

[Downloads \(http://www.cloudera.com/content/www/en-us/downloads.html\)](http://www.cloudera.com/content/www/en-us/downloads.html)
[Training \(http://www.cloudera.com/content/www/en-us/training.html\)](http://www.cloudera.com/content/www/en-us/training.html)
[Support Portal \(http://www.cloudera.com/content/www/en-us/support.html\)](http://www.cloudera.com/content/www/en-us/support.html)
[Partners \(http://www.cloudera.com/content/www/en-us/partners.html\)](http://www.cloudera.com/content/www/en-us/partners.html)
[Developers \(http://www.cloudera.com/content/www/en-us/developers.html\)](http://www.cloudera.com/content/www/en-us/developers.html)
[Community \(http://community.cloudera.com/\)](http://community.cloudera.com/)

cloudera
(<http://www.cloudera.com>)

Cloudera Engineering Blog

(<http://blog.cloudera.com/>)

Best practices, how-tos, use cases, and internals from Cloudera Engineering and the community

SEARCH

How-to: Predict Telco Churn with Apache Spark MLlib

February 16, 2016 (<http://blog.cloudera.com/blog/2016/02/how-to-predict-telco-churn-with-apache-spark-mlib/>) | By Juliet Hougland and Sandy Ryza (<http://blog.cloudera.com/?guest-author=Juliet Hougland and Sandy Ryza>) | No Comments (<http://blog.cloudera.com/blog/2016/02/how-to-predict-telco-churn-with-apache-spark-mlib/#comments>)

Categories: [Data Science \(http://blog.cloudera.com/blog/category/data-science/\)](http://blog.cloudera.com/blog/category/data-science/) [Spark \(http://blog.cloudera.com/blog/category/spark/\)](http://blog.cloudera.com/blog/category/spark/) [Use Case \(http://blog.cloudera.com/blog/category/use-case/\)](http://blog.cloudera.com/blog/category/use-case/)

Spark MLlib is growing in popularity for machine-learning model development due to its elegance and usability. In this post, you'll learn why.

Tweets by
[@ClouderaEng](#)



Cloudera Engin...
[@ClouderaEng](#)

BigDL on CDH and Cloudera
Data Science Workbench
[j.mp/2ppWCkh](#)

BigDL on CDH and Cl...
Introduction As compa...
[blog.cloudera.com](#)

[Embed](#) [View on Twitter](#)

Categories

[Spark MLlib \(http://spark.apache.org/docs/latest/mllib-guide.html\)](http://spark.apache.org/docs/latest/mllib-guide.html) is a library for performing machine-learning and associated tasks on massive datasets. With MLlib, fitting a machine-learning model to a billion observations can take a couple lines of code and leverage hundreds of machines. MLlib greatly simplifies the model development process.

In this post, we'll use MLlib to fit a machine-learning model that can predict which customers of a telecommunications company are likely to stop using their service. *Churn prediction*, is one of the most common applications of machine learning in the telecommunications industry, as well as many other subscriptions-based industries.

We'll carry out our analysis and modeling using the Python programming language, and we'll apply a variety of connected tools for the task. To load and manipulate the data, we'll make use of Spark's [DataFrames](http://spark.apache.org/docs/latest/sql-programming-guide.html) (<http://spark.apache.org/docs/latest/sql-programming-guide.html>) API. To perform feature engineering, model fitting, and model evaluation, we'll use Spark's [ML Pipelines](http://spark.apache.org/docs/latest/ml-guide.html) (<http://spark.apache.org/docs/latest/ml-guide.html>) API. (The core of MLlib is shipped inside CDH 5.5 supported in Cloudera Enterprise 5.5, but ML Pipelines will not be supported until a future release.)

This post is based off of the material we presented at our ["Data Science for Telecom" tutorial](http://conferences.oreilly.com/strata/big-data-conference-sg-2015/public/schedule/detail/45272) (<http://conferences.oreilly.com/strata/big-data-conference-sg-2015/public/schedule/detail/45272>) at Strata + Hadoop World Singapore 2015. The full source code, with outputs, is available in an [IPython notebook](https://github.com/jhlch/ds-for-telco/blob/master/ds-for-telco-with-output.ipynb) (<https://github.com/jhlch/ds-for-telco/blob/master/ds-for-telco-with-output.ipynb>). The repository also contains a [script](https://github.com/jhlch/ds-for-telco/blob/master/launch-ipython.sh) (<https://github.com/jhlch/ds-for-telco/blob/master/launch-ipython.sh>) showing how one might launch an IPython notebook with the required dependencies on a CDH cluster.

Loading the Data with Spark DataFrames

We'll fit our model to a churn dataset provided by the UC Irvine machine-learning repository [hosted by SGI](https://www.sgi.com/tech/mlc/db/churn.all) (<https://www.sgi.com/tech/mlc/db/churn.all>). In this dataset,

[Accumulo \(http://blog.cloudera.com/blog/category/accumulo/\)](http://blog.cloudera.com/blog/category/accumulo/) (1)
[Avro \(http://blog.cloudera.com/blog/category/avro/\)](http://blog.cloudera.com/blog/category/avro/) (21)
[Bigtop \(http://blog.cloudera.com/blog/category/bigtop/\)](http://blog.cloudera.com/blog/category/bigtop/) (6)
[Books \(http://blog.cloudera.com/blog/category/books/\)](http://blog.cloudera.com/blog/category/books/)

Popular

[Most Popular \(http://blog.cloudera.com/most-popular/\)](http://blog.cloudera.com/most-popular/)
[Most Popular in /spark/ \(http://blog.cloudera.com/most-popular-in-spark/\)](http://blog.cloudera.com/most-popular-in-spark/)

Tags

[analysis \(http://blog.cloudera.com/blog/tag/analysis/\)](http://blog.cloudera.com/blog/tag/analysis/)
[analytics \(http://blog.cloudera.com/blog/tag/analytics/\)](http://blog.cloudera.com/blog/tag/analytics/)
[apache \(http://blog.cloudera.com/blog/tag/apache/\)](http://blog.cloudera.com/blog/tag/apache/)
[apache hadoop \(http://blog.cloudera.com/blog/tag/apache-hadoop/\)](http://blog.cloudera.com/blog/tag/apache-hadoop/)
[Apache HBase \(http://blog.cloudera.com/blog/tag/apache-hbase/\)](http://blog.cloudera.com/blog/tag/apache-hbase/)
[apache hive \(http://blog.cloudera.com/blog/tag/apache-hive/\)](http://blog.cloudera.com/blog/tag/apache-hive/)
[beta \(http://blog.cloudera.com/blog/tag/beta/\)](http://blog.cloudera.com/blog/tag/beta/)
[Big Data \(http://blog.cloudera.com/blog/tag/big-data/\)](http://blog.cloudera.com/blog/tag/big-data/)
[CDH \(http://blog.cloudera.com/blog/tag/cdh/\)](http://blog.cloudera.com/blog/tag/cdh/)
[cloudera](http://blog.cloudera.com/blog/tag/cloudera/)

each record contains information corresponding to a single subscriber, as well as whether that subscriber went on to stop using the service.

The dataset contains only 5,000 observations, i.e. subscribers, many orders of magnitude smaller than what Spark can handle, but playing with data of this size makes it easy to try out the tools on a laptop.

The full set of fields, from the data subscription, are:

state
account length
area code
phone number
international plan
voice mail plan
number vmail messages
total day minutes
total day calls
total day charge
total eve minutes
total eve calls
total eve charge
total night minutes
total night calls
total night charge
total intl minutes
total intl calls
total intl charge
number customer service calls
churned

The last field, “churned”, a categorical variable that can take the values “true” or “false”, is the label we would like to predict. The rest of the fields are fair game for use in creating *independent* variables, which are used in combination with a model to generate predictions.

To load this data into a Spark DataFrame, we just need to tell Spark the type of each field. We use the [spark-csv package \(https://github.com/databricks/spark-csv\)](https://github.com/databricks/spark-csv), which lives outside of the main Spark project, to interpret CSV-formatted data:

(<http://blog.cloudera.com/blog/tag/cloudera/>) [Cloudera Manager \(http://blog.cloudera.com/blog/tag/cloudera-manager/\)](http://blog.cloudera.com/blog/tag/cloudera-manager/) [Community \(http://blog.cloudera.com/blog/tag/community/\)](http://blog.cloudera.com/blog/tag/community/) [configuration \(http://blog.cloudera.com/blog/tag/configuration/\)](http://blog.cloudera.com/blog/tag/configuration/) [data \(http://blog.cloudera.com/blog/tag/data/\)](http://blog.cloudera.com/blog/tag/data/) [developer \(http://blog.cloudera.com/blog/tag/developer/\)](http://blog.cloudera.com/blog/tag/developer/) [developers \(http://blog.cloudera.com/blog/tag/developers/\)](http://blog.cloudera.com/blog/tag/developers/) [development \(http://blog.cloudera.com/blog/tag/development/\)](http://blog.cloudera.com/blog/tag/development/) [events \(http://blog.cloudera.com/blog/tag/events-2/\)](http://blog.cloudera.com/blog/tag/events-2/) [Flume \(http://blog.cloudera.com/blog/tag/flume/\)](http://blog.cloudera.com/blog/tag/flume/) [Hadoop \(http://blog.cloudera.com/blog/tag/hadoop/\)](http://blog.cloudera.com/blog/tag/hadoop/) [hadoop world \(http://blog.cloudera.com/blog/tag/hadoop-world/\)](http://blog.cloudera.com/blog/tag/hadoop-world/) [HBase \(http://blog.cloudera.com/blog/tag/hbase/\)](http://blog.cloudera.com/blog/tag/hbase/) [HDFS \(http://blog.cloudera.com/blog/tag/hdfs/\)](http://blog.cloudera.com/blog/tag/hdfs/) [Hive \(http://blog.cloudera.com/blog/tag/hive/\)](http://blog.cloudera.com/blog/tag/hive/) [Hue \(http://blog.cloudera.com/blog/tag/hue/\)](http://blog.cloudera.com/blog/tag/hue/) [impala \(http://blog.cloudera.com/blog/tag/impala-2/\)](http://blog.cloudera.com/blog/tag/impala-2/) [installation \(http://blog.cloudera.com/blog/tag/installation/\)](http://blog.cloudera.com/blog/tag/installation/)

```

from pyspark.sql import SQLContext
from pyspark.sql.types import *

sqlContext = SQLContext(sc)
schema = StructType([ \
    StructField("state", StringType(), True), \
    StructField("account_length", DoubleType(), True), \
    StructField("area_code", StringType(), True), \
    StructField("phone_number", StringType(), True), \
    StructField("intl_plan", StringType(), True), \
    StructField("voice_mail_plan", StringType(), True), \
    StructField("number_vmail_messages", DoubleType(), True), \
    StructField("total_day_minutes", DoubleType(), True), \
    StructField("total_day_calls", DoubleType(), True), \
    StructField("total_day_charge", DoubleType(), True), \
    StructField("total_eve_minutes", DoubleType(), True), \
    StructField("total_eve_calls", DoubleType(), True), \
    StructField("total_eve_charge", DoubleType(), True), \
    StructField("total_night_minutes", DoubleType(), True), \
    StructField("total_night_calls", DoubleType(), True), \
    StructField("total_night_charge", DoubleType(), True), \
    StructField("total_intl_minutes", DoubleType(), True), \
    StructField("total_intl_calls", DoubleType(), True), \
    StructField("total_intl_charge", DoubleType(), True), \
    StructField("number_customer_service_calls", DoubleType(), True), \
    StructField("churned", StringType(), True)])

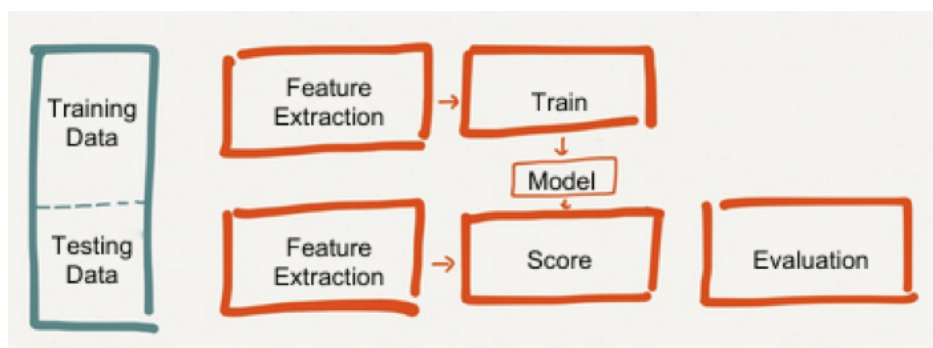
churn_data = sqlContext.read \
    .format('com.databricks.spark.csv') \
    .load('churn.all', schema = schema)

```

Fitting a Machine-Learning Model

MLlib provides a set of algorithms for fitting machine-learning models to large data sets and performing related statistical processing. In particular, here we'll make use of the ML Pipelines API, which is a framework for taking data in DataFrames, applying transformations to extract features, and feeding the extracted data into machine learning algorithms. We will use MLlib to train and evaluate a Random Forest (https://en.wikipedia.org/wiki/Random_forest) model that can predict whether a user is likely to churn.

The broad flow of supervised machine-learning model development and evaluation looks like this:



(<http://blog.cloudera.com/wp-content/uploads/2016/02/ml.png>)

[tion/\) java \(http://blog-cloudera.com/blog/tag/java/\)](http://blog.cloudera.com/blog/tag/java/)
[a/\) log \(http://blog.cloudera.com/blog/tag/log/\)](http://blog.cloudera.com/blog/tag/log/)
[logs \(http://blog.cloudera.com/blog/tag/logs/\)](http://blog.cloudera.com/blog/tag/logs/)
[Map-Reduce \(http://blog.cloudera.com/blog/tag/mapreduce/\)](http://blog.cloudera.com/blog/tag/mapreduce/)
[monitoring \(http://blog.cloudera.com/blog/tag/monitoring/\)](http://blog.cloudera.com/blog/tag/monitoring/)
[open source \(http://blog-cloudera.com/blog/tag/open-source/\)](http://blog.cloudera.com/blog/tag/open-source/)
[Pig \(http://blog.cloudera.com/blog/tag/pig/\)](http://blog.cloudera.com/blog/tag/pig/)
[platform \(http://blog.cloudera.com/blog/tag/platform/\)](http://blog.cloudera.com/blog/tag/platform/)
[python \(http://blog.cloudera.com/blog/tag/python/\)](http://blog.cloudera.com/blog/tag/python/)
[questions \(http://blog-cloudera.com/blog/tag/questions/\)](http://blog.cloudera.com/blog/tag/questions/)
[release \(http://blog.cloudera.com/blog/tag/release/\)](http://blog.cloudera.com/blog/tag/release/)
[REST \(http://blog.cloudera.com/blog/tag/rest/\)](http://blog.cloudera.com/blog/tag/rest/)
[Search \(http://blog.cloudera.com/blog/tag/search/\)](http://blog.cloudera.com/blog/tag/search/)
[security \(http://blog.cloudera.com/blog/tag/security/\)](http://blog.cloudera.com/blog/tag/security/)
[sql \(http://blog.cloudera.com/blog/tag/sql/\)](http://blog.cloudera.com/blog/tag/sql/)
[Support \(http://blog.cloudera.com/blog/tag/support-2/\)](http://blog.cloudera.com/blog/tag/support/)
[Testing \(http://blog.cloudera.com/blog/tag/testing/\)](http://blog.cloudera.com/blog/tag/testing/)
[use cases \(http://blog-cloudera.com/blog/tag/use-cases/\)](http://blog.cloudera.com/blog/tag/use-cases/)



The flow starts with a dataset, composed of columns with possibly a variety of types. In our case, this is the `churn_data` we created in the section above. We then perform *feature extraction* on this data to transform it into a set of *feature vectors* and *labels*. A feature vector is an array of floating point values representing the independent variables our model can use to make a prediction. A label is a single floating point value representing the dependent variable that our machine learning algorithm is trying to predict. In binary classification problems such as ours, we use 0.0 and 1.0 to represent the two possible predictions. In our case, 0.0 means “will not churn” and 1.0 means “will churn.”

Feature extraction refers to a wide set of possible transformations we might care to conduct produce feature vectors and labels from the input data. In our case, like to take categorical variables that are represented in the input data as strings, like `intl_plan`, and *index* them to turn them into numbers.

We’d like select a subset of the columns. For example, we don’t expect that `phone_number` is likely to be a very useful feature, so we can leave it out of our model, but `total_day_calls` is likely to be, so we’d like to include it. We incorporate these transformation steps into our pipeline by defining two stages: `StringIndexer` and `VectorAssembler`.

```
1 from pyspark.ml.feature import StringIndexer
2 from pyspark.ml.feature import VectorAssembler
3
4 label_indexer = StringIndexer(inputCol = 'churned', outputCol = 'churn_label')
5 plan_indexer = StringIndexer(inputCol = 'intl_plan', outputCol = 'intl_plan_indexed')
6
7 reduced_numeric_cols = ["account_length", "number_vmail_messages",
8                         "total_day_charge", "total_eve_calls",
9                         "total_night_calls", "total_intl_calls"]
10
11 assembler = VectorAssembler(
12     inputCols = ['intl_plan_indexed'] + reduced_numeric_cols,
13     outputCol = 'features')
```

Having extracted features, our next step is to split up our dataset into *train* and *test* sets. The train set will be used by the machine learning algorithm to fit the model. The test set will be used to evaluate the model:

```
(train, test) = churn_data.randomSplit([0.7, 0.3])
```

Now we can assemble our pipeline and finally fit the model. An advantage of defining a pipeline is that you know that the same code is getting applied for the feature-extraction step. With MLlib, this is a few short lines of code!

```
from pyspark.ml import Pipeline
from pyspark.ml.classification import RandomForestClassifier

classifier = RandomForestClassifier(labelCol = 'label', featuresCol = 'features')
pipeline = Pipeline(stages=[plan_indexer, label_indexer, assembler, classifier])
model = pipeline.fit(train)
```

Validating the Model

How do we know whether the model we've trained is a good one? Can we show that the predictions it produces are better than random guessing? For binary classification models, a useful evaluation metric is the area under the ROC curve (https://en.wikipedia.org/wiki/Receiver_operating_characteristic). An ROC curve is created by taking a binary classification predictor that uses a threshold value to assign labels given predicted continuous values. As you vary the threshold for a model you cover from the two extremes, when the true positive rate (TPR) and the false positive rate (FPR) are both 0 because everything is labeled “not churned” and when both the TPR and FPR are both 1 because everything is labeled “churned.”

A random predictor that labels a customer as churned half the time and not churned the other half would have a ROC that was a straight diagonal line. This line cuts the unit square into two equally-sized triangles, so the area under the curve is 0.5. An AUROC value of 0.5 would mean that your predictor was no better at discriminating between the two classes than random guessing. The closer the value is to 1.0, the better its predictions are. A value below 0.5 indicates that we could actually make our model produce better predictions by reversing the answer it gives us.

MLlib also makes computing the AUROC exceedingly easy. If we were to compute the ROC curve based on all of our data, our classification evaluation metrics would be overly optimistic because we would be evaluating a model with the same data we trained on. We perform model evaluation only

on our test set to avoid overly optimistic model evaluation metrics (like AUROC) as well as to help us avoid overfitting (<https://en.wikipedia.org/wiki/Overfitting>).

```
from pyspark.ml.evaluation import BinaryClassificationEvaluator

predictions = model.transform(test)
evaluator = BinaryClassificationEvaluator()
auroc = evaluator.evaluate(predictions, {evaluator.metricName: "auROC"}
```

In this case we produce an area under the curve greater than 0.8, indicating that the model's results are reasonably good, and definitely better than random guessing.



Conclusion

This post provides just one example of a possible use case for MLlib. For more examples of how-tos around machine learning and Spark in general, [see this list](http://www.cloudera.com/developers/how-tos/apache-spark-how-tos.html) (<http://www.cloudera.com/developers/how-tos/apache-spark-how-tos.html>).

Juliet Hougland is a Data Scientist at Cloudera, and contributor/committer/maintainer for the Sparkling Pandas project.

Sandy Ryza is a Data Scientist at Cloudera, and a committer to the Apache Spark and Apache Hadoop projects. He is a co-author of [Advanced Analytics with Spark](http://shop.oreilly.com/product/0636920035091.do) (<http://shop.oreilly.com/product/0636920035091.do>), from O'Reilly Media.



 mllib (<http://blog.cloudera.com/blog/tag/mllib/>)  telco (<http://blog.cloudera.com/blog/tag/telco/>)

New SQL Benchmarks: Apache Impala (incubating) Uniquely Delivers Analytic Database Performance	How-to: Build a Real-Time Search System using StreamSets, Apache Kafka, and Cloudera Search
http://blog.cloudera.com/blog/2016/02/new-sql-benchmarks-apache-impala-incubating-2-3-	http://blog.cloudera.com/blog/2016/02/how-to-build-a-real-time-search-system-using-streamsets-

uniquely-delivers-analytic-
database-performance/)

apache-kafka-and-cloudera-
search/)

Partners

(<https://www.cloudera.com/partners.html>)

Developers

(<https://www.cloudera.com/developers.html>)

Community

(<http://community.cloudera.com/>)

Resources

(<https://www.cloudera.com/resources.html>)

Documentation

(<https://www.cloudera.com/documentation.html>)

Career

(<http://jobs.jobvite.com/cloudera/>)

Contact

(<https://www.cloudera.com/contact-us.html>)

United States: +1 888 789 1488

Outside the US: +1 650 362 0488

(<https://www.linkedin.com/company/cloudera>)

(<https://www.facebook.com/cloudera>)

(<https://twitter.com/cloudera>)

(<https://www.cloudera.com/contact-us.html>)

Terms & Conditions (<https://www.cloudera.com/legal/terms-and-conditions.html>) |

Policies

(<https://www.cloudera.com/legal/policies.html>)