lead to greater confounding among the feature effects, and renders visualization of the original problem domain impractical. Also, the amount of time and memory required to process additional dimensions generally diminishes the efficiency of the modeling algorithm.

Dimension Reduction

For the most efficient and effective predictive model, it is best to reduce the set of features (inputs to the model) to include those that are most relevant and have a nontrivial impact on the target. This is referred to as feature selection. The first course of action, if feasible, should be to directly inspect and evaluate the features to ensure that the values corresponding to that feature are reasonable, and to possibly identify an obvious subset of features for modeling, or at least determine features that can be excluded. If you have some domain knowledge, you might be able to use common sense to filter out some of the features in an ad-hoc manner.

Beyond initial manual filtering, the common approach to reducing dimensionality is to perform feature engineering, which can be categorized as either feature selection or feature extraction:

- **Feature selection** is the process of determining which features have the greatest impact on your model and downselecting to include only those features as inputs. This can be accomplished through one of the following types of techniques:
  - Filtering techniques, which assess each feature's impact through **mutual information criterion (MIC)**, **information gain**, **chi-square measure**, **odds ratio, or R-square measure**. These techniques neglect any interaction effects among features.
  - Wrapper techniques such as **LASSO** (Tibshirani 1996), **elastic net** (Zou and Hastie 2005), and **forward/backward/stepwise regression** techniques that iteratively determine which terms (including interactions) to include in the model.
  - Embedded techniques, which incorporate feature selection as an integral part of the training. One example is a **decision tree**, for which the splitting rules are based on determining the most influential variables to direct the splitting at each node (for example, the variables that best preserve the purity of the split).

- **Feature extraction** is the process of transforming the existing features into a lower-dimensional space, typically generating new features that are composites of the existing features. There are a number of techniques that reduce dimensionality through such a transformation process, including the following:
  - **Principal component analysis (PCA)** and **singular value decomposition (SVD)** are unsupervised feature transformations that strive to project the original features onto new orthogonal dimensions of maximum variance. The generated or extracted features that are the top contributors to variance can be preserved as a reduced set of features to be used as inputs for training the model. Standard PCA uses a linear combination of the existing features to achieve this; however, **kernel PCA** (Schölkopf, Smola, and Müller 1998) can be used to construct nonlinear mappings from the original features to new orthogonal dimensions of maximum variance. Although PCA can be performed via eigendecomposition of the square and symmetrical covariance matrix, it is often advantageous to compute the SVD of the rectangular data matrix instead. It is then trivial to obtain the principal components from the SVD, and you avoid the intermediate step of computing the covariance matrix. In addition, greater interpretability of the final feature set can be achieved using **sparse PCA** (Zou, Hastie, and Tibshirani 2005), which enforces sparsity constraints to achieve the linear combinations in only a few input variables, although orthogonality is lost.

  - **Nonnegative matrix factorization (NMF)** (Lee and Seung 2000) is another important unsupervised feature transformation. As in PCA (and SVD if you perform some algebraic manipulation), the original data matrix is decomposed into the product of two factor matrices. The distinctive feature of NMF is that all the entries of the factor matrices are constrained to be nonnegative. This constraint aids interpretability of the factors in many settings (for example, text analysis) where the factors can be thought of as probabilities.

  - **Autoencoding neural networks** extract a highly representative set of nonlinear features from the bottleneck layer of a specialized network.

An obvious drawback to feature extraction is that the actual inputs to the model are no longer meaningful with respect to the business problem. However, you can simply consider this another transformation of the original inputs to be provided to the model, something that must be accounted for as part of the scoring process when the model is deployed.

An excellent overview of feature engineering strategies is provided in (Cunningham 2007).

Whether you perform feature selection or feature extraction, your ultimate goal is to include the subset of features that describe most, but not all, of the variance and to reduce the signal-to-noise ratio in your data. Although intuition would tell you that elimination of features equates to a loss of information, in the end this loss is compensated for by the ability of the model to more accurately map the remaining features to the target in a lower-dimensional space. The result is simpler models, shorter training times, improved generalization, and a greater ability to visualize the feature space. As a side note, it is good practice to perform feature engineering before and after any imputation you might implement and compare the results; the effect of providing artificial values for values that were missing might cause different features to be selected.

Some high-dimensional data sets require special attention to perform feature extraction efficiently; one example is a data set of user ratings for items (such as movies) in which each column represents an item and each row is a user (or vice versa). Data sets such as this are very sparse and can be reduced by converting to a **sparse data representation** such as coordinate list (COO) format, in which only nonzero items (or items that actually have values) are preserved in a data set that has three columns for row, column, and value (user, item, and rating in this example). Model training time can be reduced exponentially simply by realizing that you have sparse data, converting the data set to a format such as COO, and using feature extraction techniques that are optimized for sparse data representations.

UNDERSTAND THE EXPECTATIONS OF THE MODEL CONSUMER

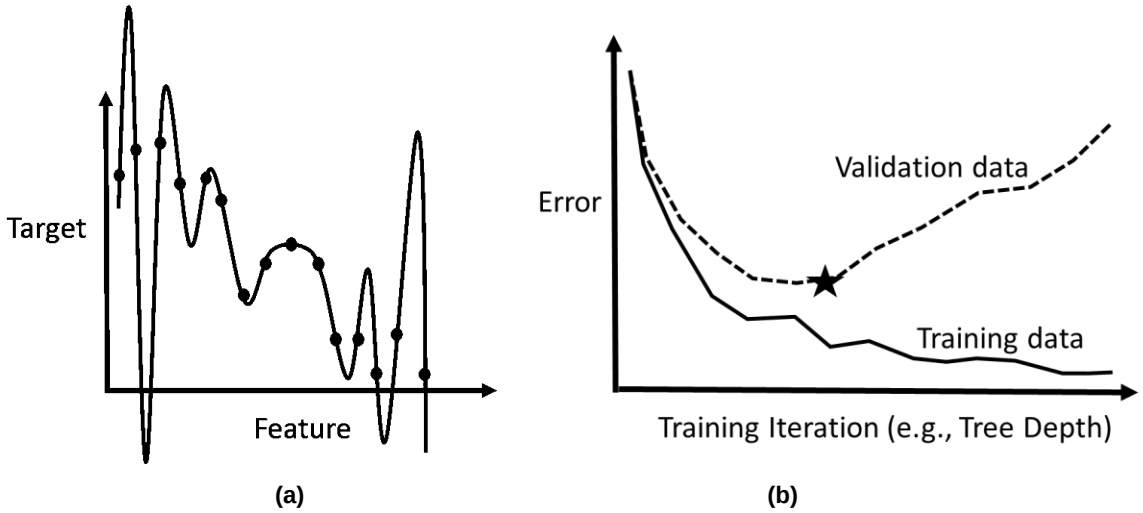
At some point, the model you ultimately generate is going to be used in some way to make predictions or other types of business decisions. A key consideration you must establish from the outset is whether the recommendations from the model will need to be justified by interpreting or explaining the logic behind how the model arrived at those conclusions; regulated environments might also have certain documentation requirements. Machine learning algorithms are usually formulated in a manner that emphasizes accuracy over interpretability. What makes these models accurate is literally what makes them difficult to understand: they are very complex. This is a fundamental tradeoff. If you know in advance that you will need to provide a business executive with some sort of reasoning behind a model's results other than "because that's what the model tells us," you might decide to limit your model candidates to those that can be more easily explained, such as regression and decision trees. It's much easier to describe the if-then hierarchical splitting logic of a decision tree than to describe the hyperplane that maximizes the margin between classes in the kernel-transformed feature space of a support vector machine. On the other hand, if you are simply looking for the most accurate generalizable model you can generate, then a more complex machine learning approach is appropriate. Either way, it's a question you should answer up front before you spend significant time and effort building models. Some guidance on answering this question is provided in the next section.

TRAINING

Now that you have ensured that you have sufficient and appropriate data, massaged the data into a form suitable for modeling, identified key features to include in your model, and established how the model is to be used, you are ready to make use of powerful machine learning algorithms to build predictive models or discover patterns in your data. This is really the phase where you should allow yourself more freedom to experiment with different approaches to identify the algorithms (and configuration of options for those algorithms) that produce the best model for your specific application. Still, as with proper data preparation, the process of training models cannot be entered into carelessly; an understanding of the main algorithm concepts and key concerns to heed will help ensure that you are using the appropriate algorithms and applying them judiciously.

OVERFITTING

A discussion of the primary issues and best practices for training models must start with the concept of overfitting. Recall that the goal is to build models that can be used to score future observations to enable you to make business decisions, such as to accept or deny credit application, flag fraudulent activity, identify tissue as cancerous or not, predict potential revenue, and so on. Machine learning algorithms are very effective at learning a mapping between the features and known target values in your existing data; if left unattended, they can often create a 100% accurate mapping, as shown in Figure 3a.



(a) **(b)**

Figure 3. (a) Example of Overfitting, (b) Training and Validation Error Compromise

Clearly, a model that is complex enough to perfectly fit the existing data will not generalize well when used to score new observations. It might provide accurate answers for some observations by chance, but in general it does not represent the trend of the data. This is referred to as overfitting. A decision tree is another prime example of an algorithm that can easily overfit the data. If the tree is allowed to continue to split the data all the way down to each observation being in its own leaf, it will be 100% accurate for every observation in the training data. But after a certain depth, the tree is not providing any information that can be applied in general.

Honest Model Assessment

Certainly you are striving to achieve low training error, but it is just as important (or more important) to achieve low generalization error. The training process needs to account for this compromise and make an honest assessment of the accuracy of the model. Machine learning typically involves training a succession of candidate configurations toward selecting a final model, and the error of each candidate must be assessed at every iteration of the training process. Assessing a candidate model on the data that are used to train the model would direct the algorithm to overfit to that training data.

Honest assessment, which is highly related to the bias-variance tradeoff, involves calculating error metrics from scoring the model on data that were not used in any way during the training process. It is very important to understand the distinctions between validation and testing, and to incorporate them as part of your model training, assessment, and selection process.

- **Validation** data are holdout data that are used to assess the model *during* training for the purpose of selecting variables and adjusting the model parameters or hyperparameters in order to generate a more accurate, generalizable model. Validation data sets are instrumental in evaluating the bias-variance tradeoff and preventing overfitting. As shown in Figure 3b, the error evaluated on the validation set will actually start to increase at some point, indicating that attempting to fit the training data any more accurately will diminish the generalization capability of the model. In lieu of a separate holdout set (which might not be feasible for smaller data sets), ***k*-fold cross validation** can be performed. In this technique, the training data set is split into *k* subsets and each subset is held out and used for validation on a model that is trained by using the other *k*–1 subsets; the error is taken to

be the average across all of the models. Whether through a validation set or through cross validation, ensure that the training process assesses the error on data that are not used to train the model in order to avoid overfitting to the training data.

- **Test** data are holdout data that are used *at the end* of model fitting to obtain a final, honest assessment of how well the trained model generalizes to new data. The reason for using test data (instead of validation data) for an unbiased assessment is that validation data play a role in the model training process and hence would yield a biased assessment similar to assessment on training data. For this reason, a test data set should be used only at the end of the analysis and should not play a role in the model training process.

JUDICIOUS ALGORITHM SELECTION AND TUNING

Ultimately, the success of your machine learning application comes down to the effectiveness of the actual model you build. The popular "no free lunch" theorem (Wolpert 1996) states that no one model works best for every problem—selecting the appropriate algorithm and configuring the hyperparameters to tune that algorithm for maximum effectiveness are critical steps in the process.

Algorithm Selection

Selecting the modeling algorithm for your machine learning application can sometimes be the most difficult part. The decision of which algorithm to use can be guided by answering a few key questions:

- What is the size and nature of your data?

  If you expect a fairly linear relationship between your features and your target, linear or logistic regression or a linear kernel support vector machine might be sufficient. Linear models are also a good choice for large data sets due to their training efficiency and due to the curse of dimensionality. As the number of features increase, the distance between points grows and observations are more likely to be linearly separable. To an extent, nonlinearity and interaction effects can be captured by adding higher-order polynomial and interaction terms in a regression model. However, as illustrated in Figure 4, more complex relationships can be modeled through the power of the more sophisticated machine learning algorithms such as decision trees, random forests, neural networks, and nonlinear kernel support vector machines. Of course, these more sophisticated algorithms can require more training time and might be unsuitable for very large data sets.

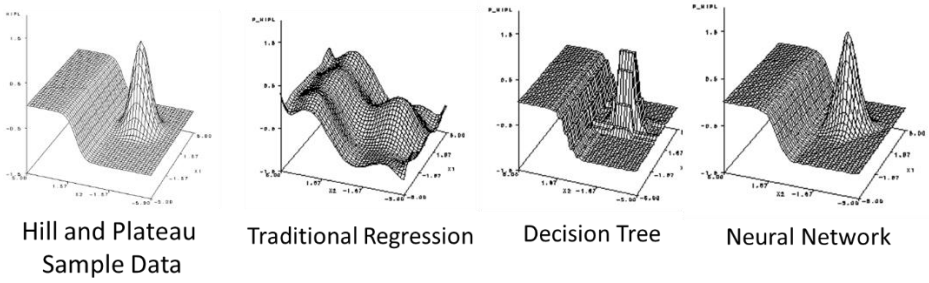


Figure 4. Various Models of a Complex Target Profile

- What are you trying to achieve with your model?

  Are you creating a model to classify observations, predict a value for an interval target, detect patterns or anomalies, or provide recommendations? Answering this question will direct you to a subset of machine learning algorithms that specialize in the particular type of problem.

- How accurate does your model need to be?

  Although you always want your model to be as accurate as possible when applied to new data, it is still always good to strive for simplicity. Simpler models train faster and are easier to understand, making it easier to explain how and why the results were achieved. Simpler models are also easier to deploy. Start with a regression model as a benchmark, and then train a more complex model such as a neural net, random forest, or gradient boosted model. If your regression model is much less accurate than the more complex model, you have probably missed some important predictor or interaction of predictors. An additional benefit of a simpler model is that it will be less prone to overfitting the training data.

- How much time do you have to train your model?

  This question goes hand-in-hand with the question of how accurate your model needs to be. If you need to train a model in a short amount of time, linear or logistic regression and decision trees are probably your best options. If training time is not an issue, take advantage of the powerful algorithms (neural networks, support vector machines, gradient boosting, and so on) that iteratively refine the model to better represent complex relationships between features and the target of interest.

- How interpretable or understandable does your model need to be?

  It is very important to establish the expectations of your model consumer in regards to how explainable your model must be. If an uninterpretable prediction is acceptable, you should use as sophisticated an algorithm as you can afford in terms of time and computational resources. Train a neural network, a support vector machine, or any flavor of ensemble model to achieve a highly accurate and generalizable model. If interpretability or explainable documentation is important, use decision trees or a regression technique, and consider using penalized regression techniques, generalized additive models, quantile regression, or model averaging to refine your model.

  If you need to ensure high accuracy but still need to explain the model results, a common approach is to train a complex model, use that model to generate predicted target values for all training observations, and then use these predicted values to train a decision tree. This decision tree is then essentially a surrogate model that acts as a proxy to the complex logic of the other algorithm. New observations are scored on the complex model for more accurate evaluation, but the surrogate model is used to explain the logic.

Table 1 in the Appendix provides a reference guide for the usage of the most common machine learning algorithms.

Even if you use these questions as guidance, selecting the single most effective algorithm to model your business problem can be very challenging. Experiment as much as you can afford to, and consider using an ensemble of multiple techniques, as described in the section “Ensemble Modeling” on page 16.

Regularization and Hyperparameter Tuning

The objective of a learning algorithm is to find the model parameters that minimize the loss function over the independent samples. For example, these parameters could be regression weights for a linear model, or they could be the adaptive weights on defining the connections within a neural network. As the complexity of your model increases, its predictive abilities often decrease after a certain point due to overfitting and multicollinearity issues. Hence, the resulting models often do not generalize well to new data, and they yield unstable parameter estimates.

Regularization methods help deal with overfitting models and multicollinearity problems by placing one or more penalties on the objective function that controls the size of the model weights. This penalty on the model weights decreases the variance of the model while increasing its bias. The total error of a model is the sum of its variance and bias. If the amount of penalty on the model weights is selected carefully, the

decrease in variance is less than the gain in bias, and hence the total error of the model decreases. This gives you a better model that has improved predictive abilities and more stable model parameters, and is known as the **bias-variance tradeoff**. Figure 5 illustrates the bias-variance tradeoff as it relates to the total error (Hastie, Tibshirani, and Friedman 2001).

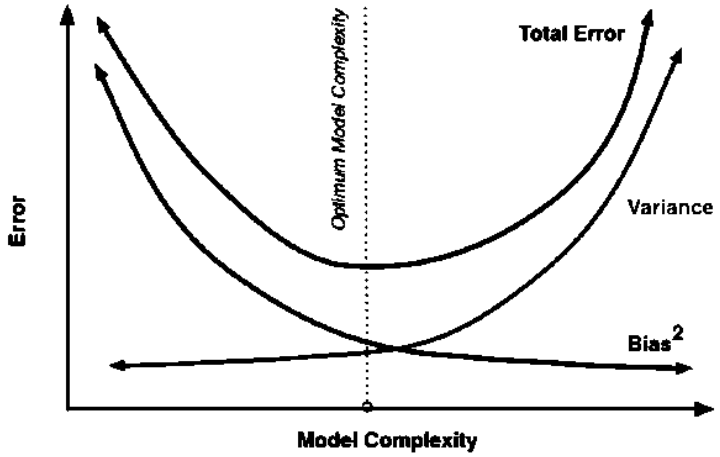


Figure 5: Bias-Variance Tradeoff

As a simple example, consider a linear regression model for which the penalty is placed on the squared error loss function as

$$arg\ min_{\beta}\{\|Y - X\beta\|^2 + \lambda p(\beta)$$

where $||Y - X\beta||^2$ is the squared error loss function, β represents the vector of regression weights, λ is a hyperparameter (tuning or regularization parameter), and $p(\beta)$ defines the form of the penalty. The form of the penalty is defined by L1 regularization (sum of the absolute value of the regression coefficients) for LASSO regression (Tibshirani 1996) and by L2 regularization (sum of the square of the regression coefficients) for ridge regression. Both L1 and L2 regularization shrink model weights toward 0. L1 regularization performs feature selection by setting some of the weights exactly to 0, whereas L2 regularization never sets them to 0. Therefore, L2 regularization is good at dealing with multicollinearity issues by shrinking the correlated regression weights toward each other. Another advantage of L2 regularization is that it can be used with any type of learning algorithm; L1 regularization is more difficult to implement with some learning algorithms, in particular those that need to calculate gradient information.

Regularization methods are very useful techniques for reducing model overfitting, but they require you to set the hyperparameters to certain values. Hyperparameters do not necessarily need to be regularization parameters. For example, the number of hidden units for a neural network or the depth and number of leaves in a decision tree are also hyperparameters.

Optimal hyperparameter settings are extremely data-dependent; therefore, it is difficult to offer a general rule about how to identify a subset of important hyperparameters for a learning algorithm or how to find optimal values of each hyperparameter that would work for all data sets. Controlling hyperparameters of a learning algorithm is very important because proper control can increase accuracy and prevent overfitting.

Hyperparameter tuning is an optimization task, and each proposed hyperparameter setting requires the model training process to derive a model for the data set and evaluate results on the holdout or cross validation data sets. After evaluating a number of hyperparameter settings, the hyperparameter tuner provides the setting that yields the best performing model. The last step is to train a new model on the entire data set (which includes both training and validation data) under the best hyperparameter setting.

Strategies for hyperparameter tuning include the following:

- **Grid search:** In grid search, the sets of possible hyperparameter values are formed by assembling every possible combination of hyperparameter values. If the number of tuning parameters is not very large (two or three parameters), the grid search approach is feasible as long as the learning algorithms are computationally efficient. Grid search is also easy to implement, and parallelization is straightforward. However, grid search suffers from the curse of dimensionality; that is, as the number of hyperparameters increases, the number of value combinations grows exponentially.
- **Random search:** Bergstra and Bengio (2012) showed that random search can often perform as well as grid search in a much more computationally efficient way. This might seem surprising, but a simple probability example makes it easy to understand. Suppose you choose only 60 random points over the entire hyperparameter space. Now imagine a 5% region around the point where the best set of hyperparameter values lies. Each random draw then has a 5% chance of lying in that region. Thus, the probability of missing this space for all 60 sample points would be $(1 – 0.05)^{60} = 0.046$, which is a very small probability (less than 5%). If you assume that the optimal region of the hyperparameters occupies at least 5% of the hyperparameter space, then a random search that uses only 60 sets of hyperparameter values will find that region with a success probability of at least 0.95 (1 – 0.046 = 0.954). The simplicity and yet surprisingly reasonable performance of random search makes it a very effective method of hyperparameter tuning. Moreover, parallelization is trivial for random search (as it is for grid search), and random search is computationally much more efficient.
- **Experimental design:** Another popular method of hyperparameter tuning is choosing the trials according to an experimental design instead of choosing them randomly. These methods are referred to as low-discrepancy point sets because they attempt to ensure that points are approximately equidistant from one another to fill the space efficiently. Some examples of these methods include the Sobol sequences (Antonov and Saleev 1979) and Latin hypercube sampling (McKay 1992).
- **Smart tuning methods:** Instead of searching through all possible candidate hyperparameter setting combinations or randomly choosing them, smart tuning methods use intelligent optimization routines (such as genetic algorithms), which start with a small set of points and use logic and information from those points to determine how to search through the space. The search is based on an objective of minimizing the model validation error, so each “evaluation” from the optimization algorithm’s perspective is a full cycle of model training and validation. These methods are designed to make intelligent use of fewer evaluations and thus save on the overall computation time. However, unlike a grid search or experimental design approach, tuning methods that are based on optimization routines are only partially parallelizable, depending on the optimization algorithm that is used. For example, a genetic algorithm can evaluate each candidate set of hyperparameter values in a population in parallel, but it must carry out crossover and mutation steps to determine the next population to be evaluated. Ultimately, a smart tuning method should find a more effective set of hyperparameter values than a random search or a discrete approach such as a grid search or experimental design finds. A benchmark study of different tuners is provided in Konen et al. (2011).

Regardless of the approach you take, understand that significant predictive power can be gained by intelligently tuning the hyperparameters for the selected algorithm. Don’t settle for the defaults.

Ensemble Modeling

Even with an understanding of some of the basic guidelines for selecting an algorithm and incorporating hyperparameter tuning, determining the single most effective machine learning algorithm (and its tuning parameters) to use for a particular problem domain and data set is daunting—even for experts. Ensemble modeling can take some of that weight off your shoulders and can give you peace of mind that the

predictions are the result of a collaborative effort, or consensus, among multiple models that are trained either from different algorithms that approach the problem from different perspectives, or from the same algorithm applied to different samples or using different tuning parameter settings, or both.

In general, ensemble modeling is all about using many models in collaboration to combine their strengths, compensate for their weaknesses, and make the resulting model generalize better for future data. If you rely on building a single model to represent the relationships and behavior in whatever your application is (even if you try to tweak and optimize that model based on the data you currently have), by focusing on one modeling method alone you're most likely sacrificing accuracy and robustness when making decisions and predictions on future data.

Many popular and effective algorithms have been formulated specifically around this concept of "consensus" prediction. These algorithms generally fall into one of the following categories:

- **Bagging:** Bagging (an abbreviation for bootstrap aggregating) is an approach in which multiple base learners (often decision trees) are trained on different sample sets of the data, which are randomly drawn with replacement, and their predictions are aggregated through a function such as majority voting or averaging. Bagging particularly focuses on combining the predictions of base learners to reduce variance and avoid overfitting; it is an efficient and ideal way to handle the bias-variance tradeoff. The **random forest** method developed by Breiman (2001) is a very effective and popular bagging algorithm that combines the predictions of multiple decision trees that are trained on different samples by using random subsets of the variables for splitting. Bagging algorithms are highly parallelizable.
- **Boosting:** In boosting (Shapire et al. 1998), a weight is applied to each observation in the training set and used as a probability distribution for sampling with replacement, a base learner (again often a decision tree) is trained, and misclassified points from the training set are provided higher weights for the next iteration of sampling and training. In this way, the algorithm is encouraged at each iteration to focus more on the points that it previously had difficulty classifying. A specific implementation of boosting called **gradient boosting** (Friedman 2001) adjusts the model on each iteration by using a random bootstrap sample to train a model to predict the residuals of previous models (which is theoretically equivalent to the stochastic gradient descent approach in optimization). Unlike bagging, boosting is a very sequential process.
- **Stacking:** Stacking involves training multiple models by using a diverse set of strong learners and then applying a higher-level "combiner" algorithm to generate a model that includes the predictions of the member models as inputs. The output of this combiner algorithm is the final prediction as a consensus among the member models. For example, a regression model (or other modeling algorithm of your choice) can be fit to weighted predictions from the base learners. The **super learner** (van der Laan, Polley, and Hubbard 2007) is a specific implementation of stacking that has proven to be quite effective.
- **Custom combinations:** In general, any number of diverse models can be trained on the data set and aggregated in some fashion to combine their strengths and compensate for their weaknesses. Each algorithm for training a model assumes some form of relationship between the inputs and the target. Combining predictions from multiple different algorithms might produce a relationship of a different form than any one of the algorithm assumes. If two models specify different relationships and fit the data well, it's likely that their average will generalize to new data much better. If not, then maybe an individual model is adequate. In practice, the best way to know is to combine some models and compare the results.

Note that if you do choose to employ an ensemble model in your application, the business decisions made using that model in the future will be difficult to explain because you cannot explicitly interpret the logic behind combining the results of multiple models. However, as previously suggested, you can alleviate this concern by using a more explainable surrogate model (such as a decision tree) that is built from the predictions of the ensemble or by using a small ensemble of interpretable models.

COMPUTATIONAL RESOURCE EXPLOITATION

Most machine learning tasks are computationally intensive. Computational resources should be used prudently to ensure efficient data preprocessing, model training, and scoring of new data. Data preparation and modeling techniques must often be refined repeatedly by the human operator to obtain the best results. Waiting for days while data is preprocessed or a model is trained not only is frustrating, but also can lead to inferior results. Most work in any field is conducted under time constraints; machine learning is no different. Preprocessing more data and training more models faster typically allows for more human iteration on the problem at hand and leads to better results.

Many researchers and practitioners still rely on older, but trusted and revered, single-threaded algorithm implementations. During prototyping, multithreading or distributing new algorithms can also be neglected. Interpreted languages are also heavily relied on in research and practice. Single-threaded algorithms and interpreted languages fail to sufficiently exploit computational resources. Researchers and practitioners should consider taking advantage of computational resources in the following ways:

- **Concurrent execution of single-threaded algorithms**: Single-threaded algorithms are necessary in certain cases. However, contemporary computers enable the execution of multiple threads, and the typical machine learning exercise requires the evaluation of multiple feature sets, tuning parameters, and optimization routines to find the best results. Instead of running these trials in serial, run them in parallel on different threads.

- **Hardware acceleration:** For I/O intensive tasks, such as using databases or SAS® software for data preparation, use solid state hard drives (SSDs). For computationally intensive, but parallelizable, tasks such as matrix algebra, use graphical processing units (GPUs). Numerous libraries are available that allow the transparent use of GPUs from high-level languages.

- **Low-level languages and libraries:** Low-level languages such as C and Fortran, though often seen as more difficult to use, are much faster than interpreted languages. Java can also offer significant performance increases over interpreted languages. When you are faced with designing a computationally intensive machine learning application that will be used repeatedly and over a longer period of time, carefully consider the tradeoff between perceived development difficulties with lower-level languages and the performance drawbacks of interpreted languages. An easier approach to increase performance might be to exploit the multiple packages and libraries that enable compiled lower-level binaries to be called from interpreted languages. A newer generation of tensor manipulation libraries even enables users of interpreted languages to transparently compile and then execute optimized low-level code on CPUs and GPUs.

- **Distributed computing**: Designing algorithms for distributed computing environments, where data and tasks are split across many connected computers, can greatly reduce execution times. However, not all distributed environments are well suited for machine learning and not all machine learning algorithms are well suited for distributed computing. Generally, shared-nothing environments can present insurmountable implementation problems for sophisticated machine learning tasks, and algorithms or implementations that require the movement of large amounts of data across the network of the distributed environment can easily negate any potential computational timing gains.

DEPLOYMENT

METHOD OF DEPLOYMENT

Given that machine learning models tend to be difficult to interpret, their primary use is to create predictions that create value (monetary or otherwise) for an organization or other entity. The actual mechanism by which machine learning models will create their predictions requires thought and attention.

For example, making predictions on an individual's laptop is a good idea only for a limited time in most cases. If a model is really useful, it needs to be used by an organization in an operational manner to make decisions quickly, if not automatically. Keep in mind that some level of data preparation has likely been applied to the data set in its original, raw form, and this must be accounted for when making predictions on new observations. Moving the logic that defines all the necessary data preparation and mathematical expressions of a sophisticated predictive model from a development environment such as a personal laptop into an operational database is one of the most difficult and tedious aspects of machine learning. Mature, successful organizations are masters of this process—called "model deployment," "deployment," or "model production."

To better understand model deployment, consider a credit card company. These companies often use logistic regression models to automatically authorize each transaction. This logistic regression model probably starts out as Python, R, or SAS code on an individual's laptop or workstation. However, because this model must be used millions of times a day in a massive number of simultaneous authorization decisions that are guaranteed to be made in milliseconds, the model simply cannot be run on an individual's laptop or workstation. Moreover, interpreted languages are probably too slow to guarantee millisecond response times. The model probably needs to be ported into a compiled language, such as C or Java. Model deployment is the process of moving the model from an individual's development environment to a large, powerful, and secure database or server where it can be used simultaneously by many mission-critical processes. Deployment as a web service that is programmatically accessible from custom applications and websites is another powerful and popular approach. Although deployment can require additional technical skills and knowledge beyond the analytics domain, many commercial machine learning software vendors provide this capability in a convenient, automated manner.

MONITORING AND UPDATING

Even after a model has been deployed, it must be monitored. Because models are often trained on static snapshots of data, their predictions typically become less accurate over time as the environment shifts away from the conditions that were captured in the training data. Consider a model for car insurance rates, which just 10 years ago did not account for behaviors such as texting while driving. Or consider a movie recommendation model that must adapt as viewers grow and mature through stages of life. After a certain period of time, the error rate on new data surpasses a predefined threshold, and models must be retrained or replaced. Champion-challenger testing is another common model deployment practice, in which a new, challenger model is compared against a currently deployed model at regular time intervals. When a challenger model outperforms a currently deployed model, the deployed model is replaced by the challenger, and the champion-challenger process is repeated. Yet another approach to refreshing a trained model is through online updates. Online updates continuously change the value of model parameters or rules based on the values of new, streaming data. It is prudent to assess the trustworthiness of real-time data streams before implementing an online modeling system.

CONCLUSION

The field of machine learning offers an array of powerful techniques for generating flexible and highly accurate models that empower businesses to effectively make use of vast amounts of data to make decisions. However, these techniques should not be considered to be push-button, black-box solutions that can be employed in an unattended manner. From data exploration and preparation, through model training, to ultimate deployment, these techniques must be used as part of a scientific approach to developing your overall machine learning application. The common issues and associated best practices described in this paper, although not necessarily comprehensive, offer some guidance on key points to consider in formulating a successful solution.

REFERENCES

Antonov, I. A., and Saleev, V. M. (1979). "An Economic Method of Computing LP_{τ}-Sequences." *USSR Computational Mathematics and Mathematical Physics* 19:252–256.

Bellman, R. E. (1957). *Dynamic Programming*. Princeton, NJ: Princeton University Press.

Bergstra, J., and Bengio, Y. (2012). "Random Search for Hyper-parameter Optimization." *Journal of Machine Learning Research* 13:281–305.

Breiman, L. (2001). "Random Forests." *Machine Learning* 45:5–32.

Cunningham, P. (2007). *Dimension Reduction*. Technical Report UCD-CSI-2007-7, University College Dublin.

Dasu, T., and Johnson, T. (2003). *Exploratory Data Mining and Data Cleaning*. Hoboken, NJ: John Wiley & Sons.

Donoho, D. (2015). "50 Years of Data Science." Available at http://courses.csail.mit.edu/18.337/2015/docs/50YearsDataScience.pdf.

Friedman, J. H. (2001). "Greedy Function Approximation: A Gradient Boosting Machine." *Annals of Statistics* 29:1189–1232.

Hastie, T., Tibshirani, R., and Friedman, J. (2001). *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. New York: Springer-Verlag.

He, H., and Garcia, E. A. (2009). "Learning from Imbalanced Data." *IEEE Transactions on Knowledge and Data Engineering* 21:1263–1284.

Huber, P. J. (1964). "Robust Estimation of a Location Parameter." *Annals of Mathematical Statistics* 35:73–101.

Konen, W., Koch, P., Flasch, O., Bartz-Beielstein, T., Friese, M., and Naujoks, B. (2011). "Tuned Data Mining: A Benchmark Study on Different Tuners." In *Proceedings of the Thirteenth Annual Conference on Genetic and Evolutionary Computation* (GECCO-2011). New York: SIGEVO/ACM.

Lee, D. D., and Seung, H. S. (2000). "Algorithms for Non-negative Matrix Factorization." In *Advances in Neural Information Processing Systems 13: Proceedings of the 2000 Conference*, edited by T. K. Leen, T. G. Dietterich, and V. Tresp, 556–562. Cambridge, MA: MIT Press.

Lohninger, H. (1999). *Teach/Me Data Analysis*. Berlin: Springer-Verlag.

Marcus, G. F. (1998). "Rethinking Eliminative Connectionism." *Cognitive Psychology* 37:243–282.

McKay, M. D. (1992). "Latin Hypercube Sampling as a Tool in Uncertainty Analysis of Computer Models." In *Proceedings of the Twenty-Fourth Conference on Winter Simulation* (WSC '92), edited by J. J. Swain, D. Goldsman, R. C. Crain, and J. R Wilson, 557–564. New York: ACM.

Mease, D., Wyner, A. J., and Buja, A. (2007). "Boosted Classification Trees and Class Probability/Quantile Estimation." *Journal of Machine Learning Research* 8:409–439.

Rubin, D. (1987). *Multiple Imputation for Nonresponse in Surveys*. New York: John Wiley & Sons. Classic edition, published in 2004.

Schapire, R. E., Freund, Y., Bartlett, P., and Lee, W. S. (1998). "Boosting the Margin: A New Explanation for the Effectiveness of Voting Methods." *Annals of Statistics* 26:1651–1686.

Schölkopf, B., Smola, A., and Müller, K.-R. (1998). "Nonlinear Component Analysis as a Kernel Eigenvalue Problem." *Neural Computation* 10:1299–1319.

Scott, A. J., and Wild, C. J. (1986). "Fitting Logistic Models under Case-Control or Choice Based Sampling." *Journal of the Royal Statistical Society B* 48:170–182.

Spruyt, V. (2014). "The Curse of Dimensionality in Classification." *Computer Vision for Dummies* (blog). http://www.visiondummy.com/2014/04/curse-dimensionality-affect-classification/.

Tibshirani, R. (1996). "Regression Shrinkage and Selection via the Lasso." *Journal of the Royal Statistical Society B* 58:267–288.

Van der Laan, M. J., Polley, E. C., and Hubbard, A. E. (2007). "Super Learner." *UC Berkeley Division of Biostatistics Working Paper Series*, No. 222. http://biostats.bepress.com/ucbbiostat/paper222.

Vapnik, V.M. (1996). *The Nature of Statistical Learning Theory*. New York: Springer-Verlag.

Wickham, H. (2014). "Tidy Data." *Journal of Statistical Software* 59:1–23.`

Wielenga, D. (2007). "Identifying and Overcoming Common Data Mining Mistakes." In *Proceedings of the SAS Global Forum 2007 Conference*. Cary, NC: SAS Institute Inc. http://www2.sas.com/proceedings/forum2007/073-2007.pdf.

Wolpert, D. H. (1996). "The Lack of A Priori Distinctions between Learning Algorithms." *Neural Computation* 8:1341–1390.

Zhang, J., and Mani, I. (2003). "kNN Approach to Unbalanced Data Distributions: A Case Study Involving Information Extraction." In *Proceedings of the Twentieth International Conference on Machine Learning* (ICML-2003), Workshop on Learning from Imbalanced Data Sets II. Palo Alto, CA: AAAI Press.

Zou, H., and Hastie, T. (2005). "Regularization and Variable Selection via the Elastic Net." *Journal of the Royal Statistical Society B* 67:301–320.

Zou, H., Hastie, T., and Tibshirani, R. (2006). "Sparse Principal Component Analysis." *Journal of Computational and Graphical Statistics* 15:265–286.

ACKNOWLEDGMENTS

The authors would like to thank Jorge Silva and Anne Baxter for their contributions to this paper.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the authors:

Brett Wujek
SAS Institute Inc.
brett.wujek@sas.com

Patrick Hall
SAS Institute Inc.
patrick.hall@sas.com

Funda Güneş
SAS Institute Inc.
funda.gunes@sas.com

MACHINE LEARNING QUICK REFERENCE: ALGORITHMS - 1

Algorithm Type	Common Usage	Suggested Usage	Suggested Scale	Interpretability	Common Concerns
Penalized Regression	• Supervised regression • Supervised classification	• Modeling linear or linearly separable phenomena • Manually specifying nonlinear and explicit interaction terms • Well suited for $N << p$	Small to large data sets	High	• Missing values • Outliers • Standardization • Parameter tuning
Naïve Bayes	Supervised classification	• Modeling linearly separable phenomena in large data sets • Well-suited for extremely large data sets where complex methods are intractable	Small to extremely large data sets	Moderate	• Strong linear independence assumption • Infrequent categorical levels
Decision Trees	• Supervised regression • Supervised classification	• Modeling nonlinear and nonlinearly separable phenomena in large, dirty data • Interactions considered automatically, but implicitly • Missing values and outliers in input variables handled automatically in many implementations • Decision tree ensembles (e.g., random forests and gradient boosting) can increase prediction accuracy and decrease overfitting, but also decrease scalability and interpretability	Medium to large data sets	Moderate	• Instability with small training data sets • Gradient boosting can be unstable with noise or outliers • Overfitting • Parameter tuning
***k*-Nearest Neighbors (kNN)**	• Supervised regression • Supervised classification	• Modeling nonlinearly separable phenomena • Can be used to match the accuracy of more sophisticated techniques, but with fewer tuning parameters	Small to medium data sets	Low	• Missing values • Overfitting • Outliers • Standardization • Curse of dimensionality
Support Vector Machines (SVM)	• Supervised regression • Supervised classification • Anomaly detection	• Modeling linear or linearly separable phenomena by using linear kernels • Modeling nonlinear or nonlinearly separable phenomena by using nonlinear kernels • Anomaly detection with one-class SVM (OSVM)	• Small to large data sets for linear kernels • Small to medium data sets for nonlinear kernels	Low	• Missing values • Overfitting • Outliers • Standardization • Parameter tuning • Accuracy versus deep neural networks depends on choice of nonlinear kernel; Gaussian and polynomial often less accurate
Artificial Neural Networks (ANN)	• Supervised regression • Supervised classification • Unsupervised clustering • Unsupervised feature extraction • Anomaly detection	• Modeling nonlinear and nonlinearly separable phenomena • Deep neural networks (e.g., deep learning) are well-suited for state-of-the-art pattern recognition in images, videos, and sound • All interactions considered in fully connected, multilayer topologies • Nonlinear feature extraction with autoencoder and restricted Boltzmann machine (RBM) networks • Anomaly detection with autoencoder networks • Clustering and visualization with self-organizing maps (SOMs)	• Usually small to medium data sets • Stochastic gradient descent (SGD) optimization drastically increases scalability	Low	• Missing values • Overfitting • Outliers • Standardization • Parameter tuning

MACHINE LEARNING QUICK REFERENCE: ALGORITHMS - 2

Algorithm Type	Common Usage	Suggested Usage	Suggested Scale	Interpretability	Common Concerns
Association Rules	• Supervised rule building • Unsupervised rule building	Building sets of complex rules by using the co-occurrence of items or events in transactional data sets	Medium to large transactional data sets	Moderate	• Instability with small training data • Overfitting • Parameter tuning
***k*-Means**	Unsupervised clustering	• Creating a known a priori number of spherical, disjoint, equally sized clusters • *k*-modes method can be used for categorical data • *k*-prototypes method can be used for mixed data	Small to large data sets	Moderate	• Missing values • Outliers • Standardization • Correct number of clusters is often unknown • Highly sensitive to initialization • Curse of dimensionality
Hierarchical Clustering	Unsupervised clustering	Creating a known a priori number of nonspherical, disjoint, or overlapping clusters of different sizes	Small data sets	Moderate	• Missing values • Standardization • Correct number of clusters is often unknown • Curse of dimensionality
Spectral Clustering	Unsupervised clustering	Creating a data-dependent number of arbitrarily shaped, disjoint, or overlapping clusters of different sizes	Small data sets	Moderate	• Missing values • Standardization • Parameter tuning • Curse of dimensionality
Principal Components Analysis (PCA)	Unsupervised feature extraction	• Extracting a data-dependent number of linear, orthogonal features, where ***N*** >> **p** • Extracted features can be rotated to increase interpretability, but orthogonality is usually lost • Singular value decomposition (SVD) is often used instead of PCA on wide or sparse data • Sparse PCA can be used to create more interpretable features, but orthogonality is lost • Kernel PCA can be used to extract nonlinear features	• Small to large data sets for traditional PCA and SVD • Small to medium data sets for sparse PCA and kernel PCA	Generally low, but higher for sparse PCA or rotated solutions	• Missing values • Outliers
Nonnegative Matrix Factorization (NMF)	Unsupervised feature extraction	Extracting a known a priori number of interpretable, linear, oblique, nonnegative features	Small to large data sets	High	• Missing values • Outliers • Standardization • Correct number of features is often unknown • Presence of negative values
Random Projections	Unsupervised feature extraction	Extracting a data-dependent number of linear, uninterpretable, randomly-oriented features of equal importance	Medium to extremely large data sets	Low	Missing values
Factorization Machines	• Supervised regression and classification • Unsupervised feature extraction	• Extracting a known a priori number of uninterpretable, oblique features from sparse or transactional data sets • Can automatically account for variable interactions • Creating models from a large number of sparse features; can outperform SVM for sparse data	Medium to extremely large sparse or transactional data sets	Moderate	• Missing values • Outliers • Standardization • Correct number of features is often unknown • Less well suited for dense data

Best Practices for Applying Deep Learning to Novel Applications

Leslie N. Smith
Navy Center for Applied Research in Artificial Intelligence
U.S. Naval Research Laboratory, Code 5514
Washington, DC 20375
leslie.smith@nrl.navy.mil

ABSTRACT

This report is targeted to groups who are subject matter experts in their application but deep learning novices. It contains practical advice for those interested in testing the use of deep neural networks on applications that are novel for deep learning. We suggest making your project more manageable by dividing it into phases. For each phase this report contains numerous recommendations and insights to assist novice practitioners.

Introduction

Although my focus is on deep learning (DL) research, I am finding that more and more frequently I am being asked to help groups without much DL experience who want to try deep learning on their novel (for DL) application. The motivation for this NRL report derives from noticing that much of my advice and guidance is similar for all such groups. Hence, this report discusses the aspects of applying DL that are more universally relevant.

While there are several useful sources of advice on best practices for machine learning [1-5], there are differences relevant to DL that this report addresses. Still, I recommend the reader read and become familiar with these references as they contain numerous gems. In addition, there are many sources on best practices on the topic of software engineering and agile methodologies that I assume the reader is already familiar with (e.g., [6, 7]). The closest reference to the material in this report can be found in Chapter 11 of "Deep Learning" [8] on "Practical Methodology" but here I discuss a number of factors and insights not covered in this textbook.

You can see below that a deep learning application project is divided into phases. However, in practice you are likely to find it helpful to return to an earlier phase. For example, while finding an analogy in phase 3, you might discover new metrics that you hadn't considered in phase 1. All of these best practices implicitly include iteratively returning to a phase and continuous improvement as the project proceeds.

Phase 1: Getting prepared

In this report I assume you are (or have access to) a subject matter expert for your application. You should be familiar with the literature and research for solving the associated problem and know the state-of-the-art solutions and performance levels. I recommend you consider here at the beginning if a deep learning solution is a worthwhile effort. You must consider the performance level of the state-of-the-art and if it is high, whether it is worthwhile to put in the efforts outlined in this report for an incremental improvement. Don't jump into deep learning only because it seems like the latest and

greatest methodology. You should also consider if you have the computer resources since each job to train a deep network will likely take days or weeks. I have made ample use of DoD's HPC systems in my own research. In addition, you should consider if machine learning is appropriate at all – remember training a deep network requires lots of labeled data, as described in phase 2.

The first step is quantitatively defining what success looks like. What will you see if this is successful, whether it is done by human or machine? This helps define your evaluation metrics. Which metrics are important? Which are less important? You need to specify all quantitative values that play a role in the success of this project and determine how to weigh each of them. You also need to define objectives for your metrics; is your goal surpass human level performance? Your objectives will strongly influence the course of the project. Knowing quantitatively what human performance is on this task should guide your objectives; how does the state-of-the-art compare to human performance? Also, knowing how a human solves this task will provide valuable information on how the machine might solve it.

Some of these metrics can also lead to the design of the loss function, which is instrumental in guiding the training of the networks. Don't feel obligated to only use softmax/cross entropy/log loss just because that is the most common loss function, although you should probably start with it. Your evaluation metrics are by definition the quantities that are important for your application. Be willing to test these metrics as weighted components of the loss function to guide the training (see phase 6).

Although you are likely considering deep learning because of its power, consider how to **make the network's "job" as easy as possible**. This is anti-intuitive because it is the power of deep networks that likely motivates you to try it out. However, the easier the job that the networks must perform, the easier it will be to train and the better the performance. Are you (or the state-of-the-art) currently using heuristics/physics that can be utilized here? Can the data be preprocessed? While the network can learn complex relationships, remember: "the easier the network's job, the better it will perform". So it is worthwhile to spend time considering what you can leverage from previous work and what the network needs to do for you. Let's say you want to improve on a complex process where the physics is highly approximated (i.e., a "spherical cow" situation); you have a choice to input the data into a deep network that will (hopefully) output the desired result or you can train the network to find the correction in the approximate result. The latter method will almost certainly outperform the former. On the other hand, do not rely on manual effort to define potential heuristics – the scarcest resource is human time so let the network learn its representations rather than require any fixed, manual preprocessing.

In addition, you might want to write down any assumptions or expectations you have regarding this state-of-the-art process as it will clarify them for yourself.

Phase 2: Preparing your data

Deep learning requires a great deal of training data. You are probably wondering "how much training data do I need?" The number of parameters in the network is correlated with the amount of training data. The number of training samples will limit your architectural choices in phase 6. The more training data, the larger and more accurate the network can be. So the amount of training data depends on the objectives you defined in phase 1.

In addition to training data, you will need a smaller amount of labeled validation or test data. This test data should be similar to the training data but not the same. The network is not trained on the test data but it is used to test the generalization ability of the network.

If the amount of training data is very limited, consider transfer learning [9] and domain adaptation [10, 11]. If this is appropriate, download datasets that are closest to your data to use for pre-training. In addition, consider creating synthetic data. Synthetic data has the advantages that you can create plenty of samples and make it diverse.

The project objectives also guides the choosing of the training data samples. Be certain that the training data is directly relevant to the task and that it is diverse enough that it **covers the problem space**. Study the statistics of each class. For example, are the classes balanced? An example of a balanced class is cats versus dogs while an unbalanced class with be cats versus all other mammals (if your problem is inherently unbalanced, talk with a deep learning expert).

What preprocessing is possible? Can you zero mean and normalize the data? This makes the network's job easier as it removes the job of learning the mean. Normalization also makes the network's job easier by creating greater similarity between training samples.

As discussed above, investigate if there are ways to lower the dimensionality of the data using a priori knowledge or known heuristics. You don't need to spend time to manually determine heuristics because the goal is to save human time and you can let the network learn its own representations. Just know that the more irrelevant data the network has to sift through, the more training data is needed and the more time it will take to train the network. So leverage what you can from prior art.

Phase 3: Find an analogy between your application and the closest deep learning applications

Experts know not to start from scratch for every project. This is what makes them experts. They reuse solutions that have worked in the past and they search the deep learning literature for solutions from other researchers. Even if no one has ever done what you are trying to do, you still need to leverage whatever you can from the experts.

Deep learning has been applied to a variety of applications. In order to create your baseline model – your starting point – you need to find the applications that are in some ways similar to your application. You should search the DL literature and consider the "problems" various applications are solving to compare with the "problem" you need to solve in your application. Find similarities and analogies between these problems. Also, note of the differences between your new application and the deep learning application because these differences might require changing the architecture in phase 6.

When you find the closest application, look for code to download. Many researchers make their code available when they publish in a wide spread effort to release reproducible research. Your first aim is to replicate the results in the paper of the closest application. Later you should modify various aspects to see the effects on the results in a "getting to know it" stage. If you are lucky, there will be several codes available and you should replicate the results for all of them. This comparison will provide you with enough information so you can create a baseline in phase 4.

There are a few "classic" applications of deep learning and well known solutions. These include image classification/object recognition (convolutional networks), processing sequential data (RNN/LSTM/GRU)

such as language processing, and complex decision making (deep reinforcement learning). There are also a number of other applications that are common, such as image segmentation and super-resolution (fully convolutional networks) and similarity matching (Siamese networks). Appendix A lists a number of recent deep learning applications, the architecture used, and links to the papers that describe this application. This can give you some ideas but should not be your source for finding deep learning applications. Instead you should carefully search https://scholar.google.com and https://arxiv.org for the deep learning applications.

Phase 4: Create a simple baseline model

Always start simple, small, and easy. Use a smaller architecture than you anticipate you might need. Start with a common objective function. Use common settings for the hyper-parameters. Use only part of the training data. This is a good place to adopt some of the practices of agile software methodologies, such as simple design, unit testing, and short releases. Only get the basic functionality now and improve on it during phase 6. That is, plan on small steps, continuous updates, and to iterate.

Choose only one of the common frameworks, such as Caffe, TensorFlow, or MXnet. Plan to only use one framework and one computer language to minimize errors from unnecessary complexity. The framework and language choice will likely be driven by the replication effort you performed in phase 3.

If the network will be part of a larger framework, here is a good place to check that the framework APIs are working properly.

Phase 5: Create visualization and debugging tools

Understanding what is happening in your model will affect the success of your project. Carpenters have an expression “measure twice, cut once”. You should think “code once, measure twice”. In addition to evaluating the output, you should visualize your architecture and measure internal entities to understand why you are getting the results you are obtaining. Without diagnostics, you will be shooting in the dark to fix problems or improve performance.

You should have a general understanding of problems related to high bias (converging on the wrong result) versus high variance (not converging well) because there are different solutions for each type of problem; for example, you might fix high bias problems with a larger network but you would handle high variance problems by increasing the size of your training dataset.

Set up visualizations so you can monitor as much as possible while the architecture evolves. When possible, set up unit tests for all of your code modifications. You should compare training error to test error and both to human level performance. You might find your network will behave strangely and you need ways to determine what is going on and why. Start debugging the worst problems first. Find out if the problems are with the training data, aspects of the architecture, or the loss function.

Keep in mind that error analysis tries to explain the difference between current performance and perfect performance. Ablative analysis tries to explain the difference between some baseline performance and current performance. One or the other or both can be useful.

One motivation for using TensorFlow as your framework is that it has a visualization system called TensorBoard that is part of the framework. One can output the necessary files from TensorFlow and TensorBoard can be used to visualize your architecture, monitor the weights and feature maps, and

explore the embedded space the network creates. Hence, the debugging and visualization tools are available in the framework. With other frameworks, you need to find these tools (they are often available online) or create your own.

Phase 6: Fine tune your model

This phase will likely take the most time. You should experiment extensively. And not just with factors you believe will improve the result but try changing every factor just to learn what happens when it changes. Change the architecture design, depth, width, pathways, weight initialization, loss function, etc. Change each hyper-parameter to learn what the effect of increasing or decreasing it is. I recommend using the learning rate range test [12] to learn about the behavior of your network over a large range of learning rates. A similar program can be made to study the effect of other hyper-parameters.

Try various regularization methods, such as data augmentation, dropout, and weight decay. Generalization is one of the key advantages of deep networks so be certain to test regularization methods in order to maximize this ability to generalize to unseen cases.

You should experiment with the loss function. You used a simple loss function in the baseline but you also created several evaluation metrics that you care about and define success. **The only difference between the evaluation metrics and the loss function is that the metrics apply to the test data and the loss function is applied to the training data in order to train the network.** Can a more complicated loss function produce a more successful result? You can add weighted components to the loss function to reflect the importance of each metric to the results. Just be very careful to not complicate the loss function with unimportant criterion because it is the heart of your model.

Earlier you found analogies between your application and existing deep learning applications and chose the closest to be your baseline. Now compare to the second closest application. Or the third. What happens if you follow another analogy and use that architecture? Can you imagine a combination of the two to test?

In the beginning, you should have successes from some low hanging fruit. As you go on it will become more difficult to improve the performance. The objectives you defined in phase 1 should guide how far you want to pursue the performance improvements. Or you might want to now revise the objectives your defined earlier.

Phase 7: End-to-end training, ensembles and other complexities

If you have the time and budget, you can investigate more complex methods and there are worlds of complexities that are possible. There exists a huge amount of deep learning literature and more papers are appearing daily. Most of these papers declare new state-of-the-art results with one twist or another and some might provide you a performance boost. This section alone could fill a long report because there are so many architectures and other options to consider but if you are at this stage, consider talking with someone with a great deal of deep learning expertise because advice at this stage is likely to be unique to your application. .

However, there are two common methods you might consider; end-to-end training and ensembles.

As a general rule, end-to-end training of a connected system will outperform a system with multiple parts because a combined system with end-to-end training allows each of the parts to adapt to the task. Hence, it is useful to consider combining parts, if it is relevant for your application.

Ensembles of diverse learners (i.e., bagging, boosting, stacking) can also improve the performance over a single model. However, this will require you to train and maintain all the members of the ensemble. If your performance objectives warrant this, it is worthwhile to test an ensemble approach.

Summary

This report lays out many factors for you to consider when experimenting with deep learning on an application where it hasn't been previously used. Not every item here will be relevant but I hope that it covers most of the factors you should consider during a project. I wish you much luck and success in your efforts.

References:

1. Martin Zinkevich, “Rules of Machine Learning: Best Practices for ML Engineering”, http://martin.zinkevich.org/rules_of_ml/rules_of_ml.pdf
2. Brett Wujek, Patrick Hall, and Funda Güneș, “Best Practices for Machine Learning Applications”, https://support.sas.com/resources/papers/proceedings16/SAS2360-2016.pdf
3. Jason Brownlee, “How to Use a Machine Learning Checklist to Get Accurate Predictions, Reliably”, http://machinelearningmastery.com/machine-learning-checklist/
4. Domingos, Pedro. "A few useful things to know about machine learning." *Communications of the ACM* 55.10 (2012): 78-87.
5. Grégoire Montavon, Geneviève Orr, Klaus-Robert Müller, “Neural Networks: Tricks of the Trade”, Springer, 2012
6. Fergus Henderson, “Software engineering at Google”, CoRR, arXiv:1702.01715, 2017.
7. Gamma, Erich. *Design patterns: elements of reusable object-oriented software*. Pearson Education India, 1995.
8. Goodfellow, I., Bengio, Y., Courville, A., “Deep Learning”, MIT Press, 2016
9. Weiss, Karl, Taghi M. Khoshgoftaar, and DingDing Wang. "A survey of transfer learning." *Journal of Big Data* 3.1 (2016): 1-40.
10. Patel, Vishal M., Raghuraman Gopalan, Ruonan Li, and Rama Chellappa. "Visual domain adaptation: A survey of recent advances." *IEEE signal processing magazine* 32, no. 3 (2015): 53-69.
11. Gabriela Csurka “Domain Adaptation for Visual Applications: A Comprehensive Survey”, CoRR, arXiv:1702.05374, 2017.
12. Leslie N. Smith. Cyclical learning rates for training neural networks. In Proceedings of the IEEE Winter Conference on Applied Computer Vision, 2017.

Appendix A: Table of various deep learning applications

The following table lists some recent applications of deep learning, the architecture used for this application and a few references to papers in the literature that describe the application in much more detail.

Application	**Architecture**	**Comments**
Colorization of Black and White Images.	Large, fully convolutional	http://www.cs.cityu.edu.hk/~qiyang/publications/iccv-15.pdf http://arxiv.org/pdf/1603.08511.pdf

Adding Sounds To Silent Movies.	CNN + LSTM	http://arxiv.org/pdf/1512.08512.pdf
Automatic Machine Translation.	Stacked networks of large LSTM recurrent neural networks	http://www.nlpr.ia.ac.cn/cip/ZongPublications/2015/IEEE-Zhang-8-5.pdf https://arxiv.org/abs/1612.06897 https://arxiv.org/abs/1611.04558
Object Classification in Photographs.	Residual CNNs, ResNeXt, Densenets	http://papers.nips.cc/paper/5207-deep-neural-networks-for-object-detection.pdf
Automatic Handwriting Generation.	RNNs	http://arxiv.org/pdf/1308.0850v5.pdf
Character Text Generation.	RNNs	http://arxiv.org/pdf/1308.0850v5.pdf
Image Caption Generation.	CNN + LSTM	http://arxiv.org/pdf/1505.00487v3.pdf
Automatic Game Playing.	Reinforcement learning + CNNs	http://www.nature.com/nature/journal/v529/n7587/full/nature16961.html https://arxiv.org/abs/1612.00380
Generating audio	WaveNet = Dilated PixelCNN	https://arxiv.org/abs/1609.03499 https://arxiv.org/abs/1612.07837 https://arxiv.org/abs/1610.09001
Object tracking	CNN + hierarchical LSTMs	https://arxiv.org/abs/1701.01909 https://arxiv.org/abs/1611.06878 https://arxiv.org/abs/1611.05666
Lip reading	CNN + LSTMs	https://arxiv.org/abs/1701.05847 https://arxiv.org/abs/1611.05358
Modifying synthetic data into labeled training data	GAN	https://arxiv.org/abs/1701.05524
Single image super-resolution	Deep, fully convolutional networks	https://arxiv.org/abs/1612.07919

		https://arxiv.org/abs/1611.03679 https://arxiv.org/abs/1511.04587 https://arxiv.org/abs/1611.00591
Speech recognition	LSTMs	https://arxiv.org/abs/1701.03360 https://arxiv.org/abs/1701.02720
Generate molecular structures	RNNs	https://arxiv.org/abs/1701.01329
Time series analysis	Resnet + RNNs	https://arxiv.org/abs/1701.01887 https://arxiv.org/abs/1611.06455
Intrusion detection	RNNs or CNNs	https://arxiv.org/abs/1701.02145
Autonomous Planning	Predictron architecture	https://arxiv.org/abs/1612.08810
Object detection		https://arxiv.org/abs/1612.08242
Multi-modality classification	CNN + GAN	https://arxiv.org/abs/1612.07976 https://arxiv.org/abs/1612.00377 https://arxiv.org/abs/1611.06306
Health monitoring	All are used	https://arxiv.org/abs/1612.07640
Robotics	CNN (perception) RL (control)	https://arxiv.org/abs/1612.07139 https://arxiv.org/abs/1611.00201
Domain adaptation		https://arxiv.org/abs/1612.06897
Self-driving	CNNs	https://arxiv.org/abs/1612.06573 https://arxiv.org/abs/1611.08788 https://arxiv.org/abs/1611.05418
Visual question answering	Resnet	https://arxiv.org/abs/1612.05386

		https://arxiv.org/abs/1611.01604 https://arxiv.org/abs/1611.05896 https://arxiv.org/abs/1611.05546
Weather prediction	Graphical RNN	https://arxiv.org/abs/1612.05054
Detecting cancer	RBM	https://arxiv.org/abs/1612.03211
Genomics	Multiple NNs	https://arxiv.org/abs/1611.09340
Semantic segmentation	Fully conv Densenet	https://arxiv.org/abs/1611.09326 https://arxiv.org/abs/1611.06612
Hyperspectral classification	CNN	https://arxiv.org/abs/1611.09007
Natural Language Processing (NLP)	LSTM, GRU	https://arxiv.org/abs/1606.0673
Face detection	CNN	https://arxiv.org/abs/1611.00851