CS 5304 Data Science in the Wild, Spring 2017
Assignment 1

Jui-Chun Chien
Net ID: JC3256

The Titanic Data Set

Data Description and Data Cleaning

Here is an introduction of the 12 features in the data set and how I processed them.

PassengerId
A unique number assigned to each passenger on file. According to the files, there are 891 people recorded in the training set, 418 people in the test set. The id numbers in fact do not contain relevant information to the survival of a passenger. **Dropped**

Pclass
The Passenger Class each passenger belongs to. 1 to 3 stands for the three classes there are, with 1 being first class and 3 being third class. The passenger class of a passenger might lead to a person's different location on a boat and her/his priority to board the lifeboats, therefore is considered relevant information.

Survived
The survival statistics of a passenger. 1 stands for survived, 0 stands for deceased. This is the label of the dataset.

Name
A string for each passenger indicating their name. From inference, there shouldn't be largely relevant to survival(except for the case where we might be able to predict the survival rate of members in a family), therefore the column is dropped. **Dropped**

Sex
Indicates whether a passenger is male or female. It could be a relevant information since male and female have different biological structures and capabilities, and different genders play

different roles. The column is kept and replaced by integers (1: male, 0: female) for ease of processing.

Age
A possible relevant information for survival as people in their stronger years (20-30) could have a bigger chance of survival. There are cells without data in this column, and I filled in the missing data with the average age of all passengers. I thought of using the average age of a subgroup (ex: if a passenger's age is missing, find out other passengers who have similar values in other fields, such as sex, parent/children number, and number of siblings and spouses, and then average the age of the specific subgroup.) but turns out there would be too little data of the same criteria, and was not adopted.

SibSp
The number of siblings and spouse a person has. This is an indicator of the role of a person, even the social status of a person. It is therefore considered relevant information.

Parch
The number of parent and children a passenger has. This could also imply the responsibility a person has and the mobility of a person while escaping the disaster.

Ticket
The ticket number of a person. It is mostly unstructured data and comes in different formats, without much information in the numbers or characters. Therefore, I did not include the column. **Dropped**

Fare
The price I person paid for the ticket. At first glance, this seems like important information. But upon inspecting it, it is not hard to realise the illogical details: price ranges for a few dollars to several thousand dollars, while a first class fare might be a few hundred and a third class fare might be more than five thousand dollars. I believe the data is somehow corrupted and best to leave it dropped instead of fixed. **Dropped**

Cabin
The specific cabin number of a person. I also dropped the column because there is too many missing information, and is correlated with the Pclass feature already. With complete information, we might have been able to infer which cabin locations were easier for passengers to escape. **Dropped**

Embarked

The place where a passenger boarded the ship. C stands for Cherbourg, Q for Queenstown, and S for Southampton. The column is also dropped eventually because the information is not directly linked to survival according to correlation. **Dropped**

Models, Analysis, and Prediction

I have chosen to build logistic regression models and random forest models for classification. Random Forest is a good choice as it segmentize some features into categories to make sense of the data instead of continuous data. According to experiment, Random Forest also has a very high accuracy on the training set. Naive Bayes may not be a good choice because it assumes all columns/features are independent of each other. Therefore when two classes of information happen to be highly related and be a clear indicator for the resulting label, it will give the same cause too much weight.

	F-Score	Accuracy
Logistic Regression	0.79620815173	0.801801565438
Random Forest	0.790934039729	0.868157548978

Women among all subgroups of people are most likely to survived. Among 500+ males, only 100+ survived, while among 300+ females, 200+ survived. This is a clear difference and is possibly due to the norm that women and children are allowed to board the lifeboats first. A young woman with less siblings has the best chance of survival.

Code Explanations

Below, I will talk briefly about my code structure and the methods used.

```
df = spark.read.csv("./train.csv", header=True, inferSchema=True)
df =
df.drop('PassengerId').drop('Name').drop('Ticket').drop('Fare').drop('Cabin').drop('E
mbarked')
df = df.na.replace(['male', 'female'], ['1', '0'], 'Sex')
df = df.withColumn('Sex', df.Sex.cast('int'))
ave_age = df.agg(F.avg(df['age'])).collect()[0][0]
```

```
df = df.na.fill({'age': ave_age})
vecAssembler = VectorAssembler(inputCols=["Pclass", "Sex", "Age", "SibSp", "Parch"],
outputCol="features")
df = vecAssembler.transform(df)
df = df.select('Survived', 'features')

df.show()
df.printSchema()
```

First of all, I used the spark.csv.read function to load the .csv file which is in the same directory of the .py file. I also set the header parameter to true so each column has a name instead of a random string to represent the data. I then executed all the drop functions at once for clarity. Next, I replaced the parameters in the Sex feature class with 1 and 0, using the na.replace function, and then used the withColumn function to change the newly generated numbers in the Sex column from datatype of string to integer. Failing to notice and do this stopped me from doing the VectorAssembler for a while. The next step was to use the aggregate function and the built-in arithmetic functions to calculate the average age of all passengers in addition to the ones without such data. The na.fill function then fills in the cells with Null values with the average age. The VectorAssembler is for creating a new feature vector with several columns combined, and it would append the whole combined feature vector to the end of the original dataframe, creating an additional column, naming the class as features. What I did is first set up the parameters and the classes i would like to combine, and then applied it on the dataset. Later on, I selected only the label, which is the Survival statistics, and the features vector at the end for further processing.

```
lr = LogisticRegression(featuresCol="features", labelCol="Survived")
grid = ParamGridBuilder().addGrid(lr.maxIter, [10, 30, 50]).addGrid(lr.regParam,
[0.01, 0.1, 1, 5]).build()
evaluator = MulticlassClassificationEvaluator(predictionCol="prediction",
labelCol="Survived")
cv = CrossValidator(estimator=lr, estimatorParamMaps=grid, evaluator=evaluator)
best_cv_Model = cv.fit(df)
f_score = best_cv_Model.avgMetrics[0]
accuracy = evaluator.evaluate(best_cv_Model.transform(df))

print 'Accuracy for Logistic Regression: ', accuracy
print 'F-Score for Logistic Regression: ', f_score
```

After I processed the test set in a similar fashion, I started implementing the logistic regression model. I first indicated in the logistic regression function that features will be the feature vector while Survived will be the labels. I then went on to map out the parameters I

would like the cross validation model to try out for me. A max iteration number of 10, 30, and 50, and regularization numbers between 0.01 and 5. Experiments have shown that it is useful only to test them with the regularization numbers being exponential to each other. Next, I made sure the prediction label will be named as prediction, and passed in the ground truth label as the Survived column in the MulticlassClassificationEvaluator function. And I let the cross validation function do the rest of the job and fit data set with the information above.

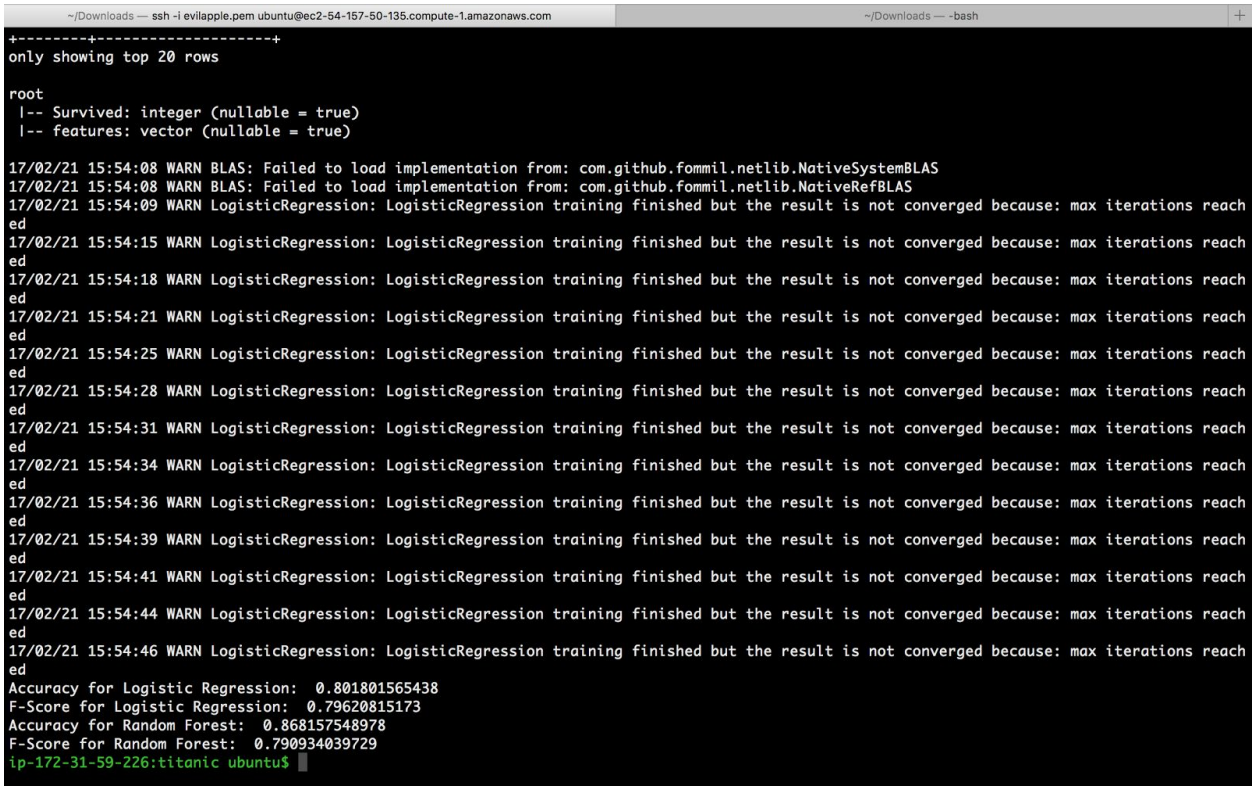
```
result = best_cv_Model.transform(test)
predict_labels = result.select('prediction')
predict_labels = predict_labels.withColumn('prediction',
predict_labels.prediction.cast('int'))
submit = predict_labels.withColumn('PassengerId',
F.monotonically_increasing_id()+892)
submit = submit.select('PassengerId', 'prediction')
submit = submit.withColumnRenamed('prediction', 'Survived')
submit.show()
submit.write.csv('submission.csv', mode='overwrite', header=True)
```

Since the model the cross validation function returns is the best function, I then used it to predict with the test set, and created new a submission file with the results.

Tier 2 Requirements

After implementing the models and doing the predictions, I began to get the application running on an AWS EC2 instance. During the time when I tried to set up Spark on my local machine, there were a lot of trial and error and looking up how to set up the path and environment. This time, it was much faster and took less than half an hour to set up all the programs needed.

I first created an EC2 instance, downloaded the keypair, and ssh-ed to the ubuntu instance from the directory where my keypair was stored. I then installed python, pip tool, java, and used the wget and tar command to fetch and decompress the Spark directory, and put it under my local/bin/directory. In order for the machine to know their location, I had to export their path to the .bash_profile in the home directory, apply the language setting to all, and then source the .bash_profile file for it to take effect. Very soon, I had python and pyspark running on AWS. I then used the scp command to transfer the python program to the aws instance. Here is a screenshot of it with the result of my logistic regression and random forest model.



It is certainly a resource consuming task to run on the local machine. When moving the computation to aws, it is much easier for the local host, the EC2 instance also handled it as speedly as the local machine. I have tried out Microsoft Azure with their Machine Learning tools using Python in the past, but I dont find the online graphic user interface very handy. Google Cloud has similar service and API for machine learning as well. IBM Watson also have their solution. Generally speaking, IBM Watson has better performance than Google Cloud in terms of speed in my experience.

Tier 3 Requirements

I find this assignment to be a great chance to learn the idea and process of working with data sets with Spark and Python. The process of defining functions, building models and creating data frames (or as it would be considered vectors in NumPy) is very different from that of sk-learn, NumPy and Pandas, it is quite hard to pick up the concept, syntax and logic behind the package, and I assume RDD would be a little more complicated than data frames to work with, but functions of spark like Pipeline, Cross Validation, integration with SQL and Machine Learning language is extremely convenient and human-friendly. It is incredibly useful to utilize the Pipeline function to create a high-level pipeline to process, build and run the model. I finally

realized why it is an essential tool for data scientist who deal with much larger data sets to be familiar with Spark.

I did not have enough time to implement the program with Scala, but I would like to do it next time. In the future, I would hope to work with larger data set with data that are harder to estimate the legitimacy and usefulness of, and deploy it using the AWS EMR service. For larger data sets, I would also implement the models in different files, as well as implement a wider range of parameters during cross validation, and a more comprehensive data preprocessing procedure to deal with the bigger complexibility (automatically instead of manually detect whether missing data exists, whether the data range seem valid, corrupted, and optional normalization).