Using a predictive analytics model to foresee flight delays

Get started
Introduction

Section 1:
How to predict flight delays using Apache\textsuperscript{®} Spark\textsuperscript{™} MLlib

Section 2:
How building strong analytical models drives better business decisions

Section 3:
Gain business advantage by analyzing data and using predictive models

Appendix
This paper describes how data scientists and developers can build an application to predict flight delays using a Get-Build-Analyze methodology and IBM® Analytics for Apache Spark, a managed Apache Spark service, with interactive Jupyter Notebooks. The application uses machine learning (Spark MLlib), fueled by publicly available airplane flight data and enriched with weather data, to predict flight delays due to weather conditions. In the event that a delay is likely, the app can also help determine the degree to which the flight will be delayed.

This paper is divided into three sections:

- Section 1 focuses on collecting data from various sources.
- Section 2 explains the procedure to build the data and test a predictive model.
- Section 3 covers analyzing data for business advantage, and quantifying the impact of weather on flight delays.
Introduction

Data science is more of an interactive art form than an exact science, and requires the data scientist to continually try new things and fail fast. Data scientists and developers can gather data from a variety of sources and move it to the cloud, build a single data platform and predictive model, and then analyze both the data and the model to continuously improve and ultimately deliver business value faster.

Whether you are a data scientist or a data engineer, you can directly apply the practices and patterns in this paper to your use cases, and leverage the ecosystem of data services available in the IBM Bluemix\textsuperscript{\textregistered} platform.
Section 1: How to predict flight delays using Apache Spark MLlib

If you’ve ever found yourself stranded at an airport, on a tarmac or in a terminal, you’re not alone: delays cost airline passengers about 115 million minutes of travel time annually. But what if there was a way for travelers to use data around one of the main causes for these delays—weather conditions—to predict whether their flight would be delayed?

This section of the paper uses a flight predictor application to show data scientists and developers how to gather data from a variety of sources and get it into the cloud, build a single data platform, perform critical predictive analysis, and deliver business value faster, using next-generation analytics tools.

Disclaimer: All steps in the Get-Build-Analyze methodology used throughout this document are very iterative and can frequently overlap.

This image illustrates the tasks and products used in each of the three phases of the Get-Build-Analyze methodology.
Get data: Putting the pieces in place

In the example below, we’ll look at how a fictional data scientist named Russell might approach the Get Data phase of the Get Build Analyze methodology when building this flight prediction application.

1. **Identify the problem:** Airplane travel is complicated with inconsistent flight departure and arrival times, particularly when weather conditions aren’t ideal. In fact, according to the Bureau of Transportation Statistics, nearly one in five domestic flights is delayed. These delays are costly for airlines, and frustrating for travelers. These travelers need a way to better predict which flights will be late, and how late they’ll be, in order to make more accurate travel plans.

2. **Describe the expected outcome:** Given flight departure and arrival data and weather observations, the application can predict flight delays using five training classifications:
   - 0 = Canceled
   - 1 = On time
   - 2 = Delayed less than 2 hours
   - 3 = Delayed between 2 and 4 hours
   - 4 = Delayed more than 4 hours

3. **Identify the data sources to be used:** Identify and prepare the raw data to be used in the training set. This is a critical step, and one that goes a long way toward determining the predictive accuracy of a model. Small data samples, while easier to analyze, tend to produce less accurate patterns. Large data samples often create more accurate patterns, but analyzing huge volumes of data complicates the process of identifying relevant patterns.
Get data: Putting the pieces in place (cont.)

In this scenario, Russell is interested in performing predictive analysis on the main causes for flight delays. He starts by collecting raw data during a 30-day period from FlightStats.com, including origin and departure times of flights, as well as differences between expected and actual arrival times. He also collected corresponding weather data at five airports—Boston, San Francisco, Austin, Miami and Orlando—using the IBM Insights for Weather service on Bluemix:

- Past weather observations had more detail than forecast data, so the number of features used to train the models was limited to the intersections of the two.
- FlightStats data is only accessible for a 30-day free trial period and limited to 20,000 API calls, which impacted the sample size of the data.
- Training data was restricted to weather forecasts only at departure and arrival airports, and doesn’t take into account weather conditions at various points along the flight route.

While identifying data sources, Russell encountered several constraints that would affect the project:

- The Insights for Weather service provides past observations only as far as 24 hours back, so this data had to be collected once every 24 hours.
Staging the data for analysis

Typically, data scientists want all their data in one place (a warehouse), where it can be visualized using a dashboard. This is known as an operational data store. It's a simple goal, but the mechanics of moving data into an ODS are often complex, and may involve extra steps like data cleansing and transformation.

For the purposes of our flight predictor demo application, Russell first moved the raw data from the two operational data sources—flight information from FlightStats and weather information from Insights for Weather—into an operational data store using an IBM Simple Data Pipe and custom data connectors.

Disclaimer: All steps in the Get-Build-Analyze methodology used throughout this document are very iterative and can frequently overlap.
Simple data pipe and custom connectors

IBM’s Simple Data Pipe is an app for moving information into IBM dashDB™, IBM’s cloud data warehouse service. The app comes with built-in connectors for web data sources like Salesforce.com and Stripe, and IBM provides frameworks for Russell to build his own. (It’s all open-source on GitHub, and the repo includes example visualizations to help him get started.)

He used this framework to build a custom connector to collect his raw data from flightstats.com and combine it with weather observations from IBM Insights for Weather on Bluemix. Here is a screenshot of the running data connector Russell used for this application:

Because the weather service doesn’t return weather observations older than 24 hours (as discussed earlier in the data constraints), Russell had to gather the weather data set every 24 hours. Here is a sample of the data connector he built to connect to the IBM Insights for Weather service on Bluemix:
Most web apps like flightdata.com exchange data in a JSON (JavaScript Object Notation) format, because it allows them to store information in an organized, easy-to-understand and easy-to-access manner. Russell built a Simple Data Pipe connector to move the JSON data gathered via the connectors into the IBM Cloudant® NoSQL DB service to create the data sets he needed and to store the data. Russell chose Cloudant as his store because it's fast, reliable and easy to use (in large part because it's based on the CouchDB open source project). Most importantly, Cloudant is distributed in nature, which enables the flight predictor app to scale larger and remain available to users wherever they are.
Using a predictive analytics model to foresee flight delays

Section 1 > Get data > Staging data > Pipes and custom connectors > Come clean

Time to come clean

The final step in the Get Data phase—processing and preparing the data that was moved to the cloud—actually isn’t confined to this stage. All of these steps—processing and cleansing, ensuring integrity, identifying and acquiring missing points, making corrections—are iterative. These adjustments can be made throughout the life of a project, as needed, and some data scientists may even wait until later in the process to start making them.

In this case, to address the problem of missing or non-numerical values in the data set, Russell had two choices: he could cleanse the data at the source in the Simple Data Pipe stage, or he could wait.

Russell chose to leave the erroneous data in the source data and create an “if-then-else” rule that looked at data values; if the data value was missing or not numerical, he forced the value to change to 0.0.

Again, every case is different. Data scientists need flexibility when deciding how to process data. In this example, some may not want to use a value of 0.0, or they may determine that if data is missing, the data value or data source should be removed altogether. That’s up to the data scientist.

Once finished with all these checks on the flight and weather data, Russell is finally ready to build his model and actually conduct the analysis. The data is ready to be plugged into the IBM Analytics for Apache Spark, a managed Apache Spark service, where he’ll work with the data in a Jupyter Notebook.
Using a predictive analytics model to foresee flight delays

**Section 2: How building strong analytical models drives better business decisions**

Data scientists use a mix of science and creativity, and they are continually testing their hypotheses. While there are best practice guidelines, there’s no exact recipe. There are many variables that affect findings, and the key is to ascertain which features (data fields) have the biggest impact and create the most accurate, precise and repeatable predictive models as quickly as possible. Jupyter Notebook is one tool that makes it easier for data scientists to perform the art of data science.

In this section, Russell will show how you can quickly and easily build a single data platform, then build and test analytical models using the IBM Analytics for Apache Spark, a managed Apache Spark service, with interactive Jupyter Notebooks and his flight predictor demo application.

*Disclaimer: All steps in the Get-Build-Analyze methodology used throughout this document are very iterative and can frequently overlap.*
Section 2: How building strong analytical models drives better business decisions (cont.)

As explained in the first section of the document, Russell’s data is now prepared for analysis, and he is ready to progress to the Build Data phase of the Get-Build-Analyze methodology.

IBM Analytics for Apache Spark with interactive Jupyter Notebooks

For the purposes of the demo application, think of IBM Analytics for Apache Spark as the nervous system that data scientists use in conjunction with other analytics tools to acquire, understand and analyze data. IBM Analytics for Apache Spark unifies data processing and analytics, regardless of data type, location, or source, onto a single platform where end-to-end analysis is possible. This is why Apache Spark is sometimes called “the analytics operating system.”

This flexibility, also referred to as extensibility, allows data scientists to perform powerful analytics that would have been impossible previously. In addition, it all happens up to 100 times faster than before, thanks to Spark’s open source, in-memory, distributed computing framework.

As an extension of IBM Analytics for Apache Spark, a managed Apache Spark service, Jupyter Notebooks also provide interactive, iterative, and collaborative work environments for programming and analytics. Jupyter Notebooks are very easy to use for both technical and line-of-business (LOB) users. They provide the graphical interface that enables data scientists to leverage and share advanced analytics more easily.
Build data

Building the data involves importing already-processed data into tools where analytical models can be created and tested.

Note: To make the code shorter, more readable and more reusable, Russell created a Python package that encapsulates a few APIs that will do the heavy lifting, like loading data from Cloudant into Spark DataFrames and running performance metrics. The goal is to enable him to focus on his work, and not on other housekeeping tasks.
Prepare

1. **Combine data:** Remember the flight and weather data Russell processed earlier? At that point, he landed data in an operational data store so it could be analyzed. That prepared data is now ready to be combined and imported into a Notebook. As previously noted, some data scientists may choose to cleanse the data at the source. Since he didn’t do that, Russell would have to cleanse the data now.

2. **Enrich data:** Once data is joined and deduplicated, it can be enriched. Russell took the prepared training data collected from FlightStats.com and enriched it with weather observations from the IBM Insights for Weather service on Bluemix.

3. **Apply business rules:** Finally, using his enriched data, Russell created business rules to better understand relationships between weather observations and flight times. For example, if the delta between original and actual departure times was greater than 0 but less than 120 minutes, he assigned a value of 2 to the classification data field.
Explore

The next step of the Build Data phase is to explore the data. This iterative process involves visualizing the data, creating a hypothesis, and then building and testing analytical models based on learnings. As Russell begins exploring his data, he doesn’t initially know what he’s looking for. He can make educated guesses, but ultimately, he wants to explore a wide variety of relationships and determine what works and what doesn’t as quickly as possible.

1. **Load**: Russell used the Cloudant-Spark connector to load the data sets from Cloudant into a Spark DataFrame.

2. **Visualize**: He began exploring the data by selecting three data points (temperature, pressure and wind speed) and visualizing the data as clusters in the Notebook document. He also chose to look at four common classification algorithms: Naive Bayes, decision tree, logistic regression, and random forest.

Russell first took a look at some of the data visualized in scatter plots. His first scatter plot explores flight delays based on temperature in departure and arrival airports. The distribution is good, but the patterns? Not really.
It was a similar story when he visualized flights delays based on wind speed in departure and arrival airports. Again, there is good distribution of the data, but no clear pattern.

The third chart showed flight delays based on air pressure in departure and arrival airports. The chart shows tight clustering around specific pressure levels with a few outlying data points.

3. **Hypothesize**: The next step was to hypothesize predictive patterns based on what Russell observed in his visualizations. For the demo application, he is testing the hypothesis that there is high correlation between weather conditions and flight delays, and further, that temperature, pressure and wind speed have the greatest effect on time delays.

4. **Build analytical models**: Next, Russell built analytical models based on the results of the data visualization and hypothesis generation.
Test

In data science, building a model is not the end of the road. The model needs to be tested repeatedly to ensure maximum accuracy.

1. **Run diagnostic tests**: Russell’s first step in testing his models was to reload new data and run diagnostic tests. For strong validation, he made sure to use blind data sets and not the training data sets that were used to create the models.

2. **Evaluate performance**: Using the findings from the diagnostic tests, Russell then evaluated the performance of each analytical model. He asked himself a few questions: Does the new data visualize in the same way as his training data? Does this make sense? What’s different and why? For the purposes of the demo application, he changed the number of trees from five to 250 and saw that his random forest accuracy went down from 57 percent to 55 percent.

Based on this, he deduced that 250 is too many trees. Then he tried using 100 trees, and did not see much difference. He therefore deduced that the number of trees does not have a significant impact, and began changing other variables.

There are many techniques to follow when attempting to improve analytical models. One method is to obtain additional training data. Russell can also select different features (data points) by calculating the correlation between specific features and results to improve predictive capability. Some data scientists will even go back to their training data to scale and normalize it using statistical smoothing functionality.

3. **Compare metrics**: Russell compared the accuracy, precision and recall achieved by his four models, and this is what he found:

   
   ![Table showing model performance metrics](image)

   
   From talking to other data scientists about the flight predictor project, Russell can tell that these are actually encouraging results—but he’d still like to improve them.
Best practices for data analysis

To improve the accuracy of his model, Russell will acquire more data from FlightStats.com. He will also pull data from more airports outside of the five he’s currently analyzing. His findings are in the third section of this document, where we will go into more detail about analyzing data for business advantage, and provide ideas for improving the accuracy of the models.
Section 3: Gain business advantage by analyzing data and using predictive models

This section explains how data scientists can quickly and easily use the Get-Build-Analyze methodology—along with IBM Analytics for Apache Spark service and its integrated, interactive Jupyter Notebook—to develop and improve predictive models. The first section of this document explained how Russell obtained and prepared his data for the flight predictor demo application, while the second section demonstrated how he built and tested a single data platform and predictive model. This third section will discuss how he has improved the accuracy of his flight predictor model.
Analyzing the data involves communicating key findings, making new discoveries and implementing the predictive model.

Disclaimer: All steps in the Get-Build-Analyze methodology used throughout this document are very iterative and can frequently overlap.
Communicate

Russell knows that the first step in the Analyze Data phase is to communicate the business value of his model to management and customers. He makes sure to gather specific data on the key performance indicators of his model, and stores this data in a data warehouse so that he can be simple and clear in how he communicates key trends and findings, and to make sure that he promotes the return on investment from his project. He’ll use prescriptive analytics to recommend actions and show likely outcomes, and he’ll also include mechanisms for notifying customers and management about situations they care about (such as how many flights are delayed per week). It is important to also include alerting capabilities, so that corrective action can be taken should specific types of data go above or below predetermined thresholds.

For purposes of the flight delay predictor demo, Russell can gather performance data such as how many flights were delayed during a specific time period, how many flight delays were correctly predicted and which airports have the most flight delays. From this, he could define success metrics such as the number of hours people saved by rescheduling flights to avoid delays. With some basic assumptions (for example, $50 in cost savings per hour per person), Russell can estimate how much money a company could potentially save by using this application. He could also alert management that flights from a specific city are typically delayed, and suggest new travel policies be created to route employees through other cities to avoid those delays.

Russell considers using business intelligence and reporting tools such as IBM Watson™ Analytics, Looker, or Tableau to create compelling graphics that visualize how his project is going, and communicate these results through reports, a custom front-end dashboard and other data visualization tools. Depending on the type of project, the business value might be measured in the number of new customers obtained, the number of existing customers retained, or the amount of incremental revenue or cost savings realized. He makes sure that he shows what the data means in a way that impacts his audience. This is his opportunity to prove his project’s business value to his management team and customers.

An important takeaway is that the iterative nature of the Get-Build-Analyze methodology enables data scientists to quickly analyze situations and try new things. If the changes are not working, data scientists can quickly try something else and see if it works better. If the changes are working, data scientists can continue going in that direction or extend this success with additional changes. In this way, data scientists can discover and communicate high-value insights faster than ever before.
Discover

For Russell, the next step of the Analyze Data phase is to find new insights about his data and better measure the performance of the predictive model. This iterative step typically involves creating interactive KPI dashboards, developing insights based on newly revealed patterns, and identifying new opportunities. Here, he will try to better understand how his models behave for each of the classes, and then use this information to further refine the training data.

Metrics tables, confusion matrices and histograms

Three tools that can help analyze a model's accuracy and discover new information are metrics tables, confusion matrices, and histograms. Russell first looked at a metrics table during the Test step in the Build Data phase to assess the precision, accuracy and recall metrics for his flight predictor model. Metrics tables do a good job of indicating the overall quality of a prediction, but they only provide part of the story. They do not give detailed performance information about each class or indicate which class is doing better. That's where a confusion matrix and histogram can help, because they allow a deeper dive into the numbers.

IBM Analytics for Apache Spark has a library (MLlib) of APIs that provide many types of functionality that help data scientists explore data and improve understanding. As part of the MLlib native library, Russell was able to generate a confusion matrix with a single API call, and see which classes were underperforming. This capability was helpful in troubleshooting his flight predictor model. For example, he may see that a particular class (such as flights delayed more than four hours) may have low accuracy.
Discover (cont.)

In looking at the confusion matrix, Russell can see that Class A and Class B are performing well, with 415 and 1494 correct predictions, respectively. He can also see that Class C is performing poorly, with only 90 correct predictions. Based on this, he must now determine why this class has been confused. Is his training data missing label points in this class? Does he have enough training data for each class (normal distribution)? Does he need more predictive features (explanatory variables)? Are there features that are counterproductive?

To determine if he has enough training data for each class, Russell created a histogram and specified 40 bins to better understand the distribution of records from delta Departure:
Discover (cont.)

Using the MLlib native library, Russell was able to see the distribution of his training data into bins, which told him how much data was in each bin. He was looking for a normal distribution, with the same amount of data in each bin. With one quick command, he was able to see that his data does not have a normal distribution across the bins. Having a few bins with a lot of data is normally not good for machine training purposes. He needed more than a handful of data outliers. With IBM Analytics for Apache Spark, he didn’t have to extract his training data into separate tools and spend a lot of time analyzing the data to figure this out. He was able to use a single tool to iterate and share data with other scientists quickly, without writing too much code.

Combined, a confusion matrix and histogram showed Russell how his training data was distributed across classes. This helped identify problems with his model, and let him iteratively go back and adjust things. To fix the distribution, Russell could obtain more training data in the required bins. Or, he could reclassify to better adapt his flight predictor model to the existing training data. (However, if he does this, he should make sure that any new classifications still match his customers’ needs.)

To improve his model, he chose to add new features and refine how classes were segmented as part of an iterative process, and then remeasure the improved model. He chose to use only three classes, not five, and he also added more features to see how the model reacted. This was very simple, as all he did was write a few lines of code into a cell and rerun what already existed. By using APIs from his Python package, he was able to change the configuration and aspects of his model quickly and easily.
Discover (cont.)

For Russell, the important thing is to never lose sight of his customers and to stay in touch with what they want. Do customers even care if a flight is more than two hours delayed? The key thing is to go back and adjust the training data and then rebuild his model. It’s an iterative loop that is never really done, because it’s all about continuous improvement. It’s important that he always be in touch with customers to make sure he knows how to improve the model in ways that add value to them. In this way, analytics are always helping his customers and his business.
Implement

The final part in the Get-Build-Analyze model is to implement the predictive model in a way that provides value to customers and companies alike. This includes planning, making recommendations, and measuring the return on investment (ROI) of the model. The key thing is understanding how to integrate predictive models into systems of engagement.

For the purposes of his flight predictor application, Russell used a Google card plug in. If the app detects a high probability of delay (60% or higher), it sends an alert to the customer. Consider a customer that’s getting ready for a flight from London to New York: the customer receives a notification that his flight will likely be delayed. He is then redirected to a third-party service, such as Freebird.com, and given the opportunity to change to a different flight. This also enables him to change other plans, such as dinner reservations.

The flight predictor model provides business value because it enables a company to partner with new third parties, and to add revenue streams by charging for new, high-value features. It adds value for customers by saving them time, because they can pre-emptively change their travel plans, and no longer have to waste countless hours due to flight delays.

This concludes this paper’s description of the Get-Build-Analyze framework, and the use of IBM Analytics for Apache Spark, along with other Bluemix services, to develop a flight delay prediction business application.
# Next steps

1. See the flight delay application on [Github](https://github.com).

2. Sign up for a free 30-day trial of the IBM Analytics for Apache Spark. Get started today, and jump right into a Notebook environment!

<table>
<thead>
<tr>
<th>Introduction</th>
<th>Section 1: How to predict flight delays using Apache Spark MLlib</th>
<th>Section 2: How building strong analytical models drives better business decisions</th>
<th>Section 3: Gain business advantage by analyzing data and using predictive models</th>
<th>Appendix</th>
</tr>
</thead>
</table>

<28>
Introduction

Section 1: How to predict flight delays using Apache Spark MLlib

Section 2: How building strong analytical models drives better business decisions

Section 3: Gain business advantage by analyzing data and using predictive models

Appendix

The following image illustrates the tasks and products used in each of the three phases of the Get-Build-Analyze methodology:

Get Data phase
- Cloudant
- dashDB
- Simple Data Pipe
- Twitter
- SQLDB
- DataWorks
- Weather
- Secure Gateway

Build Data phase
- Streaming Analytics
- Spark
- Notebooks
- R Studio
- IBM BigInsights®
- Predictive Analytics

Analyze Data phase
- Chartio
- Looker
- Watson Analytics
- Other business intelligence tools
- Custom dashboards