AI – From Zero to Hero

SQL Saturday

2018

1 Titanic - Who's gonna make it?

A ship is sinking. Today you are going to predict who will survive. We provide you with a dataset of the passengers of the Titanic including the information whether the passenger survived or not. Your task is to train a model that we can use to predict passenger survival. . . just in case somebody will ever start a Titanic again.

We will exemplify the typical data science task based on the given dataset and the above task.

1.1 Load data

The first thing to do is to load the data. Before that you might need to import some Python packages:

```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import matplotlib.pyplot as plt #for plotting graphs
```

The latter is stored as a csv file. It is available at the mount point "/mnt/data/titanic/train.csv".

Load the data and convert it to a Pandas dataframe:

```
rdd = sqlContext.read.csv([MOUNT_POINT], header=True, inferSchema=True)
```

Solution:

```
rdd = sqlContext.read.csv('/mnt/data/titanic/train.csv', header=True,
                                                 inferSchema=True)
```

An alternative way of loading the data is to create temporary SQL table. This can be achieved via:

```
%sql
CREATE TEMPORARY TABLE temp_titanic
  USING csv
  OPTIONS (path "[MOUNT_POINT]", header "true", mode "FAILFAST")
```

Note that the mode ”FAILFAST” will abort file parsing with a RuntimeException if any malformed lines are encountered.

Solution:

```
%sql
CREATE TEMPORARY TABLE temp_titanic
  USING csv
  OPTIONS (path "/mnt/data/titanic/train.csv", header "true", mode "
                                              FAILFAST")
```

1.2 Explore your data

Ok, it is time that you get to know your data.

In Python you should first try to convert your rdd object to a Pandas dataframe and then call its `head()` method.

```
df = rdd.toPandas()
df.head()
```

Try to call `df.shape` too. Make sure it is the last program statement of the cell as it won’t be displayed otherwise.

If you created a temporary table in SQL, try:

```
%sql SELECT * FROM temp_titanic
```

and experiment with the plot options below. Customize your SQL query to your needs.

In general the data exploration phase aims at finding good features for an machine learning algorithm. We are first of all interested whether the data could be incomplete, i.e., some values may be missing. ML algorithms cannot handle this and we need to get rid of them, either by providing our own guess for missing values or by deleting incomplete data points. Another thing ML algorithms cannot handle is string data. In fact ML algorithms can only handle numerical data. So if we want to use any of it, we need to convert into something else “that is meaningful”.

Pandas dataframes have some very convenient methods for the exploration task. Try some of the following and look at the output:

```
df.isnull().sum()
df['<column_name>'].isnull()
df['<column_name>'].hist()
```

There are plenty of plots you can create from your data within a few steps. Consider visiting `https://matplotlib.org/tutorials/index.html` to get to know the matplotlib library. In some cases your plot won’t appear in Databricks, in which case you well have to envelope your plot statement with e.g.,

```
df.hist()
display(plt.show())
# or
displayHTML(..)
# if you know it is HTML content
```

Solution:

```
import pandas_profiling
report = pandas_profiling.ProfileReport(df)
displayHTML(report.to_html())
```

```
#new cell
df.dtypes
```

```
pd.options.display.line_width = 200
df.head()
```

```
df.shape
```

1.3 Clean Up

However, even if we took care of all missing values, we might technically be ready to run ML on it, but it doesn't mean, all of the data is useful. Some of our columns might have no value to our particular classification task, e.g., the passenger ID won't influence a person's survival.

Further experiment with plots that will expose meaningless data or highly correlated data. The latter usually means your data is redundant in some sense and you can drop some of it.

At this point, we hope you came to conclusion that the columns `PassengerId`, `Cabin`, `Ticket`, `Fare` and `Name` are useless. You can drop a column by the following command:

```
df.drop(a_list_of_column_names, axis=1, inplace=True)
```

Solution:

```
df.drop(['PassengerId', 'Cabin', 'Ticket', 'Fare'], axis=1, inplace=True)
```

Wait before you drop the `Name` column as we can use it to fill some of the missing `Age` values.

Try to come up with a good guess for each missing `Age` value.

Solution:

```
import re
df_ini=[] # list for storing initials
for i in df['Name']:
    df_ini.append(re.findall(r"\w+\.",i))

#checking the indexes where age is null
df_null_age_index = np.where(df['Age'].isnull())

#define new age
import random
from random import randint
```

```
df_newages=[]
for i in range(len(df_null_age_index)):
    if 'Mr.' or 'Mrs.' in df_ini[df_null_age_index[i]]:
        df_newages.append(random.randint(20,35))

    elif 'Master.' or 'Miss.' in df_ini[df_null_age_index[i]]:
        df_newages.append(random.randint(10,18))

#assign values to missing indices
for i in range(len(df_newages)):
    df['Age'][df_null_age_index.pop()]=df_newages[i]

#eventually drop Name column
df.drop(['Name'], axis=1, inplace=True)
```

```
#last check if anything is null
df.isnull().any()
```

```
#as a final step, organize the data already into a feature dataframe X
                                    and its labels y
X = df.drop('Survived', axis=1)
y = df['Survived']
X.head()
```

1.4 Categorical Data

Ok, our data should be cleaned up at this point and we should have a good idea of what's in our data. Still, we have categorical data which would be the columns `Sex` and `Embarked`. `Pclass` is actually also a categorical feature, but we can leave as it is.

These must be encoded as binary features for which you can use `LabelEncoder`[1] and `OneHotEncoder`[2] Check out their documentation.

Solution:

```
from sklearn.preprocessing import LabelEncoder, OneHotEncoder

labelencoder_X = LabelEncoder()
#transform columns 1 and 5
X.iloc[:,1] = labelencoder_X.fit_transform(X.iloc[:,1])
X.iloc[:,5] = labelencoder_X.fit_transform(X.iloc[:,5])

onehotencoder = OneHotEncoder(categorical_features=[5], sparse=False)
oh_encoded = onehotencoder.fit_transform(X)
```

[1]http://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.LabelEncoder.html

[2]http://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.OneHotEncoder.html

1.5 Split Data into Training and Test Data

Whenever you train a model, you also want test its performance. Thus, you will need to reserve a dataset for testing, for which you need to ensure that is not exposed to your model during the training. Otherwise, your model could simply remember the label for each training sample, which would trivialize prediction on known data. For unknown data the results will likely be bad. There is a convenience method[3] that can help you.

Solution:

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
```

1.6 Training your model

There are plenty of classifier algorithms you could use at this point. Instantiate one, train your model and compute the accuracy of its predictions. Checkout this page for an example of different classifier algorithms you could use including the required code: `http://scikit-learn.org/stable/auto_examples/classification/plot_classifier_comparison.html`

Solution:

```
#instantiate your classifier
random_forest = RandomForestClassifier(n_estimators=100)
#fit (train) the model on the training data
random_forest.fit(X_train, y_train)
#evaluate on your test data
random_forest.score(X_test, y_test)
# the result of the last call will output the accuracy of your model
```

other alternatives:

```
sgd = SGDClassifier()
sgd.fit(X_train, y_train)
sgd.score(X_test, y_test)
```

```
dtc = DecisionTreeClassifier()
dtc.fit(X_train, y_train)
dtc.score(X_test, y_test)
```

2 Higgs - Let's detect the famous Higgs Boson (The Spark Way)

Some of you may have heard about the Higgs Boson particle. It was found just a few years ago, Not only is its physics difficult, but also the classification between random noise and an actually detected Higgs particle. This time, we are going to detect Higgs particles within random noise.

[3] `http://scikit-learn.org/stable/modules/generated/sklearn.cross_validation.train_test_split.html`

2.1 Load Data

Just as previously, we can use the following command, but since the csv file does not have header `header` is `False`.

```
rdd = sqlContext.read.csv([MOUNT_POINT], header=False, inferSchema=True)
```

Now try to convert the data to a Pandas dataframe:

```
df = rdd.toPandas()
```

Note that the dataset contains 11 million entries and if your machine does not crash, it should take at least quite a while to finish. The alternative is to use Spark dataframes, which are probably less convenient than Pandas dataframes, but enable us to handle really large dataset. If you choose to continue using Pandas, follow the steps from the Titanic task. Your `rdd` object is already a Spark dataframe. So, we are done here.

2.2 Explore (and Clean) your Data

Again explore your data and try to find missing values or maybe incorrect types. Check the type of each column. Correct it, if necessary. If you want to gather some more information on the data, check out `https://archive.ics.uci.edu/ml/datasets/HIGGS`

```
df.printSchema()
```

```
display(df.take(5))
```

```
from pyspark.sql.types import *
df = df.withColumn('_c0', df['_c0'].cast(IntegerType()))
```

```
display(df.describe())
```

Feel free to select a subset of the given columns for the training. The choice is entirely up to you. In case of the Higgs dataset, the cleanup can even be ignored entirely as you should see during your data exploration. Note, as a data scientist you will rarely see a dataset that clean ☺

2.3 Training your Model

Ok, at this point, we have to setup the machine learning pipeline. The ML algorithms in Spark assume you have a label vector and a feature vector. Each value in the feature vector is again a vector and consists of different values (what was previously a row in a dataframe). You will need the following imports

```
from pyspark.ml.linalg import DenseVector
from pyspark.ml.feature import MinMaxScaler
from pyspark.ml.classification import RandomForestClassifier,
                                       LogisticRegression
```

The final dataframe should have two columns: 'label' and 'features'. Do not forget to scale your data, since some ML algorithm can be very sensitive to data being scaled or not.

```
# note the first column becomes the label, the remainder is the features

from pyspark.ml.linalg import DenseVector
# Define the `input_data`
input_data = df.rdd.map(lambda x: (x[0], DenseVector(x[1:])))

df_input = spark.createDataFrame(input_data, ["label", "features"])
df_input
```

```
from pyspark.ml.feature import MinMaxScaler
# Initialize the `standardScaler`
scaler = MinMaxScaler(inputCol="features", outputCol="features_scaled")
# Fit the DataFrame to the scaler
scaler = scaler.fit(df_input)
```

```
# Transform the data in `df` with the scaler
scaled_df = scaler.transform(df_input)
scaled_df.first()
```

Again, separate your data into training and test data. Once this is finished, instantiate the classifiction algorithm of your choice and train your model.

Eventually, predict on your test dataset and evaluate your prediction.

```
train_data, test_data = scaled_df.randomSplit([.8,.2], seed=7)

from pyspark.ml.classification import RandomForestClassifier,
                                      LogisticRegression

# Initialize `lr`
lr = LogisticRegression(labelCol="label",featuresCol='features_scaled',
                                      maxIter=10, regParam=0.3,
                                      elasticNetParam=0.8)
rfr = RandomForestClassifier(labelCol='label', featuresCol='
                                      features_scaled')

# Fit the data to the model
linearModel = lr.fit(train_data)
randomForestModel = rfr.fit(train_data)
```

```
#predict on test data
pred_log = linearModel.transform(test_data)
pred_rand = randomForestModel.transform(test_data)
```

```
#evaluate
import sklearn.metrics as metrics
# This is not a good idea for big data
```

```
acc_log = metrics.accuracy_score(pred_log.select('label').collect(),
                                    pred_log.select("prediction").
                                    collect())
acc_rand = metrics.accuracy_score(pred_rand.select('label').collect(),
                                    pred_rand.select("prediction").
                                    collect())

print(acc_log)
print(acc_rand)
```