O'REILLY®

Compliments of MEMSQL

The Path to Predictive Analytics and Machine Learning

Conor Doherty, Steven Camiña,
Kevin White & Gary Orenstein

MEMSQL

ADAPT AND LEARN IN REAL TIME

The Database Platform for Real-Time Analytics

The Path to Predictive Analytics and Machine Learning

Conor Doherty, Steven Camiña, Kevin White, and Gary Orenstein

Beijing • Boston • Farnham • Sebastopol • Tokyo O'REILLY®

The Path to Predictive Analytics and Machine Learning
by Conor Doherty, Steven Camiña, Kevin White, and Gary Orenstein

Printed in the United States of America.

Published by O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472.

O'Reilly books may be purchased for educational, business, or sales promotional use. Online editions are also available for most titles (*http://safaribooksonline.com*). For more information, contact our corporate/institutional sales department: 800-998-9938 or *corporate@oreilly.com*.

Editors: Tim McGovern and Debbie Hardin
Production Editor: Colleen Lobner
Copyeditor: Octal Publishing, Inc.

Interior Designer: David Futato
Cover Designer: Randy Comer
Illustrator: Rebecca Demarest

September 2016: First Edition

Revision History for the First Edition
2016-08-25: First Release

978-1-491-96966-3

[LSI]

Table of Contents

Introduction

An Anthropological Perspective

If you believe that as a species, communication advanced our evolution and position, let us take a quick look from cave paintings, to scrolls, to the printing press, to the modern day data storage industry.

Marked by the invention of disk drives in the 1950s, data storage advanced information sharing broadly. We could now record, copy, and share bits of information digitally. From there emerged superior CPUs, more powerful networks, the Internet, and a dizzying array of connected devices.

Today, every piece of digital technology is constantly sharing, processing, analyzing, discovering, and propagating an endless stream of zeros and ones. This web of devices tells us more about ourselves and each other than ever before.

Of course, to meet these information sharing developments, we need tools across the board to help. Faster devices, faster networks, faster central processing, and software to help us discover and harness new opportunities.

Often, it will be fine to wait an hour, a day, even sometimes a week, for the information that enriches our digital lives. But more frequently, it's becoming imperative to operate *in the now*.

In late 2014, we saw emerging interest and adoption of multiple in-memory, distributed architectures to build real-time data pipelines. In particular, the adoption of a message queue like Kafka, transformation engines like Spark, and persistent databases like MemSQL

opened up a new world of capabilities for fast business to understand real-time data and adapt instantly.

This pattern led us to document the trend of real-time analytics in our first book, *Building Real-Time Data Pipelines: Unifying Applications and Analytics with In-Memory Architectures* (O'Reilly, 2015). There, we covered the emergence of in-memory architectures, the playbook for building real-time pipelines, and best practices for deployment.

Since then, the world's fastest companies have pushed these architectures even further with machine learning and predictive analytics. In this book, we aim to share this next step of the real-time analytics journey.

— Conor Doherty, Steven Camiña, Kevin White, and Gary Orenstein

CHAPTER 1
Building Real-Time Data Pipelines

Discussions of predictive analytics and machine learning often gloss over the details of a difficult but crucial component of success in business: implementation. The ability to use machine learning models in production is what separates revenue generation and cost savings from mere intellectual novelty. In addition to providing an overview of the theoretical foundations of machine learning, this book discusses pragmatic concerns related to building and deploying scalable, production-ready machine learning applications. There is a heavy focus on real-time uses cases including both *operational* applications, for which a machine learning model is used to automate a decision-making process, and *interactive* applications, for which machine learning informs a decision made by a human.

Given the focus of this book on implementing and deploying predictive analytics applications, it is important to establish context around the technologies and architectures that will be used in production. In addition to the theoretical advantages and limitations of particular techniques, business decision makers need an understanding of the systems in which machine learning applications will be deployed. The interactive tools used by data scientists to develop models, including domain-specific languages like R, in general do not suit low-latency production environments. Deploying models in production forces businesses to consider factors like model training latency, prediction (or "scoring") latency, and whether particular algorithms can be made to run in distributed data processing environments.

Before discussing particular machine learning techniques, the first few chapters of this book will examine modern data processing architectures and the leading technologies available for data processing, analysis, and visualization. These topics are discussed in greater depth in a prior book (*Building Real-Time Data Pipelines: Unifying Applications and Analytics with In-Memory Architectures* [O'Reilly, 2015]); however, the overview provided in the following chapters offers sufficient background to understand the rest of the book.

Modern Technologies for Going Real-Time

To build real-time data pipelines, we need infrastructure and technologies that accommodate ultrafast data capture and processing. Real-time technologies share the following characteristics: 1) in-memory data storage for high-speed ingest, 2) distributed architecture for horizontal scalability, and 3) they are queryable for real-time, interactive data exploration. These characteristics are illustrated in Figure 1-1.

Figure 1-1. Characteristics of real-time technologies

High-Throughput Messaging Systems

Many real-time data pipelines begin with capturing data at its source and using a high-throughput messaging system to ensure that every data point is recorded in its right place. Data can come from a wide range of sources, including logging information, web events, sensor data, financial market streams, and mobile applications. From there it is written to file systems, object stores, and databases.

Apache Kafka is an example of a high-throughput, distributed messaging system and is widely used across many industries. According to the Apache Kafka website, "Kafka is a distributed, partitioned, replicated commit log service." Kafka acts as a broker between producers (processes that publish their records to a topic) and consumers (processes that subscribe to one or more topics). Kafka can handle terabytes of messages without performance impact. This process is outlined in Figure 1-2.

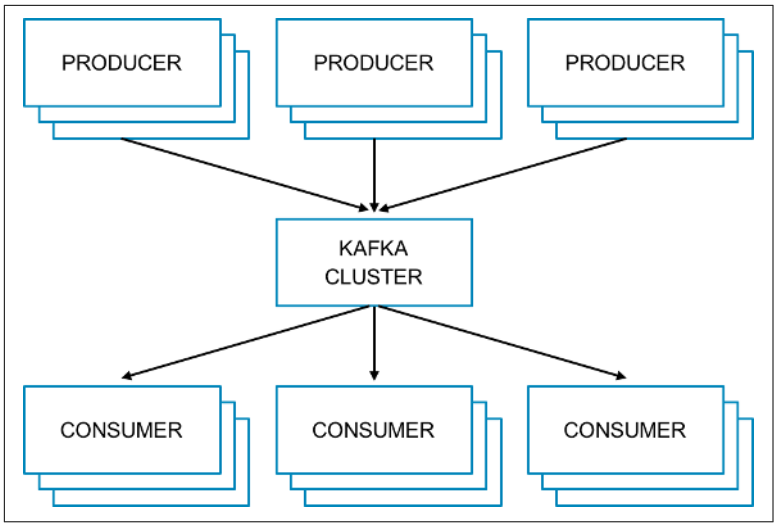

Figure 1-2. Kafka producers and consumers

Because of its distributed characteristics, Kafka is built to scale producers and consumers with ease by simply adding servers to the cluster. Kafka's effective use of memory, combined with a commit log on disk, provides ideal performance for real-time pipelines and durability in the event of server failure.

With our message queue in place, we can move to the next piece of data pipelines: the transformation tier.

Data Transformation

The data transformation tier takes raw data, processes it, and outputs the data in a format more conducive to analysis. Transformers serve a number of purposes including data enrichment, filtering, and aggregation.

Apache Spark is often used for data transformation (see Figure 1-3). Like Kafka, Spark is a distributed, memory-optimized system that is ideal for real-time use cases. Spark also includes a streaming library and a set of programming interfaces to make data processing and transformation easier.

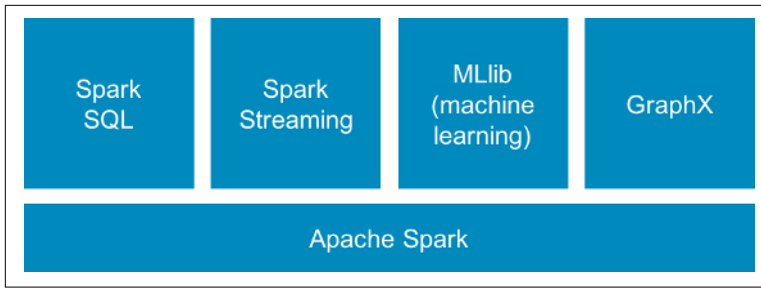

Figure 1-3. Spark data processing framework

When building real-time data pipelines, Spark can be used to extract data from Kafka, filter down to a smaller dataset, run enrichment operations, augment data, and then push that refined dataset to a persistent datastore. Spark does not include a storage engine, which is where an operational database comes into play, and is our next step (see Figure 1-4).

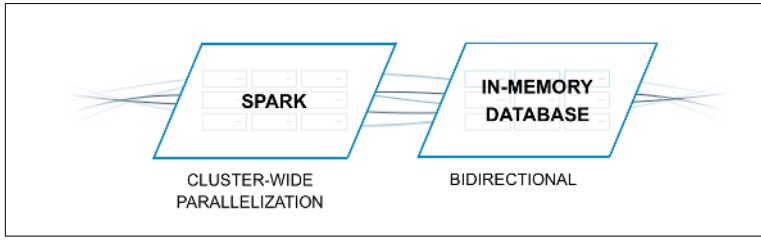

Figure 1-4. High-throughput connectivity between an in-memory database and Spark

Persistent Datastore

To analyze both real-time and historical data, it must be maintained beyond the streaming and transformations layers of our pipeline, and into a permanent datastore. Although unstructured systems like Hadoop Distributed File System (HDFS) or Amazon S3 can be used for historical data persistence, neither offer the performance required for real-time analytics.

On the other hand, a memory-optimized database can provide persistence for real-time and historical data as well as the ability to query both in a single system. By combining transactions and analytics in a memory-optimized system, data can be rapidly ingested from our transformation tier and held in a datastore. This allows

applications to be built on top of an operational database that supplies the application with the most recent data available.

Moving from Data Silos to Real-Time Data Pipelines

In a world in which users expect tailored content, short load times, and up-to-date information, building real-time applications at scale on legacy data processing systems is not possible. This is because traditional data architectures are siloed, using an Online Transaction Processing (OLTP)-optimized database for operational data processing and a separate Online Analytical Processing (OLAP)-optimized data warehouse for analytics.

The Enterprise Architecture Gap

In practice, OLTP and OLAP systems ingest data differently, and transferring data from one to the other requires Extract, Transform, and Load (ETL) functionality, as Figure 1-5 demonstrates.

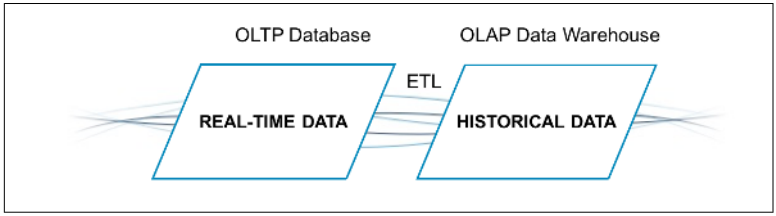

Figure 1-5. Legacy data processing model

OLAP silo

OLAP-optimized data warehouses cannot handle one-off inserts and updates. Instead, data must be organized and loaded all at once—as a large batch—which results in an offline operation that runs overnight or during off-hours. The tradeoff with this approach is that streaming data cannot be queried by the analytical database until a batch load runs. With such an architecture, standing up a real-time application or enabling analyst to query your freshest dataset cannot be achieved.

OLTP silo

On the other hand, an OLTP database typically can handle high-throughput transactions, but is not able to simultaneously run analytical queries. This is especially true for OLTP databases that use

disk as a primary storage medium, because they cannot handle mixed OLTP/OLAP workloads at scale.

The fundamental flaw in a batch processing system can be illustrated through an example of any real-time application. For instance, if we take a digital advertising application that combines user attributes and click history to serve optimized display ads before a web page loads, it's easy to spot where the siloed model breaks. As long as data remains siloed in two systems, it will not be able to meet Service-Level Agreements (SLAs) required for any real-time application.

Real-Time Pipelines and Converged Processing

Businesses implement real-time data pipelines in many ways, and each pipeline can look different depending on the type of data, workload, and processing architecture. However, all real-time pipelines follow these fundamental principles:

- Data must be processed and transformed on-the-fly so that it is immediately available for querying when it reaches a persistent datastore
- An operational datastore must be able to run analytics with low latency
- The system of record must be converged with the system of insight

One common example of a real-time pipeline configuration can be found using the technologies mentioned in the previous section—Kafka to Spark to a memory-optimized database. In this pipeline, Kafka is our message broker, and functions as a central location for Spark to read data streams. Spark acts as a transformation layer to process and enrich data into microbatches. Our memory-optimized database serves as a persistent datastore that ingests enriched data streams from Spark. Because data flows from one end of this pipeline to the other in under a second, an application or an analyst can query data upon its arrival.

CHAPTER 2

Processing Transactions and Analytics in a Single Database

Historically, businesses have separated operations from analytics both conceptually and practically. Although every large company likely employs one or more "operations analysts," generally these individuals produce reports and recommendations to be implemented by others, in future weeks and months, to optimize business operations. For instance, an analyst at a shipping company might detect trends correlating to departure time and total travel times. The analyst might offer the recommendation that the business should shift its delivery schedule forward by an hour to avoid traffic. To borrow a term from computer science, this kind of analysis occurs *asynchronously* relative to day-to-day operations. If the analyst calls in sick one day before finishing her report, the trucks still hit the road and the deliveries still happen at the normal time. What happens in the warehouses and on the roads that day is not tied to the outcome of any predictive model. It is not until someone reads the analyst's report and issues a company-wide memo that deliveries are to start one hour earlier that the results of the analysis trickle down to day-to-day operations.

Legacy data processing paradigms further entrench this separation between operations and analytics. Historically, limitations in both software and hardware necessitated the separation of transaction processing (INSERTs, UPDATEs, and DELETEs) from analytical data processing (queries that return some interpretable result without changing the underlying data). As the rest of this chapter

will discuss, modern data processing frameworks take advantage of distributed architectures and in-memory storage to enable the convergence of transactions and analytics.

To further motivate this discussion, envision a shipping network in which the schedules and routes are determined programmatically by using predictive models. The models might take weather and traffic data and combine them with past shipping logs to predict the time and route that will result in the most efficient delivery. In this case, day-to-day operations are contingent on the results of analytic predictive models. This kind of on-the-fly automated optimization is not possible when transactions and analytics happen in separate siloes.

Hybrid Data Processing Requirements

For a database management system to meet the requirements for converged transactional and analytical processing, the following criteria must be met:

Memory optimized
: Storing data in memory allows reads and writes to occur at real-time speeds, which is especially valuable for concurrent transactional and analytical workloads. In-memory operation is also necessary for converged data processing because no purely disk-based system can deliver the input/output (I/O) required for real-time operations.

Access to real-time and historical data
: Converging OLTP and OLAP systems requires the ability to compare real-time data to statistical models and aggregations of historical data. To do so, our database must accommodate two types of workloads: high-throughput operational transactions, and fast analytical queries.

Compiled query execution plans
: By eliminating disk I/O, queries execute so rapidly that dynamic SQL interpretation can become a bottleneck. To tackle this, some databases use a caching layer on top of their Relational Database Management System (RDBMS). However, this leads to cache invalidation issues that result in minimal, if any, performance benefit. Executing a query directly in memory is a better

approach because it maintains query performance (see Figure 2-1).

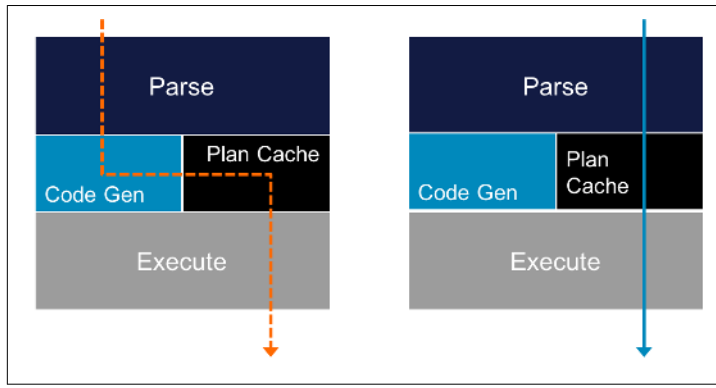

Figure 2-1. Compiled query execution plans

Multiversion concurrency control
: Reaching the high-throughput necessary for a hybrid, real-time engine can be achieved through lock-free data structures and multiversion concurrency control (MVCC). MVCC enables data to be accessed simultaneously, avoiding locking on both reads and writes.

Fault tolerance and ACID compliance
: Fault tolerance and Atomicity, Consistency, Isolation, Durability (ACID) compliance are prerequisites for any converged data system because datastores cannot lose data. A database should support redundancy in the cluster and cross-datacenter replication for disaster recovery to ensure that data is never lost.

With each of the aforementioned technology requirements in place, transactions and analytics can be consolidated into a single system built for real-time performance. Moving to a hybrid database architecture opens doors to untapped insights and new business opportunities.

Benefits of a Hybrid Data System

For data-centric organizations, a single engine to process transactions and analytics results in new sources of revenue and a simplified computing structure that reduces costs and administrative overhead.

New Sources of Revenue

Achieving true "real-time" analytics is very different from incrementally faster response times. Analytics that capture the value of data before it reaches a specified time threshold—often a fraction of a second—and can have a huge impact on top-line revenue.

An example of this can be illustrated in the financial services sector. Financial investors and analyst must be able to respond to market volatility in an instant. Any delay is money out of their pockets. Limitations with OLTP to OLAP batch processing do not allow financial organizations to respond to fluctuating market conditions as they happen. A single database approach provides more value to investors every second because they can respond to market swings in an instant.

Reducing Administrative and Development Overhead

By converging transactions and analytics, data no longer needs to move from an operational database to a siloed data warehouse to deliver insights. This gives data analysts and administrators more time to concentrate efforts on business strategy, as ETL often takes hours to days.

When speaking of in-memory computing, questions of data persistence and high availability always arise. The upcoming section dives into the details of in-memory, distributed, relational database systems and how they can be designed to guarantee data durability and high availability.

Data Persistence and Availability

By definition an operational database must have the ability to store information durably with resistance to unexpected machine failures. More specifically, an operational database must do the following:

- Save all of its information to disk storage for durability
- Ensure that the data is highly available by maintaining a readily accessible second copy of all data, and automatically fail-over without downtime in case of server crashes

These steps are illustrated in Figure 2-2.

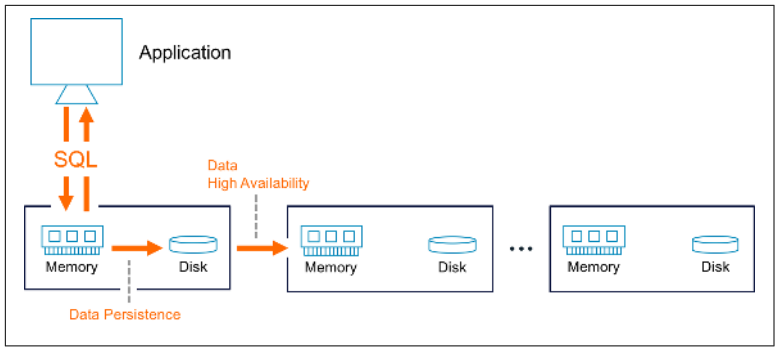

Figure 2-2. In-memory database persistence and high availability

Data Durability

For data storage to be durable, it must survive any server failures. After a failure, data should also be recoverable into a transactionally consistent state without loss or corruption to data.

Any well-designed in-memory database will guarantee durability by periodically flushing snapshots from the in-memory store into a durable disk-based copy. Upon a server restart, an in-memory database should also maintain transaction logs and replay snapshot and transaction logs.

This is illustrated through the following scenario:

Suppose that an application inserts a new record into a database. The following events will occur as soon as a commit is issued:

1. The inserted record will be written to the datastore in-memory.
2. A log of the transaction will be stored in a transaction log buffer in memory.
3. When the transaction log buffer is filled, its contents are flushed to disk.

   The size of the transaction log buffer is configurable, so if it is set to 0, the transaction log will be flushed to disk after each committed transaction.
4. Periodically, full snapshots of the database are taken and written to disk.

The number of snapshots to keep on disk and the size of the transaction log at which a snapshot is taken are configurable. Reasonable defaults are typically set.

An ideal database engine will include numerous settings to control data persistence, and will allow a user the flexibility to configure the engine to support full persistence to disk or no durability at all.

Data Availability

For the most part, in a multimachine system, it's acceptable for data to be lost in one machine, as long as data is persisted elsewhere in the system. Upon querying the data, it should still return a transactionally consistent result. This is where high availability enters the equation. For data to be highly available, it must be queryable from a system regardless of failures from some machines within a system.

This is better illustrated by using an example from a distributed system, in which any number of machines can fail. If failure occurs, the following should happen:

1. The machine is marked as failed throughout the system.
2. A second copy of data in the failed machine, already existing in another machine, is promoted to be the "master" copy of data.
3. The entire system fails over to the new "master" data copy, removing any system reliance on data present in the failed system.
4. The system remains online (i.e., queryable) throughout the machine failure and data failover times.
5. If the failed machine recovers, the machine is integrated back into the system.

A distributed database system that guarantees high availability must also have mechanisms for maintaining at least two copies of data at all times. Distributed systems should also be robust, so that failures of different components are mostly recoverable, and machines are reintroduced efficiently and without loss of service. Finally, distributed systems should facilitate cross-datacenter replication, allowing for data replication across wide distances, often times to a disaster recovery center offsite.

Data Backup

In addition to durability and high availability, an in-memory database system should also provide ways to create backups for the database. This is typically done by issuing a command to create on-disk copies of the current state of the database. Such backups can also be restored into both existing and new database instances in the future for historical analysis and long-term storage.

CHAPTER 3
Dawn of the Real-Time Dashboard

Before delving further into the systems and techniques that power predictive analytics applications, human consumption of analytics merits further discussion. Although this book focuses largely on applications using machine learning models to make decisions autonomously, we cannot forget that it is ultimately humans designing, building, evaluating, and maintaining these applications. In fact, the emergence of this type of application only increases the need for trained data scientists capable of understanding, interpreting, and communicating how and how well a predictive analytics application works.

Moreover, despite this book's emphasis on operational applications, more traditional human-centric, report-oriented analytics will not go away. If anything, its value will only increase as data processing technology improves, enabling faster and more sophisticated reporting. Improvements like reduced Extract, Transform, and Load (ETL) latency and faster query execution empowers data scientists and increases the impact they can have in an organization.

Data visualization is arguably the single most powerful method for enabling humans to understand and spot patterns in a dataset. No one can look at a spreadsheet with thousands or millions of rows and make sense of it. Even the results of a database query, meant to summarize characteristics of the dataset through aggregation, can be difficult to parse when it is just lines and lines of numbers. Moreover, visualizations are often the best and sometimes only way to communicate findings to a nontechnical audience.

Business Intelligence (BI) software enables analysts to pull data from multiple sources, aggregate the data, and build custom visualizations while writing little or no code. These tools come with templates that allow analysts to create sophisticated, even interactive, visualization without being expert frontend programmers. For example, an online retail site deciding which geographical region to target its next ad campaign could look at all user activity (e.g., browsing and purchases) in a geographical map. This will help it to visually recognize where user activity is coming from and make better decisions regarding which region to target. An example of such a visualization is shown in Figure 3-1.

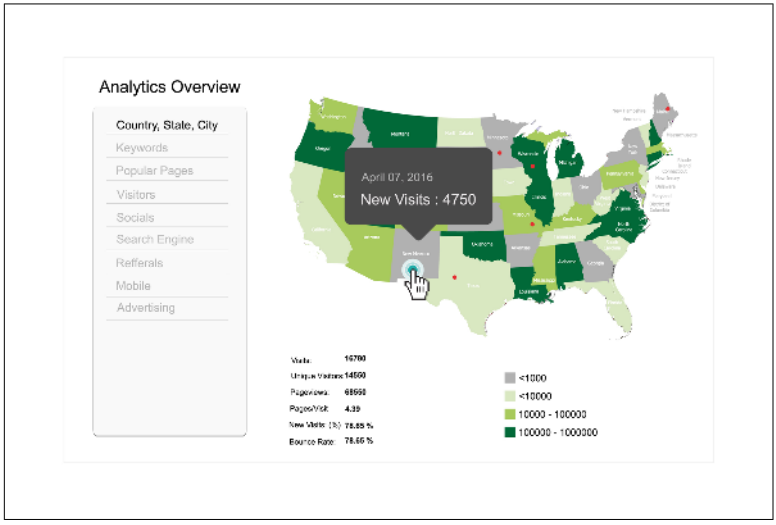

Figure 3-1. Sample geographic visualization dashboard

Other related visualizations for an online retail site could be a bar chart that shows the distribution of web activity throughout the different hours of each day, or a pie chart that shows the categories of products purchased on the site over a given time period.

Historically, out-of-the-box visual BI dashboards have been optimized for data warehouse technologies. Data warehouses typically require complex ETL jobs that load data from real-time systems, thus creating latency between when events happen and when information is available and actionable. As described in the last chapters, technology has progressed—there are now modern databases capable of ingesting large amounts of data and making that data immediately actionable without the need for complex ETL jobs. Fur-

thermore, visual dashboards exist in the market that accommodate interoperability with real-time databases.

Choosing a BI Dashboard

Choosing a BI dashboard must be done carefully depending on existing requirements in your enterprise. This section will not make specific vendor recommendations, but it will cite several examples of real-time dashboards.

For those who choose to go with an existing, third-party, out-of-the-box BI dashboard vendor, here are some things to keep in mind:

Real-time dashboards allow instantaneous queries to the underlying data source
: Dashboards that are designed to be real-time must be able to query underlying sources in real-time, without needing to cache any data. Historically, dashboards have been optimized for data warehouse solutions, which take a long time to query. To get around this limitation, several BI dashboards store or cache information in the visual frontend as a performance optimization, thus sacrificing real-time in exchange for performance.

Real-time dashboards are easily and instantly shareable
: Real-time dashboards facilitate real-time decision making, which is enabled by how fast knowledge or insights from the visual dashboard can be shared to a larger group to validate a decision or gather consensus. Hence, real-time dashboards must be easily and instantaneously shareable; ideally hosted on a public website that allows key stakeholders to access the visualization.

Real-time dashboards are easily customizable and intuitive
: Customizable and intuitive dashboards are a basic requirement for all good BI dashboards, and this condition is even more important for real-time dashboards. The easier it is to build and modify a visual dashboard, the faster it would be to take action and make decisions.

Real-Time Dashboard Examples

The rest of this chapter will dive into more detail around modern dashboards that provide real-time capabilities out of the box. Note that the vendors described here do not represent the full set of BI dashboards in the market. The point here is to inform you of possible solutions that you can adopt within your enterprise. The aim of describing the following dashboards is not to recommend one over the other. Building custom dashboards will be covered later in this chapter.

Tableau

As far as BI dashboard vendors are concerned, Tableau has among the largest market share in the industry. Tableau has a desktop version and a server version that either your company can host or Tableau can host for you (i.e., Tableau Online). Tableau can connect to real-time databases such as MemSQL with an out-of-the-box connector or using the MySQL protocol connector. Figure 3-2 shows a screenshot of an interactive map visualization created using Tableau.

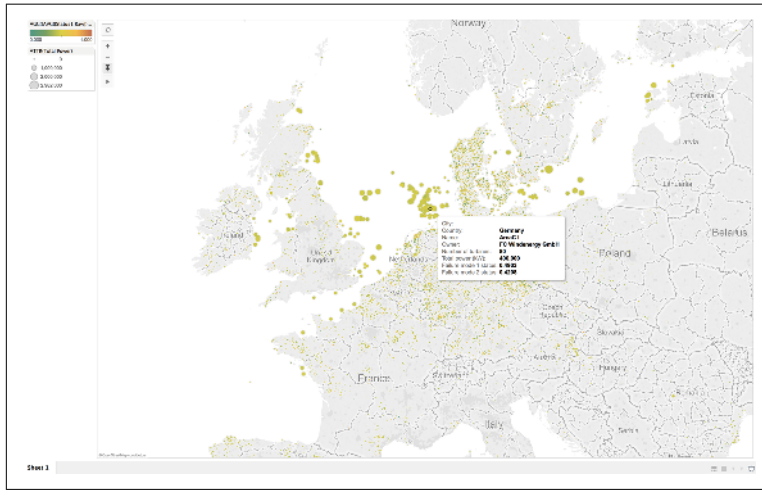

Figure 3-2. Tableau dashboard showing geographic distribution of wind farms in Europe

Zoomdata

Among the examples given in this chapter, Zoomdata facilitates real-time visualization most efficiently, allowing users to configure zero data cache for the visualization frontend. Zoomdata can connect to real-time databases such as MemSQL with an out-of-the-box connector or the MySQL protocol connector. Figure 3-3 presents a screenshot of a custom dashboard showing taxi trip information in New York City, built using Zoomdata.

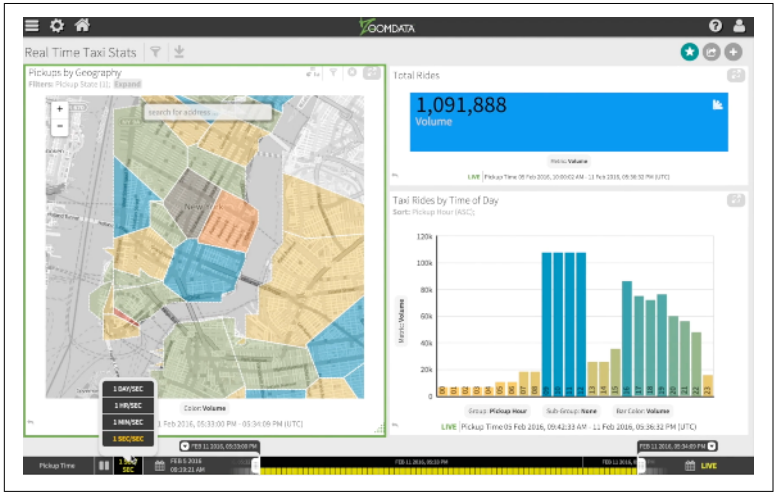

Figure 3-3. Zoomdata dashboard showing taxi trip information in New York City

Looker

Looker is another powerful BI tool that helps you to create real-time dashboards with ease. Looker also utilizes its own custom language, called LookML, for describing dimensions, fields, aggregates and relationships in a SQL database. The Looker app uses a model written in LookML to construct SQL queries against SQL databases, like MemSQL. Figure 3-4 is an example of an exploratory visualization of orders in an online retail store.

These examples are excellent starting points for users looking to build real-time dashboards.

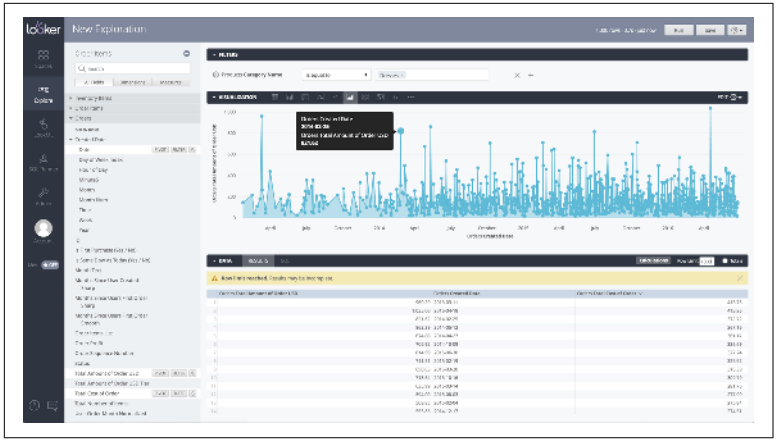

Figure 3-4. Looker dashboard showing a visualization of orders in an online retail store

Building Custom Real-Time Dashboards

Although out-of-the-box BI dashboards provide a lot of functionality and flexibility for building visual dashboards, they do not necessarily provide the required performance or specific visual features needed for your enterprise use case. Furthermore, these dashboards are also separate pieces of software, incurring extra cost and requiring you to work with a third-party vendor to support the technology. For specific real-time analysis use cases for which you know exactly what information to extract and visualize from your real-time data pipeline, it is often faster and cheaper to build a custom real-time dashboard in-house instead of relying on a third-party vendor.

Database Requirements for Real-Time Dashboards

Building a custom visual dashboard on top of a real-time database requires that the database have the characteristics detailed in the following subsections.

Support for various programming languages

The choice of which programming language to use for a custom real-time dashboard is at the discretion of the developers. There is no "proper" programming language or protocol that is best for developing custom real-time dashboards. It is recommended to go

with what your developers are familiar with, and what your enterprise has access to. For example, several modern custom real-time dashboards are designed to be opened in a web browser, with the dashboard itself built with a JavaScript frontend, and websocket connectivity between the web client and backend server, communicating with a performant relational database.

All real-time databases must provide clear interfaces through which the custom dashboard can interact. The best programmatic interfaces are those based on known standards, and those that already provide native support for a variety of programming languages. A good example of such an interface is SQL. SQL is a known standard with a variety of interfaces for popular programming languages—Java, C, Python, Ruby, Go, PHP, and more. Relational databases (full SQL databases) facilitate easy building of custom dashboards by allowing the dashboards to be created using almost any programming language.

Fast data retrieval

Good visual real-time dashboards require fast data retrieval in addition to fast data ingest. When building real-time data pipelines, the focus tends to be on the latter, but for real-time data visual dashboards, the focus is on the former. There are several databases that have very good data ingest rates but poor data retrieval rates. Good real-time databases have both. A real-time dashboard is only as "real-time" as the speed that it can render its data, which is a function of how fast the data can be retrieved from the underlying database. It also should be noted that visual dashboards are typically interactive, which means the viewer should be able to click or drill down into certain aspects of the visualizations. Drilling down typically requires retrieving more data from the database each time an action is taken on the dashboard's user interface. For those clicks to return quickly, data must be retrieved quickly from the underlying database.

Ability to combine separate datasets in the database

Building a custom visual dashboard might require combining information of different types coming from different sources. Good real-time databases should support this. For example, consider building a custom real-time visual dashboard from an online commerce website that captures information about the products sold, customer

reviews, and user navigation clicks. The visual dashboard built for this can contain several charts—one for popular products sold, another for top customers, and one for the top reviewed products based on customer reviews. The dashboard must be able to join these separate datasets. This data joining can happen within the underlying database or in the visual dashboard. For the sake of performance, it is better to join within the underlying database. If the database is unable to join data before sending it to the custom dashboard, the burden of performing the join will fall to the dashboard application, which leads to sluggish performance.

Ability to store real-time and historical datasets

The most insightful visual dashboards are those that are able to display lengthy trends and future predictions. And the best databases for those dashboards store both real-time and historical data in one database, with the ability to join the two. This present and past combination provides the ideal architecture for predictive analytics.

CHAPTER 4

Redeploying Batch Models in Real Time

For all the greenfield opportunities to apply machine learning to business problems, chances are your organization already uses some form of predictive analytics. As mentioned in previous chapters, traditionally analytical computing has been batch oriented in order to work around the limitations of ETL pipelines and data warehouses that are not designed for real-time processing. In this chapter, we take a look at opportunities to apply machine learning to real-time problems by repurposing existing models.

Future opportunities for machine learning and predictive analytics span infinite possibilities, but there is still an incredible amount of easily accessible opportunities today. These come by applying existing batch processes based on statistical models to real-time data pipelines. The good news is that there are straightforward ways to accomplish this that quickly put the business rapidly ahead. Even for circumstances in which batch processes cannot be eliminated entirely, simple improvements to architectures and data processing pipelines can drastically reduce latency and enable businesses to update predictive models more frequently and with larger training datasets.

Batch Approaches to Machine Learning

Historically, machine learning approaches were often constrained to batch processing. This resulted from the amount of data required

for successful modeling, and the restricted performance of traditional systems.

For example, conventional server systems (and the software optimized for those systems) had limited processing power such as a set number of CPUs and cores within a single server. Those systems also had limited high-speed storage, fixed memory footprints, and namespaces confined to a single server.

Ultimately these system constraints led to a choice: either process a small amount of data quickly or process large amounts of data in batches. Because machine learning relies on historical data and comparisons to train models, a batch approach was frequently chosen (see Figure 4-1).

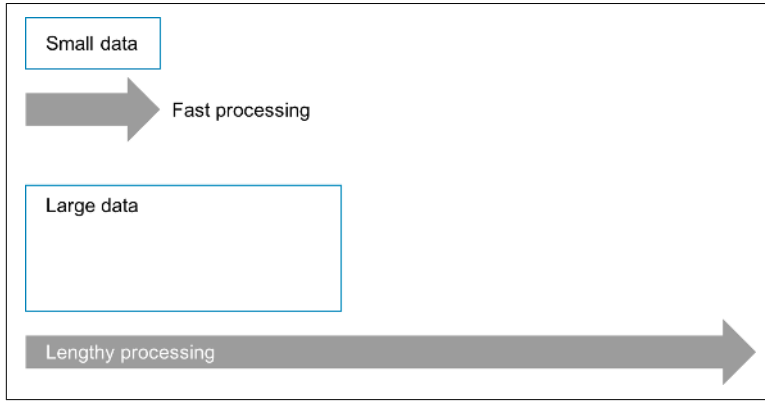

Figure 4-1. Batch approach to machine learning

With the advent of distributed systems, initial constraints were removed. For example, the Hadoop Distributed File System (HDFS) provided a plentiful approach to low-cost storage. New scalable streaming and database technologies provided the ability to process and serve data in real time. Coupling these systems together provides both a real-time and batch architecture.

This approach is often referred to as a *Lambda architecture*. A Lambda architecture often consists of three layer: a speed layer, a batch layer, and a serving layer, as illustrated in Figure 4-2.

The advantage to Lambda is a comprehensive approach to batch and real-time workflows. The disadvantage is that maintaining two pipelines can lead to excessive management and administration to achieve effective results.

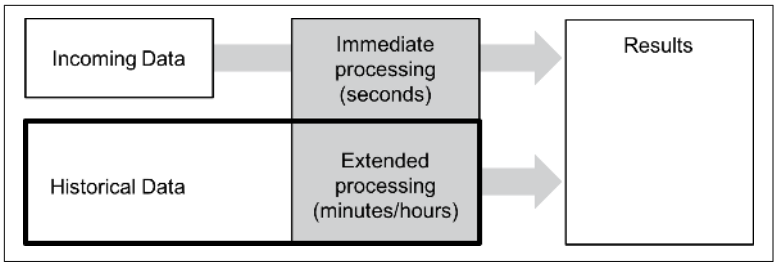

Figure 4-2. Lambda architecture

Moving to Real Time: A Race Against Time

Although not every application requires real-time data, virtually every industry requires real-time solutions. For example, in real estate, transactions do not necessarily need to be logged to the millisecond. However, when every real estate transaction is logged to a database, and a company wants to provide ad hoc access to that data, a real-time solution is likely required.

Other areas for machine learning and predictive analytics applications include the following:

- Information assets
  - — Optimizing commerce, recommendations, preferences
- Manufacturing assets
  - — Driving cost efficiency and system productivity
- Distribution assets
  - — Ensuring comprehensive fulfillment

Let's take a look at manufacturing as just one example.

Manufacturing Example

Manufacturing is often a high-stakes, high–capital investment, high-scale production operation. We see this across mega-industries including automotive, electronics, energy, chemicals, engineering, food, aerospace, and pharmaceuticals.

Companies will frequently collect high-volume sensor data from sources such as these:

- Manufacturing equipment
- Robotics
- Process measurements
- Energy rigs
- Construction equipment

The sensor information provides readings on the health and efficiency of the assets, and is critical in areas of high capital expenditure combined with high operational expenditure.

Let's consider the application of an energy rig. With drill bit and rig costs ranging in the millions, making use of these assets efficiently is paramount.

Original Batch Approach

Energy drilling is a high-tech business. To optimize the direction and speed of drill bits, energy companies collect information from the bits on temperature, pressure, vibration, and direction to assist in determining the best approach.

Traditional pipelines involve collecting drill bit information and sending that through a traditional enterprise message bus, overnight batch processing, and guidance for the next day's operations. Companies frequently rely on statistical modeling software from companies like SAS to provide analytics on sensor information. Figure 4-3 offers an example of an original batch approach.

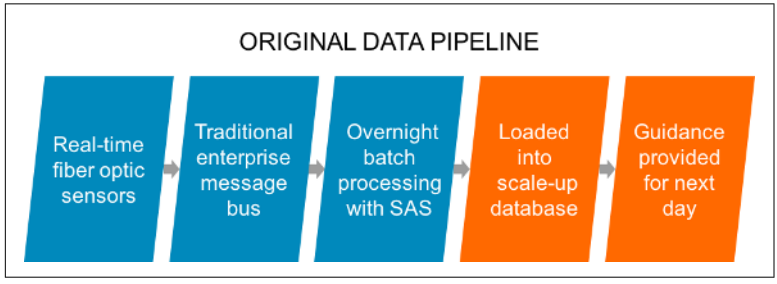

Figure 4-3. Original batch approach

Real-Time Approach

To improve operations, energy companies seek easier facilitation of adding and adjusting new data pipelines. They also desire the ability to process both real-time and historical data within a single system to avoid ETL, and they want real-time scoring of existing models.

By shifting to a real-time data pipeline supported by Kafka, Spark, and an in-memory database such as MemSQL, these objectives are easily reached (see Figure 4-4).

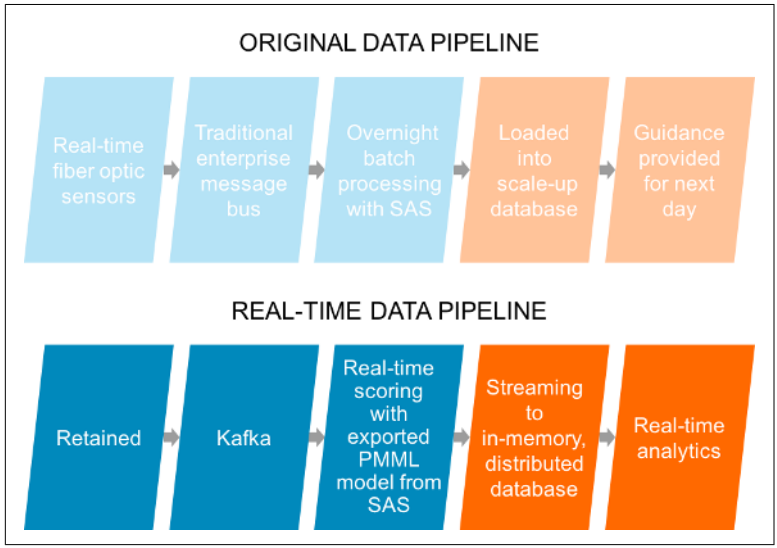

Figure 4-4. Real-time data pipeline supported by Kafka, Spark, and in-memory database

Technical Integration and Real-Time Scoring

The new real-time solution begins with the same sensor inputs. Typically, the software for edge sensor monitoring can be directed to feed sensor information to Kafka.

After the data is in Kafka, it is passed to Spark for transformation and scoring. This step is the crux of the pipeline. Spark enables the scoring by running incoming data through existing models.

In this example, an SAS model can be exported as Predictive Model Markup Language (PMML) and embedded inside the pipeline as part of a Java Archive (JAR) file.

After the data has been scored, both the raw sensor data and the results of the model on that data are saved in the database in the same table.

When real-time scoring information is colocated with the sensor data, it becomes immediately available for query without the need for precomputing or batch processing.

Immediate Benefits from Batch to Real-Time Learning

The following are some of the benefits of a real-time pipeline designed as described in the previous section:

Consistency with existing models
: By using existing models and bringing them into a real-time workflow, companies can maintain consistency of modeling.

Speed to production
: Using existing models means more rapid deployment and an existing knowledge base around those models.

Immediate familiarity with real-time streaming and analytics
: By not changing models, but changing the speed, companies can get immediate familiarity with modern data pipelines.

Harness the power of distributed systems
: Pipelines built with Kafka, Spark, and MemSQL harness the power of distributed systems and let companies benefit from the flexibility and performance of such systems. For example, companies can use readily available industry standard servers, or cloud instances to stand up new data pipelines.

Cost savings
: Most important, these real-time pipelines facilitate dramatic cost savings. In the case of energy drilling, companies need to determine the health and efficiency of the drilling operation. Push a drill bit too far and it will break, costing millions to replace and lost time for the overall rig. Retire a drill bit too early and money is left on the table. Going to a real-time model lets companies make use of assets to their fullest extent without pushing too far to cause breakage or a disruption to rig operations.

CHAPTER 5

Applied Introduction to Machine Learning

Even though the forefront of artificial intelligence research captures headlines and our imaginations, do not let the esoteric reputation of machine learning distract from the full range of techniques with practical business applications. In fact, the power of machine learning has never been more accessible. Whereas some especially oblique problems require complex solutions, often, simpler methods can solve immediate business needs, and simultaneously offer additional advantages like faster training and scoring. Choosing the proper machine learning technique requires evaluating a series of tradeoffs like training and scoring latency, bias and variance, and in some cases accuracy versus complexity.

This chapter provides a broad introduction to applied machine learning with emphasis on resolving these tradeoffs with business objectives in mind. We present a conceptual overview of the theory underpinning machine learning. Later chapters will expand the discussion to include system design considerations and practical advice for implementing predictive analytics applications. Given the experimental nature of applied data science, the theme of flexibility will show up many times. In addition to the theoretical, computational, and mathematical features of machine learning techniques, the reality of running a business with limited resources, especially limited time, affects how you should choose and deploy strategies.

Before delving into the theory behind machine learning, we will discuss the problem it is meant to solve: enabling machines to make decisions informed by data, where the machine has "learned" to perform some task through exposure to training data. The main abstraction underpinning machine learning is the notion of a model, which is a program that takes an input data point and then outputs a prediction.

There are many types of machine learning models and each formulates predictions differently. This and subsequent chapters will focus primarily on two categories of techniques: supervised and unsupervised learning.

Supervised Learning

The distinguishing feature of supervised learning is that the training data is labeled. This means that, for every record in the training dataset, there are both features and a label. Features are the data representing observed measurements. Labels are either categories (in a classification model) or values in some continuous output space (in a regression model). Every record associates with some outcome.

For instance, a precipitation model might take features such as humidity, barometric pressure, and other meteorological information and then output a prediction about the probability of rain. A regression model might output a prediction or "score" representing estimated inches of rain. A classification model might output a prediction as "precipitation" or "no precipitation." Figure 5-1 depicts the two stages of supervised learning.

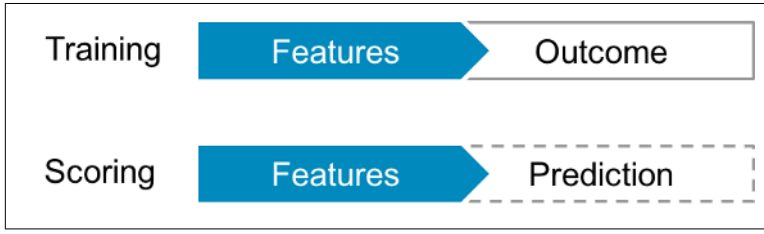

Figure 5-1. Training and scoring phases of supervised learning

"Supervised" refers to the fact that features in training data correspond to some observed outcome. Note that "supervised" does not refer to, and certainly does not guarantee, any degree of data quality. In supervised learning, as in any area of data science, discerning

data quality—and separating signal from noise—is as critical as any other part of the process. By interpreting the results of a query or predictions from a model, you make assumptions about the quality of the data. Being aware of the assumptions you make is crucial to producing confidence in your conclusions.

Regression

Regression models are supervised learning models that output results as a value in a continuous prediction space (as opposed to a classification model, which has a discrete output space). The solution to a regression problem is the function that best approximates the relationship between features and outcomes, where "best" is measured according to an error function. The standard error measurement function is simply Euclidian distance—in short, how far apart are the predicted and actual outcomes?

Regression models will never perfectly fit real-world data. In fact, error measurements approaching zero usually points to *overfitting*, which means the model does not account for "noise" or variance in the data. *Underfitting* occurs when there is too much bias in the model, meaning flawed assumptions prevent the model from accurately learning relationships between features and outputs.

Figure 5-2 shows some examples of different forms of regression. The simplest type of regression is linear regression, in which the solution takes the form of the line, plane, or hyperplane (depending on the number of dimensions) that best fits the data (see Figure 5-3). Scoring with a linear regression model is computationally cheap because the prediction function in linear, so scoring is simply a matter of multiplying each feature by the "slope" in that direction and then adding an intercept.

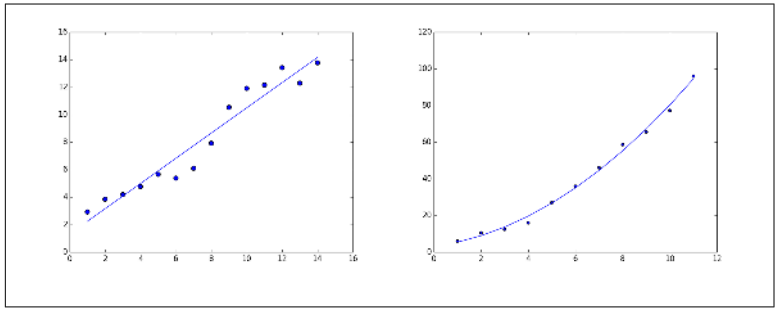

Figure 5-2. Examples of linear and polynomial regression

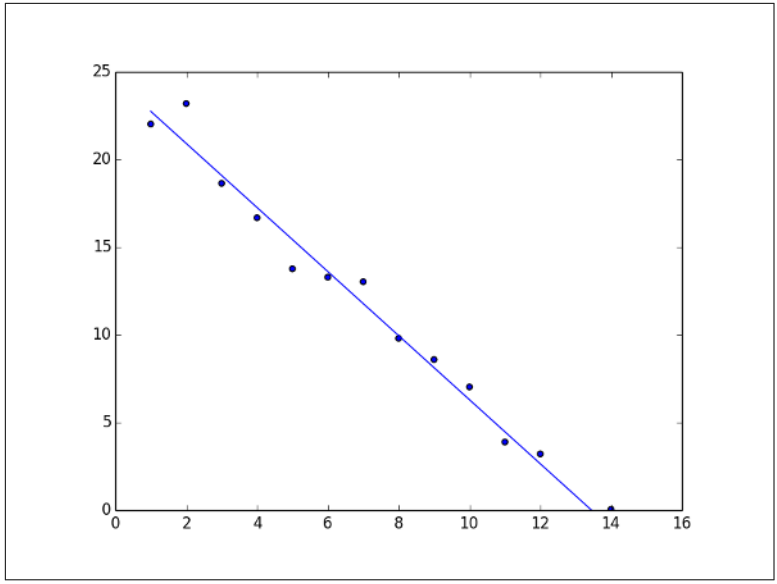

Figure 5-3. Linear regression in two dimensions

There are many types of regression and layers of categorization—this is true of many machine learning techniques. One way to categorize regression techniques is by the mathematical format of the solution. One form of solution is linear, where the prediction function takes the form of a line in two dimensions, and a plane or hyperplane in higher dimensions. Solutions in n dimensions take the following form:

$$a_1x_1 + a_2x_2 + ... + a_{n-1}x_{n-1} + b$$

One advantage of linear models is the ease of scoring. Even in high dimensions—when there are several features—scoring consists of just scalar addition and multiplication. Other regression techniques give a solution as a polynomial or a logistic function. The following table describes the characteristics of different forms of regression.

Regression model	Solution in two dimensions	Output space
Linear	$ax + b$	Continuous
Polynomial	$a_1x^n + a_2x^{n-1} + ... + a_nx + a_{n+1}$	Continuous
Logistic	$L/\left(1 + e^{-k(x-x_0)}\right)$	Continuous (e.g., population modeling) or discrete (binary categorical response)

It is also useful to categorize regression techniques by how they measure error. The format of the solution—linear, polynomial, logistic—does not completely characterize the regression technique. In fact, different error measurement functions can result in different solutions, even if the solutions take the same form. For instance, you could compute multiple linear regressions with different error measurement functions. Each regression will yield a linear solution, but the solutions can have different slopes or intercepts depending on error function.

The method of *least squares* is the most common technique for measuring error. In least-squares approaches, you compute the total error as the sum of squares of the errors the solution relative to each record in the training data. The "best fit" is the function that minimizes the sum of squared errors. Figure 5-4 is a scatterplot and regression function, with red lines drawn in representing the prediction error for a given point. Recall that the error is the distance between the predicted outcome and the actual outcome. The solution with the "best fit" is the one that minimizes the sum of each error squared.

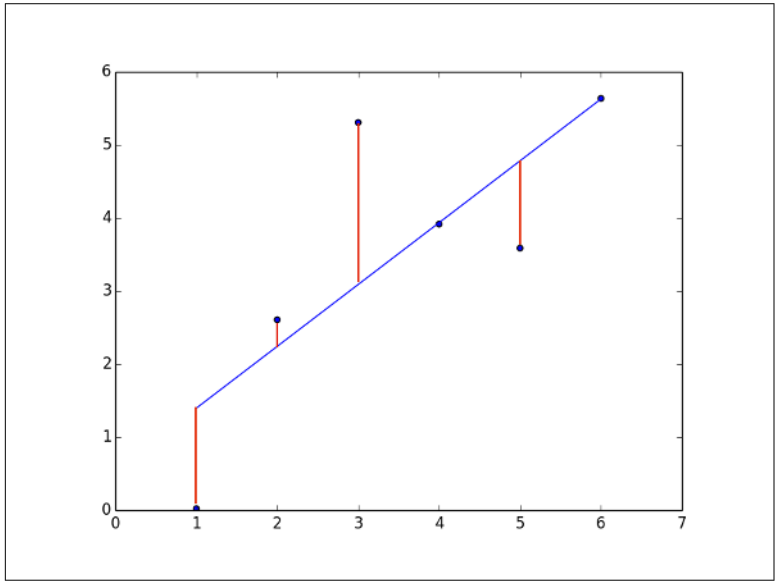

Figure 5-4. A linear regression, with red lines representing prediction error for a given training data point

Least squares is commonly associated with linear regression. In particular, a technique called Ordinary Least Squares is a common way of finding the regression solution with the best fit. However, least-squares techniques can be used with polynomial regression, as well. Whether the regression solution is linear or a higher degree polynomial, least squares is simply a method of measuring error. The format of the solution, linear or polynomial, determines what shape you are trying to fit to the data. However, in either case, the problem is still finding the prediction function that minimizes error over the training dataset.

Although Ordinary Least Squares provides a strong intuition for what the error measurement function represents, there are many ways of defining error in a regression problem. There are many variants on least-squares error function, such as weighted least squares, in which some observations are given more or less weight according to some metric that assesses data quality. There are also various approaches that fall under *regularization*, which is a family of techniques used to make solutions more generalizable rather than overfit to a particular training set. Popular techniques for regularized least squares includes Ridge Regression and LASSO.

Whether you're using the method of least squares or any other technique for quantifying error, there are two sources of error: bias, flawed assumptions in model that conceal relationships between the features and outcomes of a dataset, and variance, which is naturally occurring "noise" in a dataset. Too much bias in the model causes underfitting, whereas too much variance causes overfitting. Bias and variance tend to inversely correlate—when one goes up the other goes down—which is why data scientists talk about a "bias-variance tradeoff." Well-fit models find a balance between the two sources of error.

Classification

Classification is very similar to regression and uses many of the same underlying techniques. The main difference is the format of the prediction. The intuition for regression is that you're matching a line/plane/surface to approximate some trend in a dataset, and every combination of features corresponds to some point on that surface. Formulating a prediction is a matter of looking at the score at a given point. Binary classification is similar, except instead of predicting by using a point on the surface, it predicts one of two cate-

gories based on where the point resides relative to the surface (above or below). Figure 5-5 shows a simple example of a linear binary classifier.

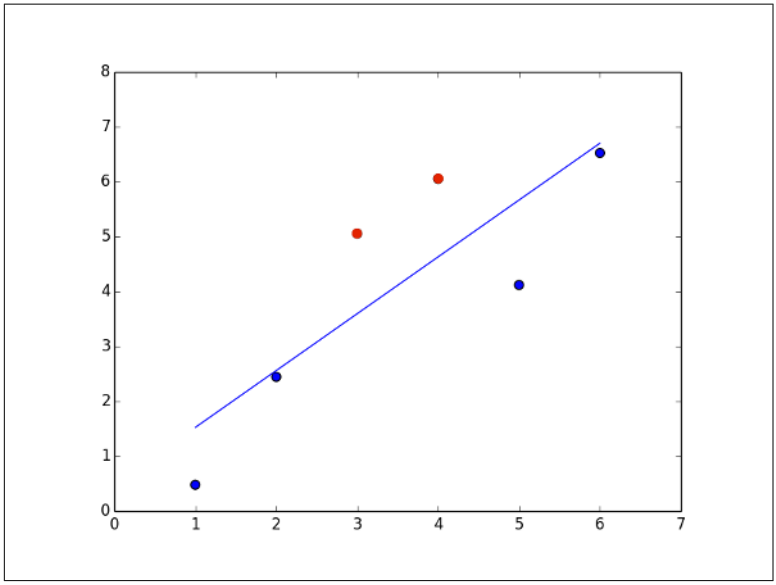

Figure 5-5. Linear binary classifier

Binary classification is the most commonly used and best-understood type of classifier, in large part because of its relationship with regression. There are many techniques and algorithms that are used for training both regression and classification models.

There are also "multiclass" classifiers, which can use more than two categories. A classic example of a multiclass classifier is a handwriting recognition program, which must analyze every character and then classify what letter, number, or symbol it represents.

Unsupervised Learning

The distinguishing feature of unsupervised learning is that data is unlabeled. This means that there are no outcomes, scores, or categorizations associated with features in training data. As with supervised learning, "unsupervised" does not refer to data quality. As in any area of data science, training data for unsupervised learning will not be perfect, and separating signal from noise is a crucial component of training a model.

The purpose of unsupervised learning is to discern patterns in data that are not known beforehand. One of its most significant applications is in analyzing clusters of data. What the clusters represent, or even the number of clusters, is often not known in advance of building the model. This is the fundamental difference between unsupervised and supervised learning, and why unsupervised learning is often associated with data mining—many of the applications for unsupervised learning are exploratory.

It is easy to confuse concepts in supervised and unsupervised learning. In particular, cluster analysis in unsupervised learning and classification in supervised learning might seem like similar concepts. The difference is in the framing of the problem and information you have when training a model. When posing a classification problem, you know the categories in advance and features in the training data are labeled with their associated categories. When posing a clustering problem, the data is unlabeled and you do not even know the categories before training the model.

The fundamental differences in approach actually create opportunities to use unsupervised and supervised learning methods together to attack business problems. For example, suppose that you have a set of historical online shopping data and you want to formulate a series of marketing campaigns for different types of shoppers. Furthermore, you want a model that can classify a wider audience, including potential customers with no purchase history.

This is a problem that requires a multistep solution. First you need to explore an unlabeled dataset. Every shopper is different and, although you might be able to recognize some patterns, it is probably not obvious how you want to segment your customers for inclusion in different marketing campaigns. In this case, you might apply an unsupervised clustering algorithm to find cohorts of products purchased together. Using this clustering information to your purchase data then allows you to build a supervised classification model that correlates purchasing cohort with other demographic information, allowing you to classify your marketing audience members without a purchase history. Using an unsupervised learning model to label data in order to build a supervised classification model is an example of *semi-supervised learning*.

Cluster Analysis

Cluster analysis programs detect patterns in the grouping of data. There are many approaches to clustering problems, but each has some measure of "closeness" and then optimizes arrangements of clusters to minimize the distance between points within a cluster.

Perhaps the easiest method of cluster analysis to grasp are *centroid-based* techniques like k-means. Centroid-based means that clusters are defined so as to minimize distances from a central point. The central point does not need to be a record in the training data—it can be any point in the training space. Figure 5-6 includes two scatterplots, the second of which includes three centroids (k = 3).

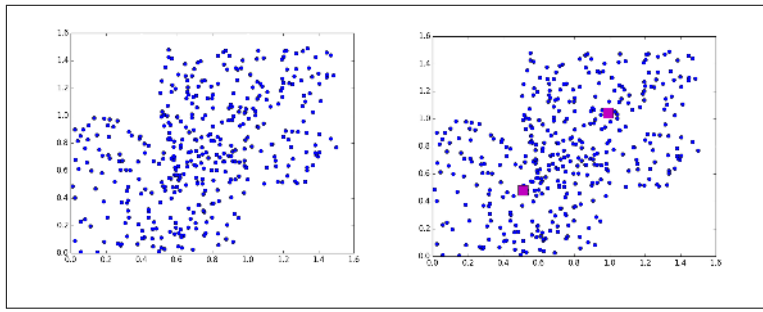

Figure 5-6. Sample clustering data with centroids determined by k-means

The "k" in k-means refers to the number of centroids. K-means algorithms iterate through values of k and choose optimal placements for the k centroids at each iteration so as to minimize the mean distance from training data points to centroid for each cluster. Figure 5-7 shows two examples of k-means applied to the same dataset but with different values of k.

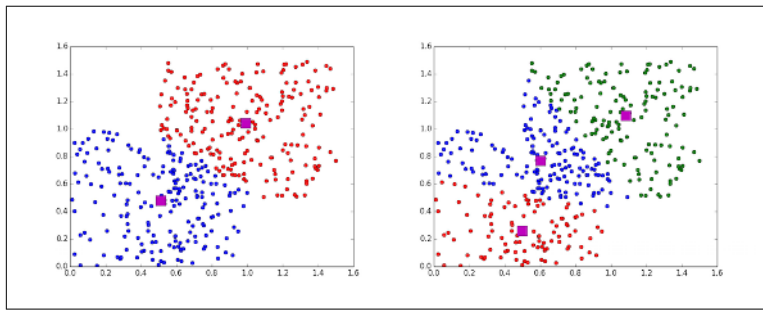

Figure 5-7. K-means examples with k = 2 and k = 3, respectively

There are many other methods for cluster analysis. Hierarchical methods derive sequences of increasingly large and inclusive clusters, as clusters combine with other nearby clusters at different stages of the model. These methods produce trees of clusters. On one end of the tree is a single node: one cluster that includes every point in the training. At the other end, every data point is its own cluster. Neither extreme is useful for analysis, but in between there is an entire series of options for dividing up the data into clusters. The method chooses an optimal depth in the hierarchy of clustering options.

Anomaly Detection

Although anomaly detection is its own area of study, one approach follows naturally from the discussion of clustering. Algorithms for clustering methods iterate through series of potential groupings; for example, k-means implementations iterate through values of k and assess them for fit. Because there is noise in any real world dataset, models that are not overfitted will leave outliers that are not part of any cluster.

Another class of methods (that still bears resemblance to cluster analysis) looks for local outliers that are unusually far away from their closest neighbors.

CHAPTER 6

Real-Time Machine Learning Applications

Combining terms like "real time" and "machine learning" runs the risk of drifting into the realm of buzz and away from business problems. However, improvements in real-time data processing systems and the ready availability of machine learning libraries make it so that applying machine learning to real-time problems is not only possible, but in many cases simply requires connecting the dots on a few crucial components.

The stereotype about data science is that its practitioners operate in silos, issuing declarations about data from on high, removed from the operational aspects of a business. This mindset reflects the latency and difficulty associated with legacy data analysis and processing toolchains. In contrast, modern database and data processing system design must embrace modularity and accessibility of data.

Real-Time Applications of Supervised Learning

The power of supervised learning applies to numerous business problems. Regression is familiar to any data scientist, finance or risk analyst, or anyone who took a statistics class in college. What has changed recently is the availability of powerful data processing soft-

ware that enables businesses to apply these tools to extremely low-latency problems.

Real-Time Scoring

Any system that automates data-driven decision making, for instance, will need to not only build and train a model, but use the model to score or make predictions. When developing a model, data scientists generally work in some development environment tailored for statistics and machine learning, such as Python, R, and Apache Spark. Using these tools, data scientists can train and test models all in one place and offer powerful abstractions for building models and making predictions while writing relatively little code.

Many data science tools are designed for interactive use by data scientists, rather than to power production systems.[1] See Figure 6-1 for an example of such interactive data analysis. Even though interactive tools are great for development, they are not designed for extremely low-latency production applications. For instance, a digital advertising network has on the order of milliseconds to choose and display an advertisement before a web page loads. In many cases, primarily interactive tools do not offer low enough latency for production use cases.

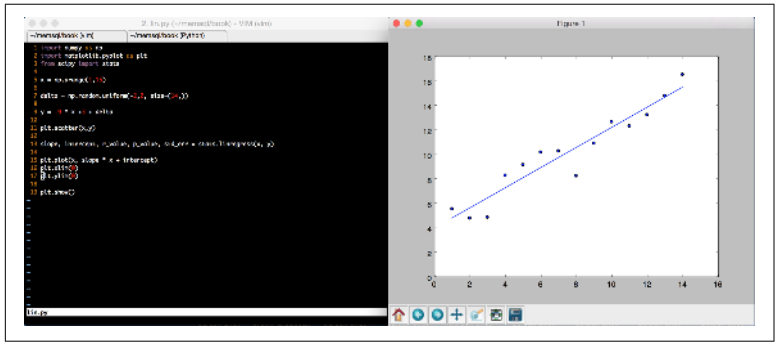

Figure 6-1. Interactive data analysis in Python

This is why when we talk about supervised learning we need to distinguish the way we think about scoring in development versus in

1 Apache Spark has a wide range of both production and development uses. The claim here is merely that starting and stopping Spark jobs is time and resource-intensive, which prevents it from being used as a real-time scoring engine.

production. Scoring latency in a development environment is generally just an annoyance. In production, the reduction or elimination of latency is a source of competitive advantage. This means that, when considering techniques and algorithms for predictive analytics, you need to evaluate not only the bias and variance of the model, but how the model will actually be used for scoring in production.

Fast Training and Retraining

Sometimes, real-world factors change (in real time) the phenomenon you are trying to model. When this happens, a production system can actually be bound by training latency rather than scoring latency. For instance, the relative weight, or ability to affect the overall outcome, of certain features might change. In this case, you cannot simply train a model once and expect it to give accurate answers forever. This scenario is especially common in "of the moment" trend spotting like in social media and digital advertising.

Deciding on the most efficient algorithm for training a given type of model exceeds the scope of this book. For many machine learning techniques, this remains an open question. However, there are straightforward system design considerations to enable fast training.

In-memory storage
: In-memory storage is a prerequisite for performing analytics on changing datasets. Locking and hardware contention make it difficult or impossible to manage fast-changing datasets on disk, let alone to make the data simultaneously available for machine learning.

Access to real-time and historical data
: A major component in modeling changing systems is understanding when your model needs to change. Making this decision requires simultaneous access to both the most recent data, and historical data on which the current and/or previous models are based. Whereas an offline data lake might satisfy the needs of early-stage model development, modeling a dynamic system requires faster access to data.

Convergence of systems
: Any large-scale data processing pipeline will include multiple systems, but limiting the number reduces latency associated

with data transfer and other intersystem communication. For instance, suppose that you want to write a program that builds a model using data from the last fraction of a second. By, for instance, converging scoring and transaction processing in a single database, you save valuable time that would have been lost to data transfer. For real-time applications like digital advertising, a fraction of a second is the entire window for the transaction.

Unsupervised Learning

Given the nature of unsupervised learning, its real-time applications can be less intuitive than the applications for supervised learning. Recall that unsupervised learning occurs on unlabeled data, which it is commonly associated with more offline tasks like data mining. Many unsupervised learning problems do not have an analogue to real-time scoring in a regression or classification model. However, advances in data processing technology have opened opportunities to use unsupervised learning techniques like clustering and anomaly detection in real-time capacities.

Real-Time Anomaly Detection

One of the most promising applications of real-time unsupervised learning is real-time anomaly detection, which you can use to strengthen monitoring applications; for example, Internet security and industrial machine data.

The nature of unsupervised learning in general, and anomaly detection in particular, is that you do not know exactly what you are looking for. Depending on the nature of the dataset and whether and how it changes over time, clusters and what constitutes an outlier can also change.

Suppose that you are monitoring network traffic. You might track information like the IP addresses, average response times, and amount of data sent over the network. The values of some of these features can change dramatically based on factors like amount of network traffic, and even factors completely beyond the scope of your system, such as an Internet Service Provider outage. Data that indicates a "normal" network user might be very different under unusual circumstances.

Real-Time Clustering

There are many scenarios for which solving a clustering problem will require frequent retraining. First, though, here's a corollary to the previous section about anomaly detection. As discussed in Chapter 5, clustering techniques are often used to detect anomalies because, in finding clusters of similar data, everything left out of a cluster is by definition an anomaly. There are also straightforward "clustering as clustering" real-time applications for which the groupings of data are changing. A high-traffic digital media company, for instance, might want to build a clustering model to determine which videos and articles are consumed together. Given modern news cycles and the dissemination of viral content, it is easy to envision the clustering of "related" videos and articles changing rapidly.

Even when the underlying phenomenon you are modeling is relatively static, meaning clusters are not shifting in real time, there can still be immense value to frequent retraining. As more training data becomes available, retraining with more information can yield different clustering results.

CHAPTER 7

Preparing Data Pipelines for Predictive Analytics and Machine Learning

Advances in data processing technology have changed the way we think about pipelines and what you can accomplish in real time. These advances also apply to machine learning—in many cases, making a predictive analytics application real-time is a question of infrastructure. Although certain techniques are better suited to real-time analytics and tight training or scoring latency requirements, the challenges preventing adoption are largely related to infrastructure rather than machine learning theory. And even though many topics in machine learning are areas of active research, there are also many useful techniques that are already well understood and can immediately provide major business impacts when implemented correctly.

Figure 7-1 shows a machine learning pipeline applied to a real-time business problem. The top row of the diagram represents the operational component of the application; this is where the model is applied to automate real-time decision making. For instance, a user accesses a web page, and the application must choose a targeted advertisement to display in the time it takes the page to load. When applied to real-time business problems, the operational component of the application always has restrictive latency requirement.

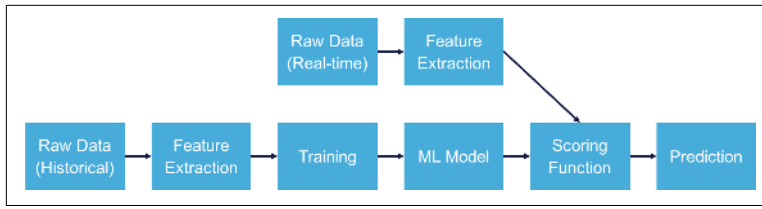

Figure 7-1. A typical machine learning pipeline powering a real-time application

The bottom row in Figure 7-1 represents the learning component of the application. It creates the model that the operational component will apply to make decisions. Depending on the application, the training component might have less stringent latency requirements—training is a one-time cost because scoring multiple data points does not require retraining. That said, many applications can benefit from reduced training latency by enabling more frequent retraining as dynamics of the underlying system change.

Real-Time Feature Extraction

Many businesses have the opportunity to dramatically improve their data intake systems. In the legacy data processing mindset, Extract, Transform, and Load (ETL) is an offline operation. We owe this assumption to the increasingly outdated convention of separating stream processing from transaction processing, and separating analytics from both. These separations were motivated largely by the latency that comes with disk-based, single-machine systems. The emergence of in-memory and distributed data processing systems has changed the way we think about datastores. See Figure 7-2 for an example of real-time pipelines.

Modern stream processing and database systems speed up and simplify the process of annotating data with, for example, additional features or labels for supervised learning.

- Use a stream processing engine to add (or remove) features before committing data to a database. Although you can certainly "capture now, analyze later," there are clear advantages to preprocessing data so that it can be analyzed as soon as it arrives in the datastore. Often, it is valuable to add metadata like timestamps, location information, or other information that is not contained in the original record.

- Use a real-time database to update records as more information becomes available. Although append-only databases (no updates or deletes) enjoyed some popularity in years past, this approach creates multiple records corresponding to a single entity. Even though this is not inherently flawed, this approach will require more time-consuming queries to retrieve information.

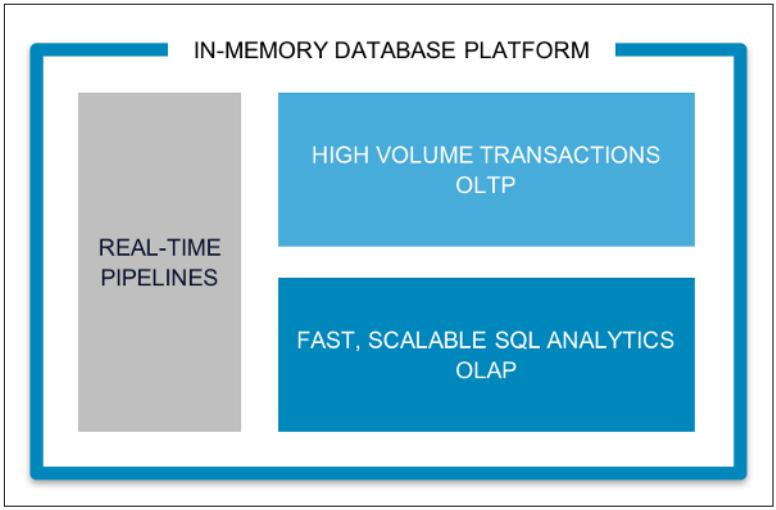

Figure 7-2. Real-time pipelines—OLTP and OLAP

Deciding when to annotate real-time data touches on issues relating to data normalization and when to store versus when to compute a particular value.

Minimizing Data Movement

The key to implementing real-time predictive analytics applications is minimizing the number of systems involved and the amount of data movement between them. Even if you carefully optimize your application code, real-time applications do not have time for unnecessary data movement. What constitutes "unnecessary" varies from case to case, but the naive solution usually falls under this category.

When first building a model, you begin with a large dataset and perform tasks like feature selection, which is the process of choosing features of the available data that will be used to train the model. For subsequent retrainings, there is no need to operate on the entire

dataset. Pushing computation to the database reduces the amount of data transferred between systems. VIEWs provide a convenient abstraction for limiting data transfer. Although traditionally associated with data warehousing workloads, VIEWs into a real-time database can perform data transformations on the fly, dramatically speeding up model training time. Advances in query performance, specifically the emergence of just-in-time compiled query plans, further expand the real-time possibilities for on-the-fly data preprocessing with VIEWs.

Scoring with supervised learning models presents another opportunity to push computation to a real-time database. Whereas it might be tempting to perform scoring in the same interactive environment where you developed the model, in most cases this will not satisfy the latency requirements of real-time use cases. Depending on the type of model, often the scoring function can be implemented in pure SQL. Then, instead of using separate systems for scoring and transaction processing, you do both in the database.

Dimensionality Reduction

Dimensionality reduction is an array of technique for reducing and simplifying the input space for a model. This can include simple feature selection, which simply removes some features because they are uncorrelated (i.e., superfluous) or otherwise impede training. Dimensionality reduction is also used for underdetermined systems for which there are more variables than there are equations defining their relationships to one another. Even if all of the features are potentially "useful" for modeling, finding a unique solution to the system requires reducing the number of variables (features) to be no more than the number of equations.

There are many dimensionality reduction techniques with different applications and various levels of complexity. The most widely used technique is Principal Component Analysis (PCA), which linearly maps the data into a lower dimensional space that preserves as much total variance as possible while removing features that contribute less variance. PCA transforms the "most important" features into a lower-dimension representation called principal components. The final result is an overdetermined system of equations (i.e., there is a unique solution) in a lower-dimension space with minimal information loss.

There are many other dimensionality reduction techniques including Linear Discriminant Analysis (LDA), Canonical Correlation Analysis (CCA), and Singular Value Decomposition (SVD). Each of the techniques mentioned so far transforms data linearly into the lower-dimensional space. There are also many nonlinear methods including a nonlinear generalization of PCA. These nonlinear techniques are also referred to as "manifold learning." Other common nonlinear techniques include Sammon Mapping, Diffusion Mapping, and Kernel PCA.

CHAPTER 8

Predictive Analytics in Use

There will be numerous applications of predictive analytics and machine learning to real-time challenges with adoption far and wide.

Expanding on the earlier discussion about taking machine learning from batch to real-time machine learning, in this chapter, we explore another use case, this one specific to the Internet of Things (IoT) and renewable energy.

Renewable Energy and Industrial IoT

The market around the Industrial Internet of Things (IIoT) is rapidly expanding, and is set to reach $150 billion in 2020, according to research firm Markets and Markets. According to the firm,

> IIoT is the integration of complex physical machinery with industrial networks and data analytics solutions to improve operational efficiency and reduce costs.

Renewable energy is an equally high growth sector. In mid-2016, Germany announced that it had developed almost all of its power from renewable energy ("Germany Just Got Almost All of Its Power From Renewable Energy," May 15, 2016).

The overall global growth in renewable energy continues at a breakneck pace, with the investment in renewables reaching $286 billion worldwide in 2015 (BBC).

PowerStream: A Showcase Application of Predictive Analytics for Renewable Energy and IIoT

PowerStream is an application that predicts the global health of wind turbines. It tracks the status of nearly 200,000 wind turbines around the world across approximately 20,000 wind farms. With each wind turbine reporting multiple sensor updates per second, the entire workload ranges between 1 to 2 million inserts per second.

The goal of the showcase is to understand the sensor readings and predict the health of the turbines. The technical crux is the volume of data as well as its continuous, dynamic flow. The underlying data pipeline needs to be able to capture and process this in real-time.

PowerStream Software Architecture

The PowerStream software architecture follows a similar architecture as you've seen in earlier chapters, which you can see in Figure 8-1.

PowerStream Hardware Configuration

With the power of distributed systems, you can implement rich functionality on a small consolidated hardware footprint. For example, the PowerStream architecture supports high-volume real-time data pipelines across the following:

- A message queue
- A transformation and scoring engine, Spark
- A stateful, persistent, memory-optimized database
- A graphing and visualization layer

The entire pipeline runs on just seven cloud server instances, in this case Amazon Web Services (AWS) C-2x large. At roughly $0.31 per hour per machine, the annual hardware total is approximately $19,000 for AWS instances.

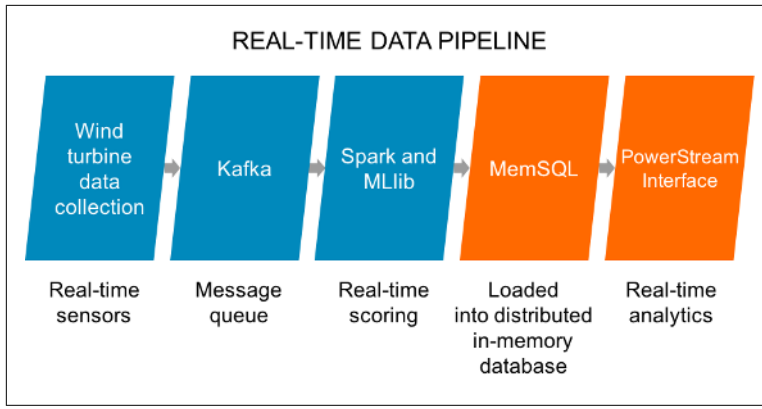

Figure 8-1. PowerStream software architecture

The capability to handle such a massive task with such a small hardware footprint is a testament to the advances in distributed and memory-optimized systems.

PowerStream Application Introduction

The basics of the interface include a visualization layer. The windfarm and turbine colors represent state based on data sent from the turbines to Kafka, then through Streamliner for machine learning with Spark, then on to MemSQL. This stream begins with integers and tuples (doubles?) emitted from each turbine.

After data is in a memory-optimized database, you can take interesting approaches such as building applications on that live, streaming data. In PowerStream, we use a developer-friendly mapping tool called Mapbox which provides a mapping layer embedded in a browser. Mapbox has an API and enables rapid development of dynamic applications.

For example, zooming in to the map (see Figure 8-2) allows for inspection from the windfarm down to the turbine level, and even seeing values of individual sensors.

One critical juncture of this application is the connection between Spark and the database, enabled by the MemSQL Spark Connector. The performance and low latency comes from both systems being distributed and memory optimized, so working with Resilient Distributed Datasets and Dataframes becomes very efficient as each node in one cluster communicates with each node in another clus-

ter. This speed makes building an application easier. The time from ingest, to real-time scoring, and then being saved to a relational database can easily be less than one second. At that point, any SQL-capable application is ready to go.

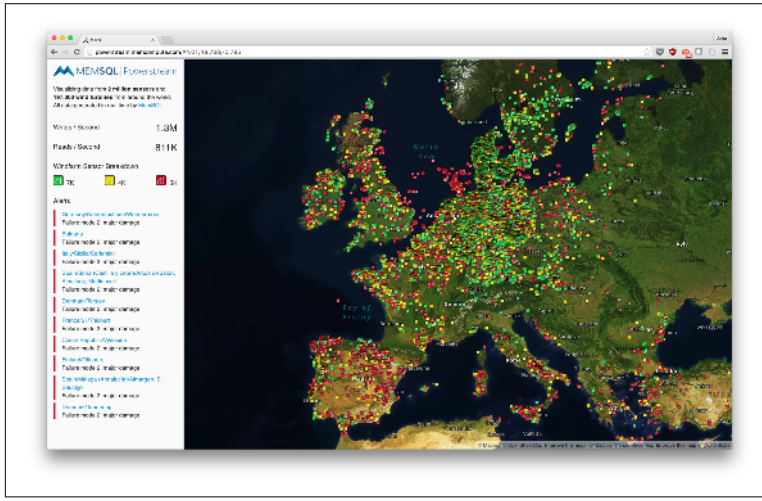

Figure 8-2. PowerStream application introduction

Adding the machine learning component delivers a rich set of applications and alerts with many types of sophisticated logic flowing in real time.

For example, we classify types of potential failures by predicting which turbines are failing and, more specifically, how they are failing. Understandably, the expense of a high-cost turbine not working is significant. Predictive alerts enable energy utilities to deploy workforces and spare parts efficiently.

Because data in a relational, SQL-capable database is extremely accessible to most enterprises, a rich set of Business Intelligence (BI) tools experience the immediate benefit of real-time data. For example, the screenshots presented in Figure 8-3 and Figure 8-4 show what is possible with Tableau.

And when failures are introduced into the system, the dashboard changes to a mix of yellow and red.

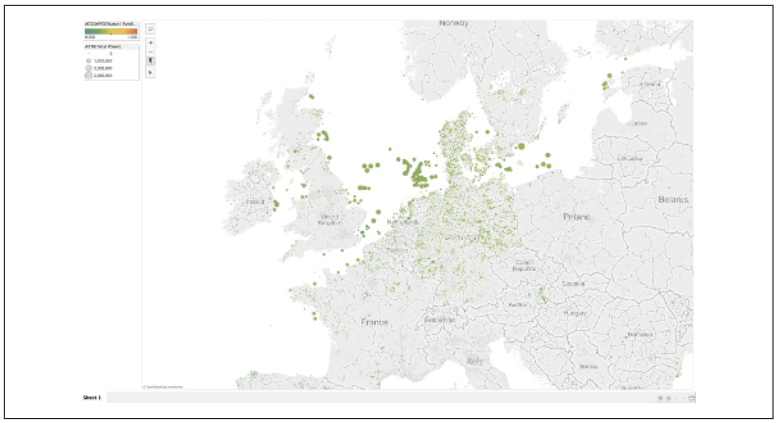

Figure 8-3. PowerStream interface using Tableau—status: healthy

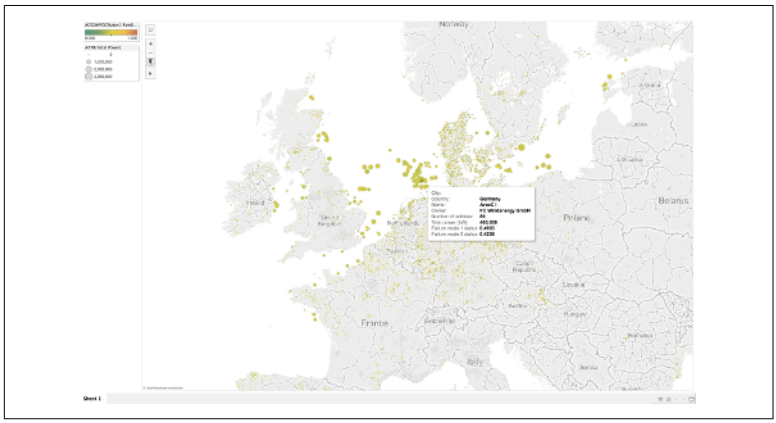

Figure 8-4. PowerStream interface using Tableau—status: warnings

PowerStream Details

One layer down in the application, updates come from simulated sensor values. Turbine locations are accurate based on a global database, with concentrations in countries like Germany, which is known for renewable energy success.

PowerStream sets a column in the turbine table to a value. Based on that value, the application can predict if the turbine is about to fail. Yellow or red indicates different failure states.

Examining the specific pipelines, we see one for scoring called ML and one for alerts (see Figure 8-5).

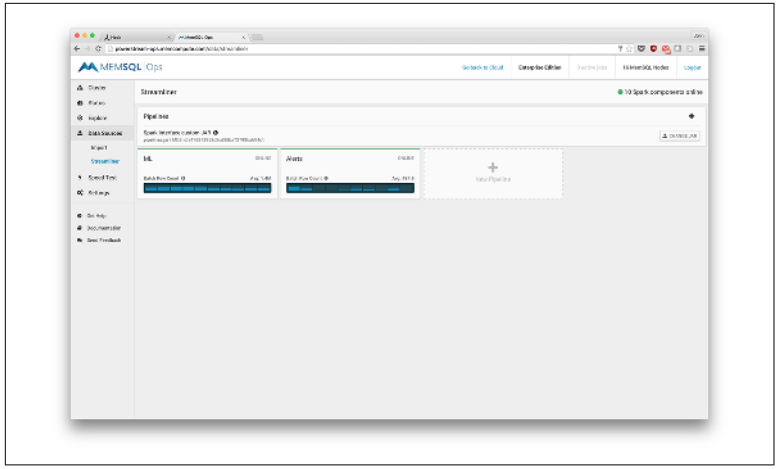

Figure 8-5. ML and Alerts pipelines

The ML pipeline shows throughput in the 1 to 2–million-transaction-per-second range (see Figure 8-6).

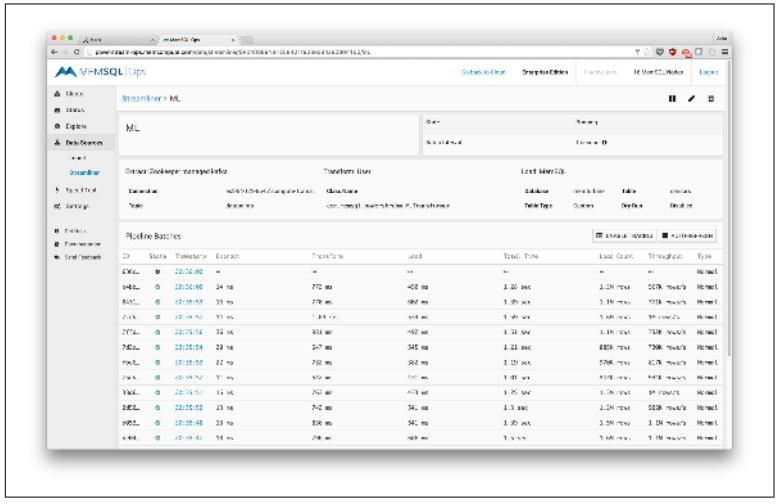

Figure 8-6. The ML pipeline showing throughput in the 1 to 2 million transactions-per-second range

The Alerts pipeline shows throughput in the 500-rows-per-second range (see Figure 8-7).

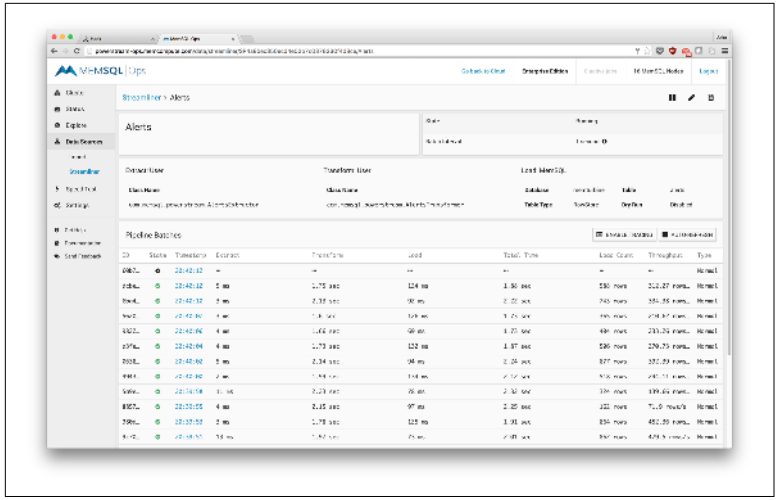

Figure 8-7. The Alerts pipeline showing throughput in the 500-rows-per-second range

The data from Kafka comes by simply identifying the Zookeeper quorum IP address for the Kafka cluster.

Within the Streamliner application, you can create a transform in Scala or Python. If you're using Python, you can edit the transformation within the browser. If you're using Scala, you determine the Class and you can change the JAR file that has the definition of that class.

Implementation is straightforward as Streamliner and Spark apply the machine learning model for each record coming in from Kafka and pushed into the database, under the table name Sensors.

One advantage of storing the pipeline in a database is the ability to store data efficiently. In this pipeline, on duplicate key behavior, data is replaced. More specifically, every data point updates a value for a particular wind turbine and instead of just accumulating this data, we update it.

Advantages of Spark Coupled with a Distributed, Relational, Memory-Optimized Database

These two technologies work well together. For example, MemSQL can focus on speed, storage, and managing state, whereas Spark can focus on transformation. This easy pipeline development comes

through the MemSQL Spark Connector for easy and performant data pipelines.

Inside the transformation stage, SQL commands can be sent to MemSQL or Spark SQL. When going to Spark SQL, you also can choose to push SQL into a SQL-optimized database, essentially delegating parts of the SQL query that query the database to the database itself. Put another way, filters, joins, aggregations, and group-by operations execute directly in the database for maximum performance.

Developers can also join data between MemSQL and data stored elsewhere such as Hadoop, taking advantage of both database functionality and code written for Spark. For real-time applications, developers can focus on SQL in MemSQL for speed and maximum concurrency, and simultaneously run overnight batch operations using Spark SQL across MemSQL Resilient Distributed Datasets (RDDs) and other RDDs without necessarily requiring pushdown. This ensures that batch jobs do not consume real-time resources.

SQL Pushdown Details

Unlike a job-scheduled system like Spark, an in-memory database like MemSQL is continually operating in real time, and requests are processed live. There are no jobs created, and there is no job management. This architecture responds at the millisecond level and delivers great concurrency.

This low latency and high concurrency support for SQL compliments Spark capabilities, and provides a persistent, transactional storage layer. The overarching use case is Spark as a high-level interface, with key functions such as persistence, accelerated query execution, and concurrency pushed down to the in-memory database.

PowerStream at the Command Line

To examine the power of Spark and MemSQL together, we can jump into the Spark shell and take a look at a query and its definition.

In the example that follows, we are selecting a turbine ID with two subselects joining on a complex condition. This is not an equality join but rather a join on distance, whether the status of one row is within a particular distance from another side of the join.

```
scala> query
res11: String =
"
SELECT a.turbine_id
FROM
    (SELECT * FROM turbines_model WHERE windfarm_id < 100) a,
    (SELECT * FROM turbines_model WHERE windfarm_id < 100) b
WHERE
    abs(a.status1_raw - b.status1_raw) < 0.00000000001
"
```

If we look at the explain plan, we see how Spark would run the query. One operator of particular note is `CartesianProduct`, which likely has a lengthy impact on query execution.

```
scala> val dfNoPushdown = mscNoPushdown.sql(query)
dfNoPushdown: org.apache.spark.sql.DataFrame = [turbine_id: int]

scala> dfNoPushdown.explain()
== Physical Plan ==
TungstenProject [turbine_id#16]
 Filter (abs((status1_raw#17 - status1_raw#24)) < 1.0E-11)
  CartesianProduct
   ConvertToSafe
    TungstenProject [turbine_id#16,status1_raw#17]
     Filter (windfarm_id#15L < 100)
      Scan MemSQLTableRelation(MemSQLCluster(MemSQLConf
      (10.0.1.140,3306,root,KPVyNu98Kv8CjRcm,memturbine,
        ErrorIfExists,DatabaseAndTable,10000,GZip)),
          `memturbine`.`turbines_model_out`,
            com.memsql.spark.connector.MemSQLContext@4402630e)
              [windfarm_id#15L,turbine_id#16,status1_raw#17,
                status1#18L,status2_raw#19,
                  status2#20L,status#21L]
   ConvertToSafe
    TungstenProject [status1_raw#24]
     Filter (windfarm_id#22L < 100)
      Scan MemSQLTableRelation(MemSQLCluster(MemSQLConf
      (10.0.1.140,3306,root,KPVyNu98Kv8CjRcm,memturbine,
        ErrorIfExists,
        DatabaseAndTable,10000,GZip)),
          `memturbine`.`turbines_model_out`,
          com.memsql.spark.connector.MemSQLContext@4402630e)
            [windfarm_id#22L,turbine_id#23,
              status1_raw#24,status1#25L,
              status2_raw#26,status2#27L,status#28L]
```

If we enable SQL pushdown, we can see at the top the `MemSQL RDD`, which pushes the query execution directly into the database.

```
scala> val df = msc.sql(query)
df: org.apache.spark.sql.DataFrame = [turbine_id: int]

scala> df.explain()
== Physical Plan ==
MemSQLPhysicalRDD[SELECT (`f_7`) AS `turbine_id` FROM (SELECT
(`query_1`.`f_4`) AS `f_7` FROM
(SELECT (`query_1_1`.`f_1`) AS `f_4`,
(`query_1_1`.`f_2`) AS `f_5`, (`query_2_1`.`f_3`) AS `f_6` FROM
(SELECT (`query_1_2`.`turbine_id`) AS `f_1`,
(`query_1_2`.`status1_raw`) AS `f_2` FROM (SELECT * FROM
(SELECT * FROM `memturbine`.`turbines_model_out`) AS `query_1_3`
  WHERE (`query_1_3`.`windfarm_id` < ?)) AS `query_1_2`)
    AS `query_1_1` INNER JOIN (SELECT
    (`query_2_2`.`status1_raw`) AS `f_3` FROM (SELECT * FROM
    (SELECT * FROM `memturbine`.`turbines_model_out`)
      AS `query_2_3`  WHERE
      (`query_2_3`.`windfarm_id` < ?)) AS `query_2_2`)
        AS `query_2_1` WHERE
        (ABS((`query_1_1`.`f_2` - `query_2_1`.`f_3`)) < ?))
          AS `query_1`) AS `query_0`]
  PartialQuery[query_0, f_7#45] ((`query_1`.`f_4`) AS `f_7`) ()
    JoinQuery[query_1, f_4#45,f_5#46,f_6#53]
    ((`query_1_1`.`f_1`) AS `f_4`, (`query_1_1`.`f_2`) AS `f_5`,
    (`query_2_1`.`f_3`) AS `f_6`) ((ABS
    ((`query_1_1`.`f_2` - `query_2_1`.`f_3`)) < ?))
      PartialQuery[query_1_1, f_1#45,f_2#46]
      ((`query_1_2`.`turbine_id`) AS `f_1`,
      (`query_1_2`.`status1_raw`) AS `f_2`) ()
        PartialQuery[query_1_2, windfarm_id#44L,turbine_id#45,
          status1_raw#46,status1#47L,status2_raw#48,status2#49L,
            status#50L] () ( WHERE
            (`query_1_3`.`windfarm_id` < ?))
          BaseQuery[query_1_3, windfarm_id#44L,turbine_id#45,
            status1_raw#46,status1#47L,status2_raw#48,
              status2#49L,status#50L]
              (SELECT * FROM `memturbine`.`turbines_model_out`)
      PartialQuery[query_2_1, f_3#53]
      ((`query_2_2`.`status1_raw`) AS `f_3`) ()
        PartialQuery[query_2_2, windfarm_id#51L,turbine_id#52,
          status1_raw#53,status1#54L,status2_raw#55,
            status2#56L,status#57L] () ( WHERE
            (`query_2_3`.`windfarm_id` < ?))
          BaseQuery[query_2_3, windfarm_id#51L,turbine_id#52,
            status1_raw#53,status1#54L,
              status2_raw#55,status2#56L,status#57L]
              (SELECT * FROM `memturbine`.`turbines_model_out`)
()

scala> df.count()
res7: Long = 1615
```

```
scala> df.count()
res8: Long = 1427

scala> df.count()
res9: Long = 1385

scala> df.count()
res10: Long = 1565
```

With this type of query, a SQL-optimized database like MemSQL can achieve a response in two to three seconds. With Spark, it can take upward of 30 minutes. This is due to the investment made in a SQL-optimized engine and executing sophisticated queries that utilize the query optimizer and query execution facilities of MemSQL. Databases also make use of indexes, which keep query latency low.

Of course, you also can access MemSQL directly. Here, we are taking the same query from the preceding example and showing the explain plan in MemSQL:

```
EXPLAIN SELECT a.turbine_id
    FROM
        (SELECT
            *
        FROM
            turbines_model_out
        WHERE
            windfarm_id < 100) a,
        (SELECT
            *
        FROM
            turbines_model_out
        WHERE
            windfarm_id < 100) b
    WHERE
        abs(a.status1_raw - b.status1_raw) < 0.00000000001;
+-----------------------------------------------------------------+
| EXPLAIN
+-----------------------------------------------------------------+
| Project [a.turbine_id]
| Filter [ABS(a.status1_raw - b.status1_raw) < .00000000001]
| NestedLoopJoin
| |---TableScan 1tmp AS b storage:list stream:no
| |   TempTable
| |   Gather partitions:all
| |   Project [turbines_model_out_1.status1_raw]
| |   IndexRangeScan memturbine.turbines_model_out AS
  turbines_model_out_1, PRIMARY KEY
  (windfarm_id, turbine_id) scan:[windfarm_id < 100] |
```

```
| TableScan 0tmp AS a storage:list stream:yes
| TempTable
| Gather partitions:all
| Project [turbines_model_out.turbine_id,
  turbines_model_out.status1_raw]
| IndexRangeScan memturbine.turbines_model_out, PRIMARY KEY
  (windfarm_id, turbine_id) scan:[windfarm_id < 100]
+-------------------------------------------------------------+
```

This is a distributed plan that begins with an index scan and a nested loop. Applying the predicate `windfarm_id < 100` enables the database to take advantage of that index.

This query runs in parallel. There is a `Gather` operator to pull the data into the aggregator.

We do a similar operation for other side of join, and within that perform a nested loop join.

For every record on one table, we are looking up that record in another table according to the predicate of the condition of that join.

Using the power of Kafka, Spark, and an in-memory database like MemSQL together, you can go from live data to predictive analytics in a few simple steps. This quickly gets the technology into the business to be more efficient with critical assets.

CHAPTER 9

Techniques for Predictive Analytics in Production

An overarching theme throughout this book has been the accessibility of machine learning. Many powerful, well-understood techniques have been around for decades. What has changed in the past few years are parallel advances in software and hardware that led to the rise of distributed data processing systems.

Real-Time Event Processing

The definition of real time, in terms of specifying a time window, varies dramatically by industry and application. However, a few design principles can improve predictive analytics performance in a wide variety of applications.

Designing a data processing system is a process of deciding when and where computation will happen. In general, all data requires some degree of processing before it can be analyzed. System architects must decide what processing happens at which stage of the data pipeline. At a high level, it is a decision between preprocessing data, which requires more time at the outset but makes data easier to query, versus simply capturing and storing data in its arrival format and doing additional processing at query time.

Structuring Semi-Structured Data

For example, suppose that you are tracking user behavior on an ecommerce website. Most of the information, such as event data, will arrive in a semi-structured format with information like user ID, page ID, and timestamp. In fact, there are probably several different types of events: product page view, product search, customer account login, product purchase, view related product, and so on. A single user session likely consists of tens or hundreds of events. A high-traffic ecommerce website might produce thousands of events per second.

One option is to store each event record in its arrival format. This solution requires essentially no preprocessing and records can go straight to a database (assuming the database supports JSON or another semi-structured data format). Querying the data will likely require performing some aggregation on the fly because every event is its own record.

You could use a key-value or document store, but then you would sacrifice query capabilities like JOINs. Moreover, all modern relational databases offer some mechanism for storing and querying semi-structured data. MemSQL, for example, offers a JSON column type. With a modern database that processes, stores, and queries relational data together with semi-structured data, even in the same query, there are few practical business reasons to use a datastore without familiar relational capabilities.

Suppose that you store each event as its own record in a single column table, where the column is type JSON.

```
CREATE TABLE events ( event JSON NOT NULL );

EXPLAIN events;
+-------+------+------+------+---------+-------+
| Field | Type | Null | Key  | Default | Extra |
+-------+------+------+------+---------+-------+
| event | JSON | NO   |      | NULL    |       |
+-------+------+------+------+---------+-------+
1 row in set (0.00 sec)
```

This approach requires little preprocessing before inserting a record.

```
INSERT INTO
    events ( event )
VALUES
    ( '{ "user_id": 1234, "purchase_price": 12.34 }' );
```

The query to find the sum total of all of one customer's purchases might look like the following:

```
SELECT
    event::user_id user_id,
SUM ( event::$purchase_price )total_spent
FROM
    events
WHERE
    event::$user_id = 1234
```

This approach will work for small datasets, up to tens or hundreds of thousands of records, but even then will add some query latency because you a querying schemaless objects. The database must check every JSON object to make sure it has the proper attributes (`purchase_price` for example) and also compute an aggregate. As you add more event types with different sets of attributes and also data volumes grow, this type of query becomes expensive.

A possible first step is to create computed columns that extract values of JSON attributes. You can specify these computed columns when creating the table, or after creating the table use ALTER TABLE.

```
CREATE TABLE
    events (
          event JSON NOT NULL,
          user_id AS event::$user_id PERSISTED INT,
          price AS event::$purchase_price PERSISTED FLOAT
    );
```

This will extract the values for `user_id` and `purchase_price` when the record is inserted, which eliminates computation at query execution time. You can also add indexes to the computed columns to improve scan performance if desired. Records without `user_id` or `purchase_price` attributes will have NULL values in those computed columns. Depending on the number of event types and their overlap in attributes, it might make sense to normalize the data and divide it into multiple tables. Normalization and denormalization is usually not a strict binary—you need to find a balance between insert latency and query latency. Modern relational databases like MemSQL enable a greater degree of normalization than traditional distributed datastores because of the distributed JOIN execution. Even though concepts like business intelligence and star schemas are commonly associated with offline data warehousing, in some

cases it is possible to report on real-time data using these techniques.

Suppose that you have two types events: purchases and product page views. The tables might look like this:

```
CREATE TABLE purchases (
    event JSON NOT NULL,
    user AS event::$user_id PERSISTED INT,
    session AS event::$session_id PERSISTED INT,
    product AS event::$product_id PERSISTED TEXT,
    price AS event::$purchase_price PERSISTED FLOAT
);

CREATE TABLE views (
    event JSON NOT NULL,
    user AS event::$user_id PERSISTED INT,
    session AS event::$session_id PERSISTED INT,
    product AS event::$product_id PERSISTED TEXT,
    time_on_page AS event::$time_on_page PERSISTED INT
);
```

We assume that views will contain many more records than purchases, given that people don't buy every product they view. This motivates the separation of purchase events from view events because it saves storage space and makes it much easier to scan purchase data.

Now, suppose that you want a consolidated look into both views and purchases for the purpose of training a model to predict the likelihood of purchase. One way to do this is by using a VIEW that joins purchases with views.

```
CREATE VIEW v AS
SELECT
    p.user user,
    p.product product,
    p.price price,
    COUNT(v.user) num_visits,
    SUM(v.time_on_page) total_time
FROM
    purchases p INNER JOIN views v;
```

Now, as new page view and purchase data comes in, that information will immediately be reflected in queries on the view. Although normalization and VIEWs are familiar concepts from data warehousing, it is only recently that they could be applied to real-time problems. With a real-time relational database like MemSQL, you can perform business intelligence-style analytics on changing data.

These capabilities become even more powerful when combined with transactional features `UPDATE`s and "upserts" or `INSERT ... ON DUPLICATE KEY UPDATE ...` commands. This allows you to store real-time statistics, like counts and averages, even for very-high-velocity data.

Real-Time Data Transformations

In addition to structuring data on the fly, there are many tasks traditionally thought of as offline operations that can be incorporated into real-time pipelines. In many cases, performing some transformation on data before applying a machine learning algorithm can make the algorithm run faster, give more accurate results, or both.

Feature Scaling

Many machine learning algorithms assume that the data has been standardized in some way, which generally involves scaling relative to the feature-wise mean and variance. A common and simple approach is to subtract the feature-wise mean from each sample feature, then divide by the feature-wise standard deviation. This kind of scaling helps when one or a few features affect variance significantly more than others and can have too much influence during training. Variance scaling, for example, can dramatically speed up training time for a Stochastic Gradient Descent regression model.

The following shows a variance scaling transformation using a scaling function from the scikit-learn data preprocessing library:

```
>>> from memsql.common import database
>>> from sklearn import preprocessing
>>> import numpy as np
>>> with database.connect(host="127.0.0.1", port=3306,
... user = "root", database = "sample") as conn:
...     a = conn.query("select * from t")
>>> print a
[Row({'a': 0.0, 'c': -1.0, 'b': 1.0}),
Row({'a': 2.0, 'c': 0.0, 'b': 0.0}),
Row({'a': 1.0, 'c': 2.0, 'b': -1.0})]
>>> n = np.asarray(a.rows)
>>> print n
[[ 0.  1. -1.]
 [ 2.  0.  0.]
 [ 1. -1.  2.]]
>>> n_scaled = preprocessing.scale(n)
>>> print n_scaled
```

```
[[-1.22474487  1.22474487 -1.06904497]
 [ 1.22474487  0.         -0.26726124]
 [ 0.         -1.22474487  1.33630621]]
```

This approach finds a scaled representation for a particular set of feature vectors. It also uses the feature-wise means and standard deviations to create a generalized transformation into the variance-standardized space.

```
>>> n_scaler = preprocessing.StandardScaler().fit(n)
>>> print n_scaler.mean_
[ 1.          0.          0.33333333]
>>> print n_scaler.scale_
[ 0.81649658  0.81649658  1.24721913]
```

With this information, you can express the generalized transformation as a view in the database.

```
CREATE VIEW scaled AS
SELECT
    (t.a - 1.0) / .8164 scaled_a,
    (t.b - 0.0) / .8164 scaled_b,
    (t.c - 0.33) / 1.247 scaled_c
FROM
    my_table t
```

Now, you interactively query or train a model using the scaled view. Any new records inserted into `my_table` will immediately show up in their scaled form in the scaled view.

Real-Time Decision Making

When you optimize real-time data pipelines for fast training, you open new opportunities to apply predictive analytics to business problems. Modern data processing techniques confound the terminology we traditionally use to talk about analytics. The "online" in Online Analytical Processing (OLAP) refers to the experience of an analyst or data scientist using software interactively. "Online" in machine learning refers to a class of algorithms for which the model can be updated iteratively, as new records become available, without complete retraining that needs to process the full dataset again.

With an optimized data pipeline, there is another category of application that uses models that are "offline" in the machine learning sense but also don't fit into the traditional interaction-oriented definition of OLAP. These applications fully retrain a model using the most up to date data but do so in a narrow time window. When data

is changing, a predictive model trained in the past might not reflect current trends. The frequency of retraining depends on how long a newly trained model remains accurate. This interval will vary dramatically across applications.

Suppose that you want to predict recent trends in financial market data, and you want to build an application that alerts you when a security is dramatically increasing or decreasing in value. You might even want to build an application that automatically executes trades based on trend information.

We'll use the following example schema:

```
CREATE TABLE `ask_quotes` (
    `ticker` char(4) NOT NULL,
    `ts` BIGINT UNSIGNED NOT NULL,
    `ask_price` MEDIUMINT(8) UNSIGNED NOT NULL,
    `ask_size` SMALLINT(5) UNSIGNED NOT NULL,
    `exchange` ENUM('NYS','LON','NASDAQ','TYO','FRA') NOT NULL,
    KEY `ticker` (`ticker`,`ts`),
);
```

In a real market scenario, new offers to sell securities ("asks") stream in constantly. With a real-time database, you are able to not only record and process asks, but serve data for analysis simultaneously. The following is a very simple Python program that detects trends in market data. It continuously polls the database, selecting all recent ask offers within one standard deviation of the mean price for that interval. With recent sample data, it trains a linear regression model, which returns a slope (the "trend" you are looking for) and some additional information about variance and how much confidence you should have in your model.

```
#!/usr/bin/env python

from scipy import stats
from memsql.common import connection_pool

pool = connection_pool.ConnectionPool()
db_args = [<HOST>, <PORT>, <USER>, <PASSWORD>, <DB_NAME>]

# ticker for security whose price you are modeling
TICKER = <ticker>

while True:
    with pool.connect(*db_args) as c:
        a = c.query('''
            SELECT ask_price, ts
```

```
        FROM (
            SELECT *
            FROM ask_quotes
            ORDER BY ts DESC LIMIT 10000) window
        JOIN (
            SELECT AVG(ask_price) avg_ask
            FROM ask_quotes WHERE ticker = "{0}") avg
        JOIN (
                SELECT STD(ask_price) std_ask
                FROM ask_quotes
                WHERE ticker = "{0}") std
        WHERE ticker="{0}"
            AND abs(ask_price-avg.avg_ask) < (std.std_ask);
    '''.format(TICKER))
    x = [a[i]['ts'] for i in range(len(a) - 1)]
    y = [a[i]['ask_price'] for i in range(len(a) - 1)]

    slope, int, r_val, p_val, err = stats.linregress(x, y)
```

With the information from the linear regression, you can build a wide array of applications. For instance, you could make the program send a notification when the slope of the regression line crosses certain positive or negative thresholds. A more sophisticated application might autonomously execute trades using market trend information. In the latter case, you almost certainly need to use a more complex prediction technique than linear regression. Selecting the proper technique requires balancing the need for low training latency versus the difficulty of the prediction problem and the complexity of the solution.

CHAPTER 10

From Machine Learning to Artificial Intelligence

Statistics at the Start

Machine-learning methods have changed rapidly in the past several years, but a larger trend began about a decade ago. Specifically, the field of data science emerged and we experienced an evolution from statisticians to computer engineers and algorithms (see Figure 10-1).

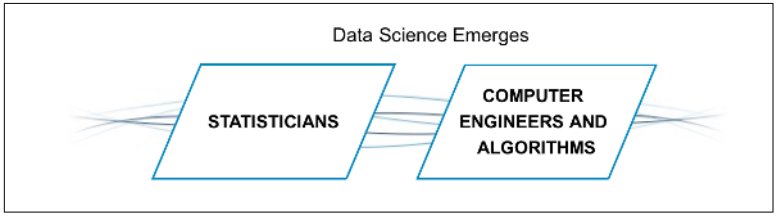

Figure 10-1. The evolution from statisticians to computer engineers and algorithms

Classical statistics was the domain of mathematics and normal distributions. Modern data science is infinitely flexible on the method or properties, as long as it uncovers a predictable outcome. The classical approach involved a unique way to solve a problem. But new approaches vary drastically, with multiple solution paths.

To set context, let's review a standard analytics and split a dataset into two parts, one for building the model, and one for testing it,

aiming for a model without overfitting the data. Overfitting can occur when assumptions from the build set do not apply in general.

For example, as a paint company seeking homeowners that might be getting ready to repaint their houses, the test set may indicate the following:

Name	Painted house within 12 months
Sam	Yes
Ian	No

Understandably, you cannot generalize on this property. But you could look at income pattern, data regarding the house purchase, and recently filed renovation permits to create a far more generalizable model. This kernel of an approach spawned the transition of an industry from statistics to machine learning.

The "Sample Data" Explosion

Just one generation ago, data was extremely expensive. There could be cases in which 100 data points was the basis of a statistical model. Today, at web-scale properties like Facebook and Google, there are hundreds of millions to billions of records captured daily.

At the same time, compute resources continue to increase in power and decline in cost. Coupled with the advent of distributed computing and cloud deployments, the resources supporting a computer driven approach became plentiful.

The statisticians will say that new approaches are not perfect, but for that matter, statistics are not, either. But what sets machine learning apart is the ability to invest and discover algorithms to cluster observations, and to do so iteratively.

An Iterative Machine Process

Where machine learning stepped ahead of the statistics pack was this ability to generate iterative tests. Examples include Random Forest, an approach that uses rules to create an ensemble of decision trees and test various branches. Random Forest is one way to reduce overfitting to the training set that is common with simpler decision tree methods.

Modern algorithms in general use more sophisticated techniques than Ordinary Least Squares (OLS) regression models.

Keep in mind that regression has a mathematical solution. You can put it into a matrix and compute the result. This is often referred to as a *closed-form approach*.

The matrix algebra is typically $(X'X)^{-1}X'Y$, which leads to a declarative set of steps to derive a fixed result. Here it is in more simple terms:

If $X + 4 = 7$, what is X?

You can solve this type of problem in a prescribed step and you do not need to try over and over again. At the same time, for far more complex data patterns, you can begin to see how an iterative approach can benefit.

Digging into Deep Learning

Deep learning takes machine learning one step further by applying the idea of neural networks. Here we are also experiencing an iterative game, but one that takes calculations and combinations as far as they can go.

The progression from machine learning to deep learning centers on two axes:

- Far more complex transfer functions, and many of them, happening at the same time. For example, take the sng(x) to the 10th power, compare the result, and then recalibrate.
- Functions in combinations and in layers. As you seek parameters that get you closest to the desired result, you can nest functions. The ability to introduce complexity is enormous. But life and data about life is inherently complex, and the more you can model, the better chance you have to drive positive results.

For example, a distribution for a certain disease might be frequency at a very young or very old age, as depicted in Figure 10-2.

Classical statistics struggled with this type of problem-solving because the root of statistical science was based heavily in normal distributions such as the example shown in Figure 10-3.

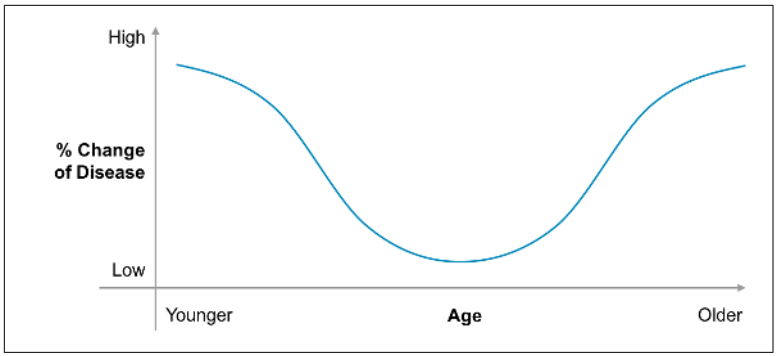

Figure 10-2. Sample distribution of disease prevalent at young and old age

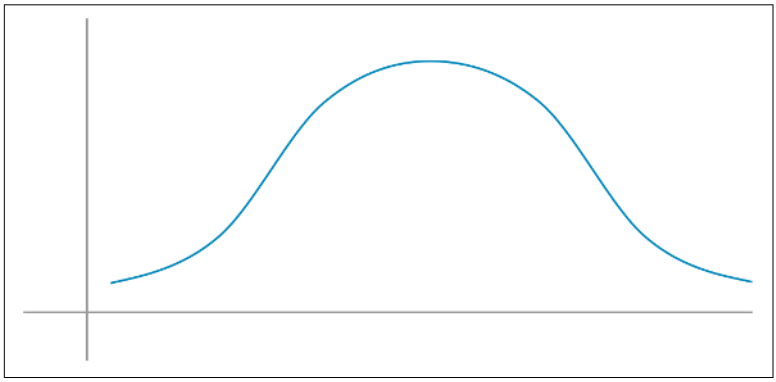

Figure 10-3. Sample normal distribution

Iterative machine learning models do far better at solving for a variety of distributions as well as handling the volume of data and the available computing capacity.

For years, machine learning methods were not possible due to excessive computing costs. This was exacerbated by the fact that analytics is an iterative exercise in and of itself, and the time and computing resources to pursue machine learning made it unreasonable, and closed form approaches reigned.

Resource Management for Deep Learning

Though compute resources are more plentiful today, they are not yet unlimited. So, models still need to be implementable to sustain and support production data workflows. The benefit of a fixed-type or

closed-loop regression is that you can quickly calculate the compute time and resources needed to solve it.

This could extend to some nonlinear models, but with a specific approach to solving them mathematically. LOGIT and PROBIT models, often used for applications like credit scoring, are one example of models that return a rank between 0 and 1 and operate in a closed-loop regression.

With machine and deep learning, computer resources are far more uncertain. Deep learning models can create thousands of lines of code to execute, which, without a powerful datastore, can be complex and time consuming to implement. Credit scoring models, on the other hand, can often be solved with 10 lines of queries shareable within an email.

So, resource management and the ability to implement models in production remains a critical step for broad adoption of deep learning. Take the following example:

- Nested JSON objects coming from S3 into a queryable datastore
- 30–50 billion observations per month
- 300–500 million users
- Query user profiles
- Identify people who fit a set of criteria
- Or, people who are near this retail store

Although a workload like this can certainly be built with some exploratory tools like Hadoop and Spark, it is less clear that this is an ongoing sustainable configuration for production deployments with required SLAs. A datastore that uses a declarative language like SQL might be better suited to meeting operational requirements.

Talent Evolution and Language Resurgence

The mix of computer engineering and algorithms favored those fluent in these trends as well as statistical methods. These data scientists program algorithms at scale, and deal with raw data in large volumes, such as data ending up in Hadoop.

This last skill is not always common among statisticians and is one of the reasons driving the popularity of SQL as a programming layer for data. Deep learning is new, and most companies will have to

bridge this gap from classical approaches. This is just one of the reasons why SQL has experienced such a resurgence as it brings a well-known approach to solving data challenges.

The Move to Artificial Intelligence

The move from machine learning to broader artificial intelligence will happen. We are already seeing the accessibility with open source machine learning libraries and widespread sharing of models.

But although computers are able to tokenize sentences, semantic meaning is not quite there. Alexa, Amazon's popular voice assistant, is looking up keywords to help you find what you seek. It does not grasp the meaning, but the machine can easily recognize directional keywords like weather, news, or music to help you.

Today, the results in Google are largely based on keywords. It is not as if the Google search engine understands exactly what we were trying to do, but it gets better all the time.

So, no Turing test yet—we speak of the well-regarded criteria to indicate that a human cannot differentiate from a human or a computer when posing a set of questions.

Therefore, complex problems are still not likely solvable in the near future, as common sense and human intuition are difficult to replicate. But our analytics and systems are continuously improving opening several opportunities.

The Intelligent Chatbot

With the power of machine learning, we are likely to see rapid innovation with intelligent chatbots in customer service industries. For example, when customer service agents are cutting and pasting scripts into chat windows, how far is that from AI? As voice recognition improves, the days of "Press 1 for X and 2 for Y" are not likely to last long.

For example, chat is popular within the auto industry as a frequent question is, "is this car on the lot?"

Wouldn't it be wonderful to receive an instant response to such questions instead of waiting on hold?

Similarly, industry watchers anticipate that more complex tasks like trip planning and personal assistants are ready for machine-driven advancements.

Broader Artificial Intelligence Functions

The path to richer artificial intelligence includes a set of capabilities broken into the following categories:

- Reasoning and logical deductions to help solve puzzles
- Knowledge about the world to provide context
- Planning and setting goals to measure actions and results
- Learning and automatic improvement to refine accuracy
- Natural-language processing to communicate
- Perception from sensor inputs to experience
- Motion and robotics, social intelligence, creativity to get closer to simulating intelligence

Each of these categories has spawned companies and often industries, for example natural language processing has become a contest of legacy titans such as Nuance along with newer entrants like Google (Google Now), Apple (Siri), and Microsoft (Cortana). Sensors and the growth of the Internet of Things has set off a race to connect every device possible. And robotics is quickly working its way into more areas of our lives, from the automatic vacuum cleaner to autonomous vehicles.

The Long Road Ahead

For all of the advancements, there are still long roads ahead. Why is it that we celebrate click through rates online of just 1 percent? In the financial markets, why is it that we can't get it consistently right? Getting philosophical for a moment, why do we have so much uncertainty in the world?

The answers might still be unknown, but more advanced techniques to get there are becoming familiar. And, if used appropriately, we might find ourselves one step closer to finding those answers.

APPENDIX A
Appendix

Sample code that generates data, runs a linear regression, and plots the results:

```
import numpy as np
import matplotlib.pyplot as plt
from scipy import stats

x = np.arange(1,15)

delta = np.random.uniform(-2,2, size=(14,))

y = .9 * x + 1 +  delta

plt.scatter(x,y, s=50)

slope, int, r_val, p_val, err = stats.linregress(x, y)

plt.plot(x, slope * x + intercept)
plt.xlim(0)
plt.ylim(0)

# calling show() will open your plot in a window
# you can save rather than opening the plot using savefig()
plt.show()
```

Sample code that generates data, runs a clustering algorithm, and plots the results:

```
import numpy as np
import matplotlib.pyplot as plt
from scipy import stats
from scipy.cluster.vq import vq, kmeans
```

```
data = np.vstack((np.random.rand(200,2) + \
       np.array([.5, .5]),np.random.rand(200,2)))

centroids2, _ = kmeans(data, 2)
idx2,_ = vq(data,centroids2)

# scatter plot without centroids
plt.figure(1)

plt.plot(data[:,0],data[:,1], 'o')

# scatter plot with 2 centroids
plt.figure(2)

plt.plot(data[:,0],data[:,1],'o')
plt.plot(centroids2[:,0],centroids2[:,1],'sm',markersize=16)

# scatter plot with 2 centroids and point colored by cluster
plt.figure(3)

plt.plot(data[idx2==0,0],data[idx2==0,1],'ob',data[idx2==1,0], \
         data[idx2==1,1],'or')
plt.plot(centroids2[:,0],centroids2[:,1],'sm',markersize=16)

centroids3, _ = kmeans(data, 3)
idx3,_ = vq(data,centroids3)

# scatter plot with 3 centroids and points colored by cluster
plt.figure(4)

plt.plot(data[idx3==0,0],data[idx3==0,1],'ob',data[idx3==1,0], \
         data[idx3==1,1],'or',data[idx3==2,0], \
         data[idx3==2,1],'og')
plt.plot(centroids3[:,0],centroids3[:,1],'sm',markersize=16)

# calling show() will open your plots in windows, each opening
# when you close the previous one
# you can save rather than opening the plots using savefig()
plt.show()
```

About the Authors

Conor Doherty is a technical marketing engineer at MemSQL, responsible for creating content around database innovation, analytics, and distributed systems. He also sits on the product management team, working closely on the Spark-MemSQL Connector. While Conor is most comfortable working on the command line, he occasionally takes time to write blog posts (and books) about databases and data processing.

Steven Camiña is a principal product manager at MemSQL. His experience spans B2B enterprise solutions, including databases and middleware platforms. He is a veteran in the in-memory space, having worked on the Oracle TimesTen database. He likes to engineer compelling products that are user-friendly and drive business value.

Kevin White is the Director of Marketing and a content contributor at MemSQL. He has worked in the digital marketing industry for more than 10 years, with deep expertise in the Software-as-a-Service (SaaS) arena. Kevin is passionate about customer experience and growth with an emphasis on data-driven decision making.

Gary Orenstein is the Chief Marketing Officer at MemSQL and leads marketing strategy, product management, communications, and customer engagement. Prior to MemSQL, Gary was the Chief Marketing Officer at Fusion-io, and he also served as Senior Vice President of Products during the company's expansion to multiple product lines. Prior to Fusion-io, Gary worked at infrastructure companies across file systems, caching, and high-speed networking.