ASSIGNMENT 1

White Wine Case Study

Classification & Regression Algorithms

Using Python, PySpark, Scala

Report Prepared by

Anirudha Deepak Bedre

Avikal Chhetri

Introduction

The following report consists analyzing and predicting the quality of white vinho verde wine using Python, PySpark and Scala.

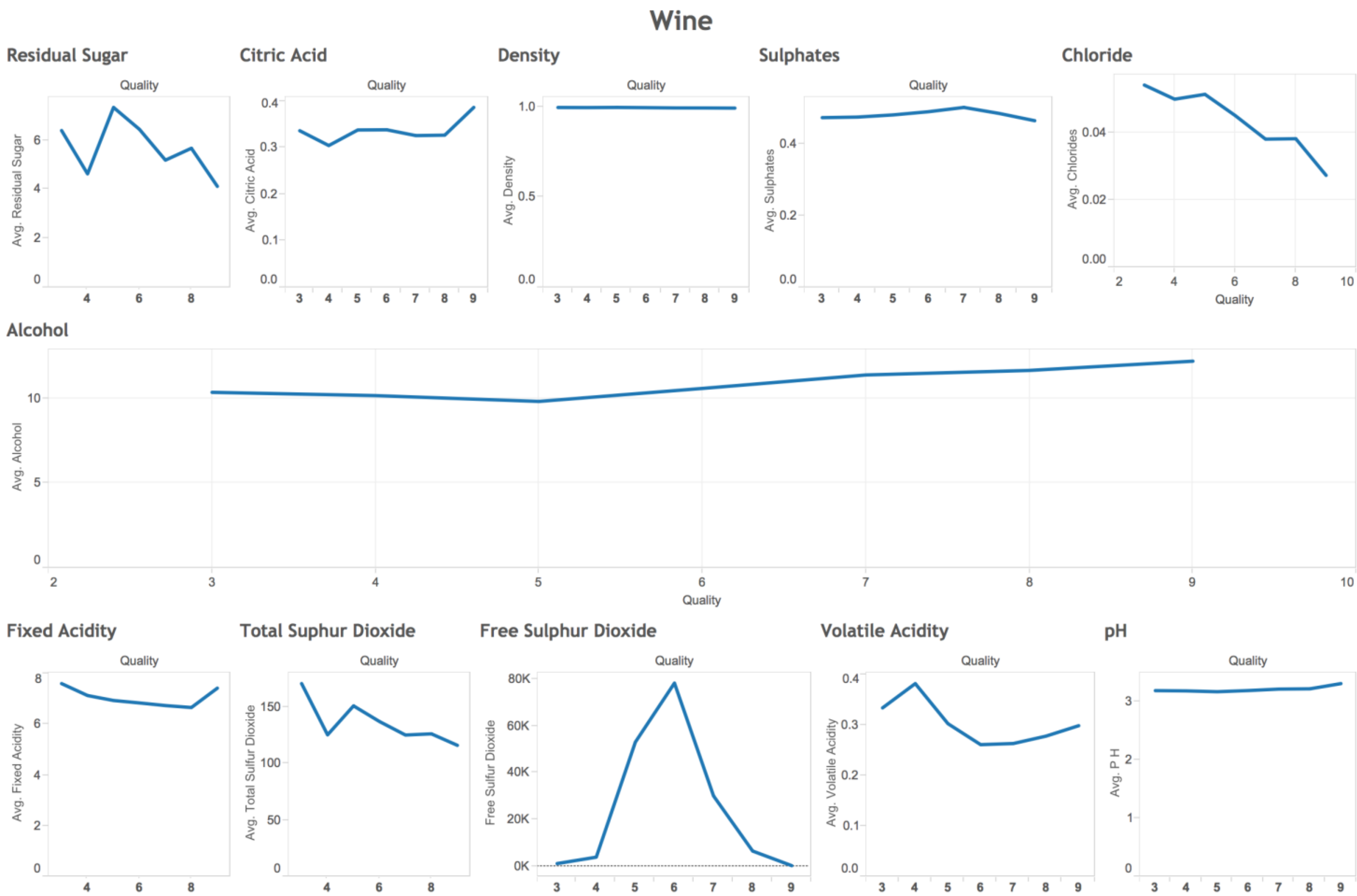
Data Cleansing

The data in the dataset was complete and had no missing values.

Data Transformation

For classification, the values target output column ‘Quality’ were changed to high/low (for python) and 1/0 (for PySpark and Scala). Values >=7 are considered ‘high’ and <7 were taken as ’low’. The data was also normalized because of varying dimensions of the features.

Exploratory Data Analysis



The dashboard represents a comparison between 11 variable for ‘white vinho verde wine samples‘ data from Portugal originally created by Paulo Cortez (Univ. Minho), Antonio Cerdeira, Fernando Almeida, Telmo Matos and Jose Reis. The physicochemical variables are compared with the quality of alcohol (having range 3-9) and the above dashboard was created.

The findings of the comparison are as follows:

Citric Acid, Density, Sulphates, pH, Fixed Acidity, Total Sulphur Dioxide and Alcohol level do not affect the data set profoundly. Their values remain constant without many deviations. Free Sulphur Dioxide follows a bell curve pattern thereby making it an important factor. Volatile Acidity gradually decreases and then increase with increase in Quality value. Residual Sugar values are erratic and show spikes at multiple occasions. Chloride values are generally seen to be decreasing.
The ‘Density’ from our analysis does not seem to have a huge impact of the quality of the wine

and hence we shall be dropping this feature.

Disclaimer!

The data set has been split into 70-30 ratio for analysis. For binary classification we have assumed values above quality level 7 to be high and below 7 as low. High/low to be represented as (1/0) respectively. Random seeds were set as '12345' across all algorithms.

Python using Scipy					
Type	Models	Train Data Regularization (L0,L1,L2) Accuracy		Test Data Regularization (L0,L1,L2) Accuracy	
Classification	sklearn.linear_model.SGDClassifier (SVM => Hinge loss)	L0 = Not Applicable L1 = 0.785 L2 = 0.782		L0 = Not Applicable L1 = 0.790 L2 =0.785	
	sklearn.linear_model.SGDClassifier (Logarithmic regression => log loss)	L0 = Not Applicable L1 = 0.776 L2 =0.781		L0 = Not Applicable L1 = 0.784 L2 = 0.785	
	sklearn.linear_model.LogisticRegression (LBFGS version)	L0 = Not Applicable L1 = Not Applicable L2 = 0.781		L0 = Not Applicable L1 = Not Applicable L2 = 0.785	
Regression		**Mean Squared Error**	**Mean Absolute Error**	**Mean Squared Error**	**Mean Absolute Error**
	sklearn.linear_model.LinearRegression	L0 = 0.584 L1 = Not Applicable L2 = Not Applicable	L0 = 0.599 L1 = Not Applicable L2 = Not Applicable	L0 = 0.603 L1 = Not Applicable L2 = Not Applicable	L0 = 0.609 L1 = Not Applicable L2 = Not Applicable
	sklearn.linear_model.SGDRegressor	L0 = Not Applicable L1 = 0.7697 L2 = 0.769	L0 = Not Applicable L1 = 0.669 L2 = 0.669	L0 = Not Applicable L1 = 0.770 L2 = 0.770	L0 = Not Applicable L1 = 0.677 L2 = 0.677
	sklearn.linear_model.Ridge	L0 = 0.705 L1 = Not Applicable L2 = Not Applicable	L0 = 0.642 L1 = Not Applicable L2 = Not Applicable	L0 = 0.719 L1 = Not Applicable L2 = Not Applicable	L0 = 0.659 L1 = Not Applicable L2 = Not Applicable
	sklearn.linear_model.Lasso	L0 = 0.784 L1 = Not Applicable L2 = Not Applicable	L0 = 0.665 L1 = Not Applicable L2 = Not Applicable	L0 = 0.783 L1 = Not Applicable L2 = Not Applicable	L0 = 0.673 L1 = Not Applicable L2 = Not Applicable

PySpark Analysis			
Type	Models	Test Data Regularization (L0,L1,L2) Accuracy	
Classification	SVMWithSGD	**L0** = 0.782 **L1** = 0.782 **L2** = 0.783	
	LogisticRegressionWithLBFGS	**L0** = 0.784 **L1** = 0.784 **L2** = 0.787	
	LogisticRegressionWithSGD	**L0** = 0.782 **L1** = 0.782 **L2** = 0.782	
Regression		**Mean Squared Error**	**Mean Squared Error**
	LinearRegressionWithSGD	L0 = 2.984 L1 = Not Applicable L2 = Not Applicable	L0 = 3.140 L1 = Not Applicable L2 = Not Applicable
	RidgeRegressionWithSGD	L0 = Not Applicable L1 = Not Applicable L2 = 3.184	L0 = Not Applicable L1 = Not Applicable L2 = 3.163
	LassoWithSGD	L0 = Not Applicable L1 = 2.950 L2 = Not Applicable	L0 = Not Applicable L1 = 3.223 L2 = Not Applicable

Scala Analysis			
Type	Models	Train Data Regularization (L0,L1,L2)	Test Data Regularization (L0,L1,L2)
Classification	SVMWithSGD	**L0 =** 0.782 **L1 =** 0.782	**L2 =** 0.783
	LogisticRegressionWithLBFGS	**L0 =** 0.784 **L1 =** 0.784	**L2 =** 0.787
	LogisticRegressionWithSGD	**L0 =** 0.782 **L1 =** 0.782	**L2 =** 0.782
Regression		**Mean Squared Error**	**Mean Squared Error**
	LinearRegressionWithSGD	L0 = 3.203 L1 = Not Applicable L2 = Not Applicable	L0 = 3.209 L1 = Not Applicable L2 = Not Applicable
	RidgeRegressionWithSGD	L0 = Not Applicable L1 = Not Applicable L2 = 3.213	L0 = Not Applicable L1 = Not Applicable L2 = 3.189
	LassoWithSGD	L0 = Not Applicable L1 = 3.171 L2 = Not Applicable	L0 = Not Applicable L1 = 3.324 L2 = Not Applicable

Disclaimer!

The data set has been split

Instructions to run the code:

Datasets

Open Dataset folder to : There are different files for Spark and Python as Spark cannot handle categorical variables which area strings and have to be converted to 0,1,…. Select regression and classification .csv files to view the dataset

For Python/PySpark

Open the Code folder to: Select Python folder

Open iPython notebook files to run python code

Select respective folder(PySpark/Python) folder open iPython notebook file to run PySpark code

For Scala

Select Scala folder, open Scala files to run scala code.
We have it as .scala files and not as .jar as we faced issues in them as an application using sbt.

Configurations Used:

The 'number of iterations' used in a majority of the algorithms were set to the default 10(in Python) and 100 (in PySpark and Scala).

The default 'stepSize' used(mostly 1) proved to be too large and so we ended up using a smaller step-size of '0.000469' to give us optimal results

The 'random-state' parameter which is the random seed number given was set to '12345' across all instances of implementation.

The 'intercept' was set to 'True' across all implementations. It was found 'true' by default in scikit's libraries, but the mlib libraries had 'False' by default.

Loss functions Used:

The default loss function used was 'squared-loss' except for instances where 'hinge-loss' and 'log-loss' were made to be used by necessity (based on the algorithm used).

Regularization Used:

A majority of the algorithms used had L2 regularization as default. 'L1' and 'L2' were individually used in specific cases like Lasso regression (L2 regularized) and Ridge regression (L2 regularized) algorithms. We didn't happen to see a huge difference between the L1 and L2 regularizers.

Performance Metrics used:

For classification, we primarily used the accuracy score as our metric. We also had a look at confusion matrix to see how distinctly each category was predicted against the actual values

For regression, the metrics used were Mean Squared Error (MSE) and Mean Absolute Error. We used mean absolute error to measure how close forecasts or predictions are to the eventual outcomes. We used the mean square error to quantify the difference of the quantity being estimated.

Compare and contrast using Just Scipy libraries vs using Apache Spark. How did the results vary? When would you use just Python libraries and when would you use Apache Spark?

Scipy libraries vs using Apache Spark:

After experimenting with both the Scipy Libaries and Apache Spark's Mlib libraries we have learnt the following:

SciPy Libraries	Apache Spark's Mlib
We would prefer to use SciPy's libraries when we don't have a huge amount of data.	We would prefer to use Apache Spark's Mlib libraries when we have a huge amount of data and take advantage of Spark's architecture and the functionality of RDDs
The algorithms are not written with respect to map/reduce concepts.	The algorithms are written with respect to map/reduce concepts and to implement the parallelization efffectively.
The SciPy documentation were found to be much better documented with plenty of examples	The Spark's mlib documentation were not found to be well documented comparatively

For Classification algorithms, the variations in using Scipy libraries and Apache Spark were found to be almost identical upto the second decimal point.
For Regression algorithms, a huge difference of nearly 3 numerical values was noted.

Conclusion

Thus the white wine data sets was analyzed and the quality was predicted using Python, Apache Spark (PySpark and Scala) successfully.