# FINAL MILESTONE REPORT

|  |  |
| --- | --- |
| SUBMITTED BY : | TEAM 3 |
| TEAM MEMBERS : | Priyatha Joji Abraham (010819432), Qiao Liu (010057281),  Shraddha Kabade (012434409), Tina Philip (010019958) |
| SELECTED ML ALGORITHM : | LOGISTIC REGRESSION |
| BONUS FEATURES ATTEMPTED BY : | Priyatha Joji Abraham (010819432), Qiao Liu (010057281),  Shraddha Kabade (012434409), Tina Philip (010019958) |
| SUBMITTED ON : | 05/21/2017 |

# 

# 

# CONTRIBUTION

|  |  |  |  |
| --- | --- | --- | --- |
| No. | What | Description | Who |
| 1 | Data preparation | Validate and filter out data used to feed MLib algorithm using Jupyter Notebook (Python). | Priyatha, Tina, Nicole |
| 2 | Data analysis | Detailed study of all 3 files to pick relevant columns used for training and testing the data. | Priyatha, Tina, Nicole, Shraddha |
| 3 | Test algorithm on prepared data-sets | Implement the algorithm on sample data-set to illustrate how it works. Study how prepared data can be processed and fed to the algorithm. | Shraddha, Nicole |
| 4 | Analyze output of each data-set | Interpret the results and provide detailed explanation | Shraddha, Nicole |
| 5 | Model Building and Training for Bonus parts | Normalize text using Regular expressions, spark tokenizer, stop word removal etc.  Use normalized data to extract features using spark TF-IDF, Word2Vec etc. | Nicole, Shraddha, Priyatha, Tina |
| 6 | Powerpoint Preparation | Prepare ppt slides to present overview of algorithm, data preparation steps and demo code walkthrough. | Nicole, Shraddha, Priyatha, Tina |
| 7 | Report | Prepare detailed report with all required contents. | Priyatha, Tina, Shraddha, Nicole |
| 8 | Powerpoint Presentation | Present the ppt in class. | Nicole, Priyatha, Tina, Shraddha |

[**FINAL MILESTONE REPORT**](#_k4k4z87uzwyw) **1**

[**CONTRIBUTION**](#_ggq2ex2zn8z8) **2**

[**GOAL**](#_u0ssuygpjotm) **5**

[**Part 1**](#_6q9avg2wbjyt) **5**

[**Data preparation**](#_hq9843tye273) **5**

[1.a Validate and clean puppy/trainer pair data out of the 3 given files. Note some puppy-trainer pairings may show up more than once or in more than one file. Explain how you pre-process the data to get the most reliable data to start with.](#_m9fo2ppwdghi) 5

[1.b Prepare 5 or more sets of data with varying number of features related to puppy info. List the features of these 5 sets in a table with feature description on top row and set number on left most column as follows. (??%)](#_omy3sgs6jw6) 17

[1.c For each of the 5 features sets decided in 1.b, come up with a csv or Excel file with puppy info and affiliated training outcome (Success/Failure) based on your 1.a and 1.b effort. How many rows of reliable data are in these file?](#_8jknfki7wujs) 19

[**PART 2  
Machine Learning Algorithm Implementation**](#_9ts6u7yzd088) **21**

[2.a ML algorithm Overview](#_stwlzph9g0fu) 21

[2.b Spark ML Library](#_90vch8a6kvko) 24

[2.c ML algorithm Implementation](#_n0i9clwlhfph) 25

[2.d Prediction on randomly split and non-randomly split data](#_1ikeghd1ex3m) 28

[2.e Optimal number of features](#_6sme9p4dpiro) 32

[2.f List of Algorithms and Parameters used.](#_8egrym9phz9c) 33

[setNumFeatures](#_l55wh5f53xw7) 38

[**PART 3  
Prediction model based on trainer info**](#_6oq7bzd3563t) **39**

[3.a 1) Data Preparation](#_hklu062y6r1k) 39

[3.a 2) Text Normalization and Feature Transformation](#_r8ke7hdi3r24) 43

[3.b Feature Extraction](#_d0yjjbiq31s) 48

[3.c Prediction Rate](#_osmn2lb7fdh0) 53

[**BONUS Features**](#_uzxwa9lsp2u1) **57**

[B.1 Prediction using both Trainer and Puppy](#_ba3qjo6gnpe0) 57

[B.2 Prediction with Random and Fixed Split](#_35dx17bqnoe7) 58

[B.3 Prediction Rate with Puppy Info text/sentence based](#_pqv9w9si7tl4) 60

[B.4 Prediction Rate with Puppy Info numeric and text/sentence based](#_fbceco5mukrw) 65

[B.5 Prediction Rate with Puppy and Trainer : numeric and text/sentence based](#_6uuoqlfctpjf) 67

[**REFERENCE**](#_tgvk58nsixw1) **70**

# 

# GOAL

The main goal of this project is to predict the training outcome with some given puppy and trainer info. Use your selected ML algorithm and 80% of the data to train the model, followed by testing it prediction capability with the other 20%.

# Part 1

# Data preparation

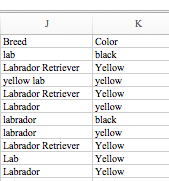
*1) Download and study the* [*data*](https://sjsu.instructure.com/courses/1229294/files/46919102/download)*. Some human errors are always part of the real world data and these files are without exception.The column on success/failure outcome in puppytraineroutcome file is dog\_SubStatusCode.dog\_SubStatusCode as 23 or 25 or 26 or 27 or 55 or 98 or 99 or 121 or 169 means successful training outcome. The rest of the code are failure outcomes.*

#### 1.a Validate and clean puppy/trainer pair data out of the 3 given files. Note some puppy-trainer pairings may show up more than once or in more than one file. Explain how you pre-process the data to get the most reliable data to start with.

**Preparation of Datasets:***Refer folder 1a*  
  
From the last milestone, we realised that selecting the features requires that we use to train the machine learning models have a huge influence on the performance that we can achieve. To achieve that we must start with preprocessing of given data. We conducted the data preprocessing through three steps : Formatting, Cleaning and   
  
**Step 1: Formatting xlsx files to csv.**  
  
The given dataset ( Refer folder: 1a/originalDataset) PuppyInfo, TrainerInfo and PuppyTrainerOutcome are in xlsx format. But we are dealing with csv files ,so as a first step we converted it into csv manually ( Refer folder: 1a/formattedDataset).

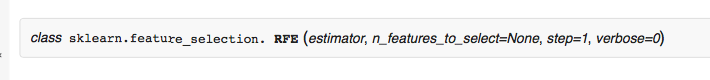
**Step 2: Cleaning the dataset**

* Software : Python version 2.7
* Platform : Mac OS

*Refer folder 1a/formattedDataset*We found that there are lot of invalid data entries and missing values in the dataset. So we must remove or fix missing data to ensure data consistency across all files. For instance, columns like Breed , Color in 1a/*formattedDataset*/PuppyInfo.csv has inconsistent and misspelled values as shown in fig.1.and some other columns like Sex in 1a/*formattedDataset*/PuppyInfo.csv contains missing or blank values as shown in fig.2.  
   
Fig.1. Misspelled values in column Breed and Color  
  
  
Screen Shot 2017-05-21 at 4.02.29 PM.png  
Fig. 2. Blank values in column Sex  
  
Also, we found redundant values in the dataset.  
But Logistic Regression takes only numeric or categorical values.So to ensure data consistency, we validated the inconsistent columns and selected features which are found as important to us because irrelevant features in the dataset can decrease the accuracy of many models, especially linear algorithms like linear and logistic regression. The 3 benefits of performing feature selection before modeling the data is: [4]

1. Reduces Overfitting: Less redundant data means less opportunity to make decisions based on noise.
2. Improves Accuracy: Less misleading data means modeling accuracy improves.
3. Reduces Training Time: Less data means that algorithms train faster.

Here, we selected the important features mainly in two ways:

1. Domain Knowledge :  
   We selected few columns using human intuition and prepared a 9 feature dataset called **Dataset A:** taken from Milestone 1 and we created a new 22 feature comprehensive features dataset (**Dataset B**) and a19 feature dataset  **(Dataset F)** by human judgement.
2. Recursive Feature Elimination (RFE):  
   This is an automatic feature ranking process by scikit learn. RFE works by recursively removing attributes and building a model on those attributes that remain. It uses the model accuracy to identify which attributes (and combination of attributes) contribute the most to predicting the target attribute.   
   During our research to find a feature selection method we found that RFE is commonly used for feature selection in Logistic Regression and gives best feature set. Also, it is said to provide improved accuracy. Therefore we wanted to try this method. From MileStone 2,we also realized that our 10 size feature set generated solely through RFE performs better than 15 size feature set generated by adding columns through human intelligence.  
     
   

**n\_features\_to\_select:** no. of features to select **step :** number of features to remove at each iteration  
 **Verbose :** Controls verbosity of output.

Given an external estimator that assigns weights to features (e.g., the coefficients of a linear model), the goal of recursive feature elimination (RFE) is to select features by recursively considering smaller and smaller sets of features.   
First, the estimator is trained on the initial set of features and weights are assigned to each one of them. Then, features whose absolute weights are the smallest are pruned from the current set features. That procedure is recursively repeated on the pruned set until the desired number of features to select is eventually reached.   
  
The example below uses RFE with the logistic regression algorithm to select the top 3 features:  
RFE(LogisticRegression(),[http://scikit-learn.org/stable/modules/generated/sklearn.feature\_selection.RFE.ht](http://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.RFE.htm). However, to get a good feature set,we need to clean the dataset.

For feature selection, we approached two ways as said before, Domain knowledge and RFE (Recursive Feature Elimination):  
**1) Domain Knowledge :**  
We selected few columns using human intuition and prepared a 9 feature dataset called **Dataset A:** taken from Milestone 1 and we created a new 22 feature comprehensive features dataset (**Dataset B**) and a19 feature dataset  **(Dataset F)**

2) Recursive Feature Elimination (RFE):  
 This is an automatic feature ranking process by scikit learn.

Through RFE we prepared two data sets:  
 - 5 column feature set (**Dataset C**)  
- 10 column feature set (**Dataset D)**

The next feature set was prepared through a combination of RFE and human intuition, ie, Selecting 10 features that are ranked high by RFE process and 5 features which we think would be relevant were added into it. Hence it had 15 features **( Dataset E).**

The final feature set,with 19 features, is prepared through human intelligence **(Dataset F)**

**Thus we have selected 5 sets of dataset:  
Dataset A - 9 features ( from Milestone 1)**

**Dataset B - 22 features ( human intelligence)**

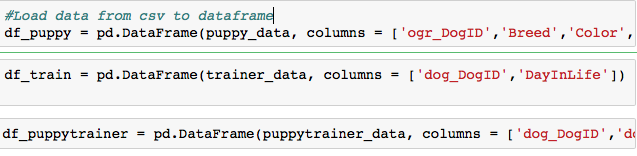
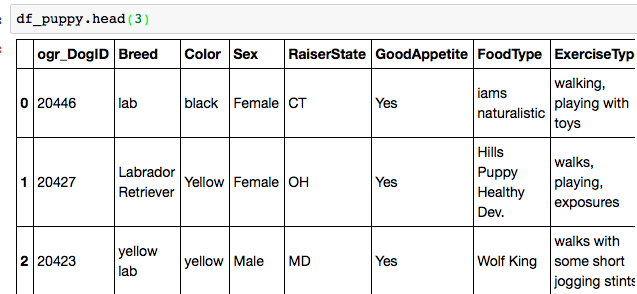
**Dataset C - 5 features ( RFE )**

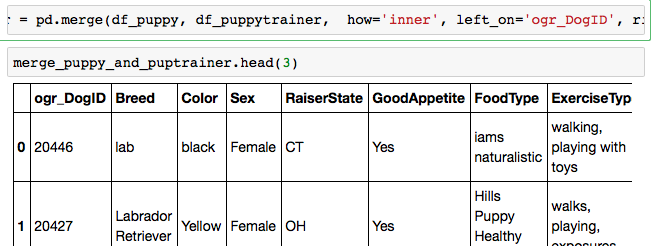
**Dataset D - 10 features(RFE)  
Dataset E - 15 features ( human intelligence + RFE )**

**Dataset F - 19 features (human intelligence)**  
The detailed account of **Dataset - Description** is given in section **1b.**

The step-by-step process of data preparation is explained in **1a**:

* Run **1a /csvfiles/clean\_merge.py** to clean the data from puppy and trainer files
* Run below command in cmd :   
   python clean\_merge.py  
    
  Below are the steps to clean the dataset,  
    
  **Step i:** Load the data from puppy, and trainer files and select the features which will be used in future using human intelligence. Remove the invalidate data using the following steps.

 **Step ii. Load columns**  
-Select desired columns from PuppyInfo into df\_puppy dataframe  
-Select desired columns from TrainerInfo.csv into df\_train dataframe  
-Select desired columns from PuppyTrainerOutcome.csv into df\_puppytrainer dataframe  
  
  
Displaying values from df\_puppy.  


**Step 3: Merge files on dogID column.**  
Merge df\_puppy that contains PuppyInfo.csv values and df\_puppytrainer that contains statuscode from PuppyTrainerOutcomes.csv values on dog id columns.  
We choose DogID column because it is appearing in all 3 files and they appears to be same.  
  


**Step 4: Ensure data consistency**

Logistic Regression takes only numeric or categorical values.So to ensure data consistency, we must validate the columns: a) Breed, b) Color, c) Sex,d) RaiserState, e) GoodAppetite, f) StatusCode.

a) Breed:  
  
All the inconsistent values in Breed column has been validated as shown in Fig. 6 below:  
1 - Labrador Retriever  
2 - German Shepherd

3 - Labrador/Golden Cross  
4 - Golden Retriever

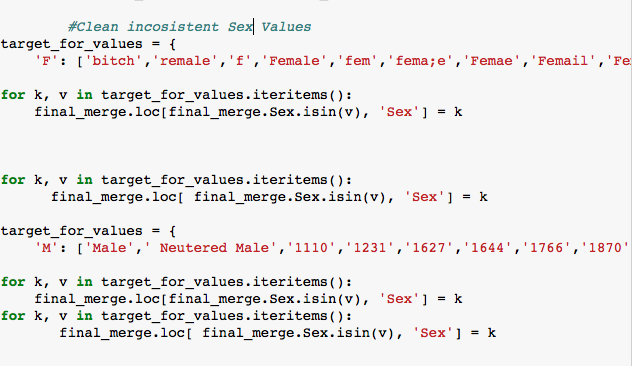


Fig. 6. All inconsistent values in Breed is categorized into 4.

b) RaiserState  
  
 RaiserStates are validated to their respective state codes as shown in Fig 7.

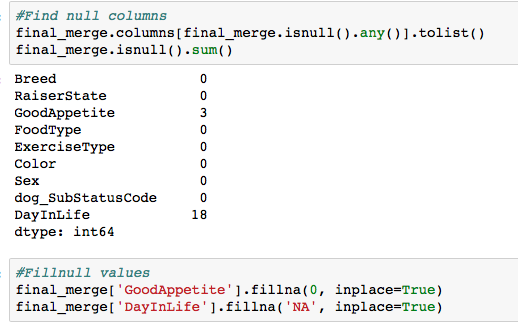


Fig. 7. Raiserstates are made consistent

c) Sex:  
  
Validate values in Sex  
  


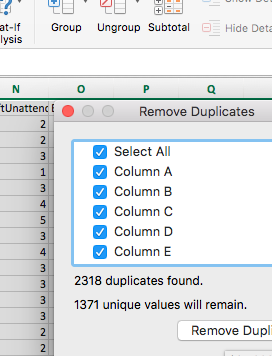
d) Color:  
  
Change color values and make them consistent.  
  


**Fill Null Values:**  
In this dataset there are null values in GoodAppetite and DayInLife column. As we cannot deal with blank values, we decided to fill NAN with 0’s for GoodAppettite using ***fillna()*** because majority are 1 in GoodAppettite column and blanks in DayinLife is filled as ‘NA’

  
  
Convert the final cleaned dataset to ***Cleaned\_merge\_new.csv***

**Step 5:Remove duplicates**

The final file is at 1a/csvfiles/***Cleaned\_merge\_new***.csv

Remove the duplicates manually and we found 1371 unique values in this dataset and has 47 columns.  


After removing duplicates, 2111 unique values remain. This file is named as

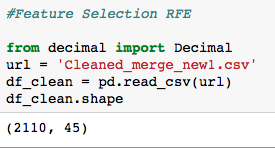
**Cleaned\_merge\_new1.csv  
  
Step 5: Recursive Feature Elimination**

Refer: **1a/csvfiles/ RFE\_5.py**

**How to run :** python RFE\_5.py

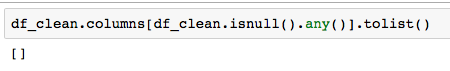
Model accuracy is used by RFE to identify the attributes that contribute the most to predicting the target attribute [4].

**a)** Import **Cleaned\_merge\_new1.csv** into our data frame, df\_clean. There are 2110 data records excluding header) and 45 columns.



Fg. 19. Df\_clean now contains Cleaned\_merge\_new1.csv

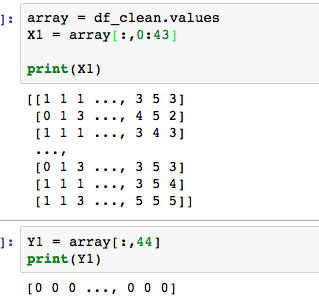
**b)** Ensuring there are no more blank values



**c)** Load all the relevant column names to “name” 

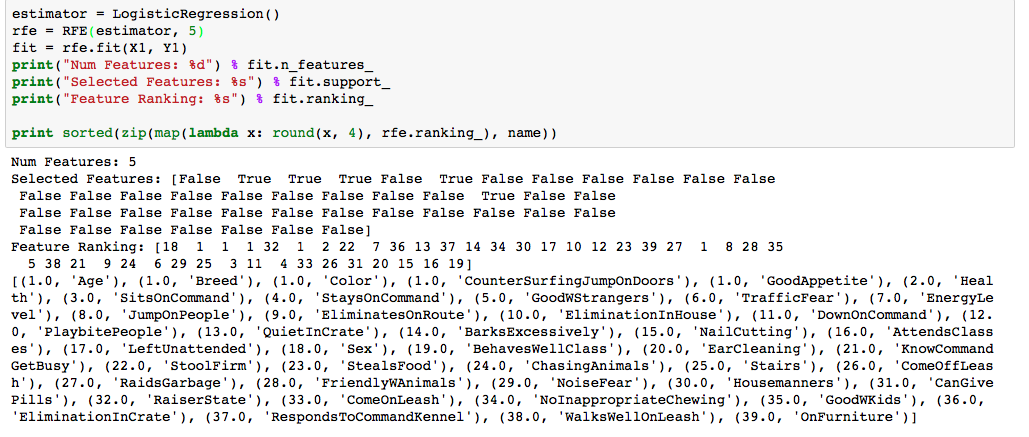
**d)** Convert dataframe to arrays

X1 is the training input values  
 Y1 is the target values



**e)** RFE for 5 most important features - **Dataset C**

LogisticRegression is used as the estimator and we pass estimator and no. of relevant features to retreive as 5 to RFE ().Fit function fits the RFE model and then the underlying estimator on the selected features.



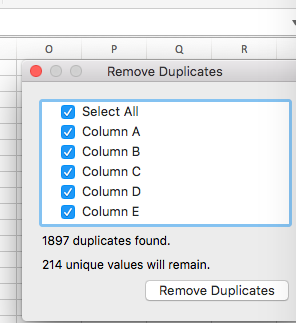
From this result, we can realize that the features with value 1.0 are the 5 most important features.

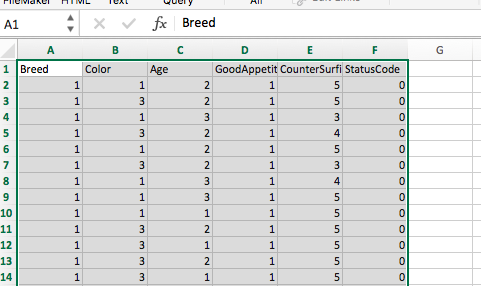
Thus the 5 most important features that forms **Dataset C** as per RFE are:  
**Age, Breed,Color, CounterSurfingJumpOnDoors, GoodAppetite**

Now delete all the columns from **Cleaned\_merge\_new1.csv** manually except the above listed ones. The new csv sheet is named **RFE\_5RelevantColWithDuplicates.csv**

Manually removed the duplicates from **RFE\_5RelevantColWithDuplicates.csv** ( 214 unique values including header)).  
The new sheet without duplicates is named

**RFE\_5RelevantCol\_With\_Header\_Unique.csv**

:



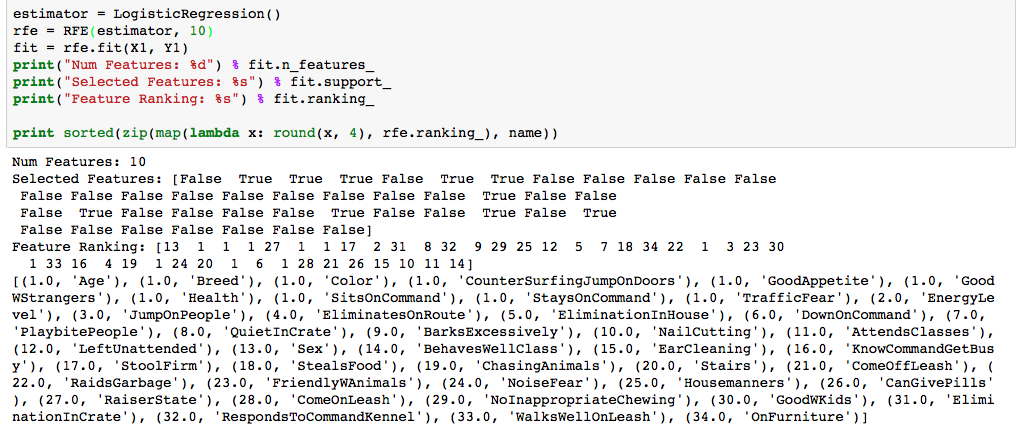
f) RFE - 10 most important features - **Dataset D**

Refer: **1a/csvfiles/ RFE\_10.py**

**How to run :** python RFE\_10.py

Repeat steps a), b), c), d)

LogisticRegression is used as the estimator and we pass estimator and no. of features to retreive as 10 to RFE ().Fit function fits the RFE model and then the underlying estimator on the selected features.X1 is the training input values and Y1 is target values.

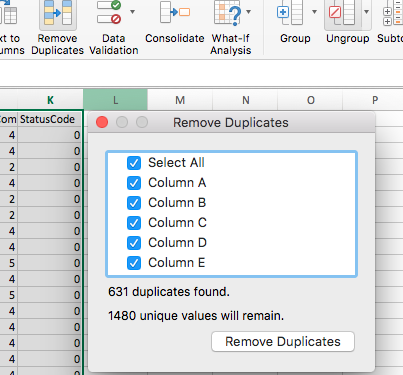


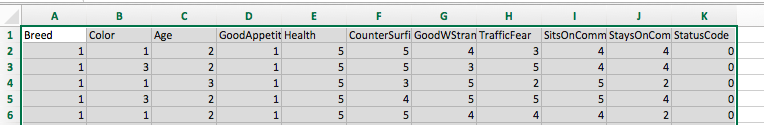
Similarly we found the 10 most important features that forms **Dataset D** as per RFE are:  
**Age, Breed, Color, CounterSurfingJumpOnDoors, GoodAppetite, GoodWStrangers,**

**Health, SitsOnCommand, StaysOnCommand, TrafficFear**

Now delete all the columns from **1a/csvfiles/Cleaned\_merge\_new1.csv** manually except the above listed ones. The new csv sheet is located at **1a/csvfiles/ RFE\_10RelevantColWithDuplicates.csv**

Manually removed the duplicates from **RFE\_10RelevantColWithDuplicates.csv** ( 1480 unique values including header)). The new sheet without duplicates is located at **1a/csvfiles/ RFE\_10RelevantCol\_With\_Header\_Unique.csv**





**Preparation of Dataset E :**Added 5 more feature that we think could be important to 10-feature set by human judgement.

**Preparation of Dataset F :**

Selected 19 feature set through human judgement.

#### 1.b Prepare 5 or more sets of data with varying number of features related to puppy info. List the features of these 5 sets in a table with feature description on top row and set number on left most column as follows. (??%)

Feature set :

**Dataset A: 9 features: From Milestone1**

Breed, Age, Sex, Health, ResponsToCommandKennel, Traffic Fear, GoodWStrangers, StayOnCommand, BehavesWellClass

**Dataset B: 22 features(2108 unique samples): More comprehensive features by human judgement.**

Sex, Breed, Color, Age, GoodAppetite, Health, StoolFirm, EnergyLevel, EliminationInCrate, QuietInCrate, RespondsToCommandKennel, BarksExcessively, RaidsGarbage, CounterSurfingJumpOnDoors, JumpOnPeople, GoodWStrangers, WalksWellOnLeash, TrafficFear, NoiseFear, SitsOnCommand, StaysOnCommand, BehavesWellClass, StatusCode

**Dataset C: 5 features (213 unique samples): The recommended features by RFE**

Breed, Color, Age, GoodAppetite, CounterSurfingJumpOnDoors

**Dataset D: 10 features(1479 unique samples): The recommended features by RFE**

Breed, Color, Age, GoodAppetite, CounterSurfingJumpOnDoors,

Health, GoodWStrangers, Traffic Fear, SitsOnCommand, StayOnCommand

**Dataset E: 15 features (1991 unique samples): Add some feature by human judgements to 10-feature set**

Breed, Color, Age, GoodAppetite, Health, CounterSurfingJumpOnDoors, GoodWStrangers, Traffic Fear, SitsOnCommand, StayOnCommand,

EliminationInCrate, FriendlyWAnimals,GoodWKids, BarksExcessively, WalksWellOnLeash

**Dataset F :19 features(2059 unique samples): More comprehensive features by human judgement.**

Breed, Sex, Health, EliminationInCrate, QuietInCrate, RespondsToCommandKennel, BarksExcessively, RaidsGarbage, CounterSurfingJumpOnDoors, JumpOnPeople, FriendlyWAnimals, GoodWKids, GoodWStrangers, TrafficFear, NoiseFear, StaysOnCommand, BehavesWellClass, StoolFirm, EnergyLevel

The 5 data sets prepared with varying number of features are listed as a table. Below the screenshot is attached. The file for which can be seen in folder 1b named datasets.xlsx

# 

*Explain why these selected feature sizes will be effective for training and subsequent prediction in your machine learning process.*

We selected different sizes of feature set to experiment and evaluate the performance of our ML Logistic Regression algorithm respected to different feature size.

● We knew, a small feature size can suffer under-fitting and a large feature-size may suffer overfitting.

● This was comprehended through our trial runs in part 2. We realized that feature size matters and small no: of feature set can give you lower prediction rate. For instance from milestone2, we realized that 5-feature set gives you less prediction rate compared to 10/ 19 size feature-set.

● Moreover, we only considered up to 22 features because training time matters and if we take a larger set our algorithm might take lot of time to process the irrelevant features that do not add any value to our prediction.

# 

#### 1.c For each of the 5 features sets decided in 1.b, come up with a csv or Excel file with puppy info and affiliated training outcome (Success/Failure) based on your 1.a and 1.b effort. How many rows of reliable data are in these file?

Refer 1c/dataset

Dataset A: 9 features: (**Intuitive\_9.csv**)  
Reliable data : (3459)

Dataset B: 22 features(Reliable : 2108 unique samples)   
(**Intuitive\_22.csv**)

Dataset C: 5 features (213 unique samples): The recommended features by RFE (**RFE\_5RelevantCol\_With\_Header\_Unique.csv**)

Dataset D: 10 features(1479 unique samples): The recommended features by RFE

(**RFE\_10RelevantCol\_With\_Header\_Unique.csv)**

Dataset E: 15 features (1991 unique samples): Add some feature by human judgements to 10-feature set (**RFE\_15RelevantCol.csv**)

Dataset F: 19 features (2059 unique samples ) (**Intuitive-19.csv**)

Refer 1.b for details feature column description.

Explain any data inconsistency you may have and how your chosen machine learning algorithm may cope with such.

We already ensured data consistency by 1a. Steps. As Logistic Regression takes only numeric or categorical values.So to ensure data consistency, we validate such that only numeric values have been is there. To do that we converted all text values or categorical values into numerical values to vectorize those values. We used stringIndexer to convert categorical values to numerical . We have also used textnormalization function to remove punctuations , diacritical marks, extra spaces from text values or sentence values.  
  
One issue we faced in bonus part is even if we select values using dataframe df, since there are commas in sentences they are considered as seperate columns. We overcame the problem by keeping statuscode as first col, then we kept in the order - numerical columns-> text columns->sentence columns.

# 

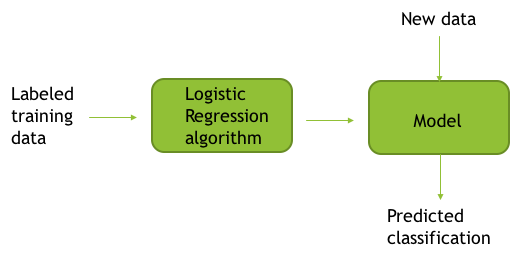
# PART 2 Machine Learning Algorithm Implementation

#### 2.a ML algorithm Overview

*Explain how your selected algorithm can be used to classify our problem. Comment on any possible drawback in using your selected algorithm for our prediction problem.*

**Regression** is a supervised machine learning algorithm where you take a known set of input data and its predicted output data, and trains a model to generate reasonable predictions for the response to new data.These predictions are discrete values in a classification problem and continuous values in regression problems.

**Logistic regression** is a statistical method for analyzing a dataset in which there are independent variables that determine an outcome. The outcome is measured with discrete variable.

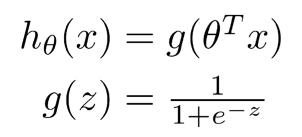


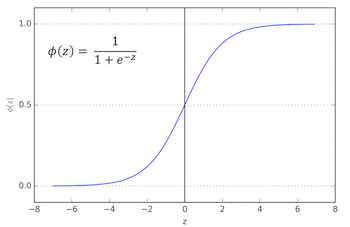
We present the input data, algorithm and output data as the formula below.

**y = h**θ**(x)**

x is the feature vector. 𝜽 is the parameter vector the algorithm will learn. h is the hypothesis. y is the classified category. Here y is discrete value. When y can take only two values, "0" and "1" , it is binomial logistic regression. When y can take more than two values, it is multinomial logistic regression.

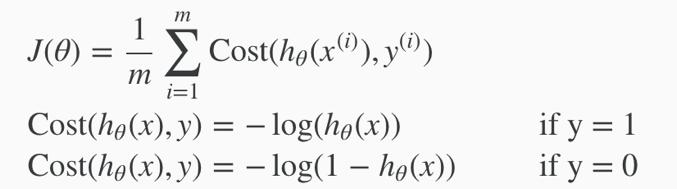
The **hypothesis** takes a sigmoid function g(z). In logistic regression, z = 𝞱Tx.



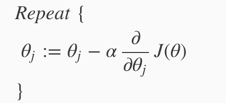


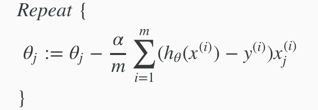
hθ(x) equals a value between 0 and 1. When y (= hθ(x)) > **decision boundary**, y = 1, otherwise, y = 0. We can interpret y as the the probability that hθ(x) = 1, given x and 𝞱

To get good parameter 𝞱, we keep updating 𝞱 to minimize the cost. The **cost function** of logistic regression takes an average difference (actually a fancier version of an average) of all the results of the hypothesis with inputs from x's and the actual output y's.



One way to optimize 𝞱 is gradient descent. each parameter 𝞱j is updated by deducting a ratio of the partial derivative. 𝜶 is know as **learning rate**. Different learning rate will influence the converging behavior of the cost function.





In practice, gradient descent is too slow, because it takes sum of all training data, which is huge, for updating each parameter in each iteration. Stochastic gradient descent is more practical. And there are some more advanced algorithm, such as BFGS.

**possible drawback:**

The pros and cons of learning algorithm depends on the problem and the feature of the data. Generally, the most significant **pros** of logistic regression is it is easy to implement and runs efficiently. However, the **cons** is the boundary of logistic function is linear. The boundary is a line, a plane, or a hyper-plane for higher dimension. We can use techniques, such as **scaling the variable,** to achieve a linear boundary for some problems. Or, we can construct some polynomial parameters to achieve non-linear boundary, which is not usually practiced, because we can use neural network rather than do so.

The possible drawback is the model can not have good performance if the boundary is not linear.

**Approach of Project** :

In this case, we need to predict the training outcome, given puppy and trainer info. We have a set of data on puppy and trainer info.Also, we know what is the output for a puppy - trainer which is dog\_SubStatusCode for selected features.Thus we feed the necessary features and labeled output into a model which is trained to predict whether a new puppy and trainer training outcome would be success or failure. Since there are two kinds of values in StatusCode column we use binary classification.

**y = h**θ**(x)**

* x: the feature vector. feature such as dog breed
* 𝞱: the parameter the algorithm need learn
* y: 0/1 indicates the dog-trainer outcome failure/success

#### 

#### 

#### 2.b Spark ML Library

*Describe which Spark ML library can be used for prediction purpose with your selected algorithm.*

**We have 2 options to train the model using spark MLib library based on RDD** :

* LogisticRegressionModel
* LogisticRegressionWithLBFGS

LogisticRegressionWithLBFGS is preferred for better results. We will practice and analyze on different algorithms with different learning rate, regularization parameter, etc along with different features to achieve different accuracy.

**We have another options to train the model using spark ML library based on Dataframe:**

* LogisticRegression

This algorithm provides many parameters which are listed in **2.f) part**

Spark ML library provide [pipeline](https://spark.apache.org/docs/2.1.0/ml-pipeline.html) algorithm which can be used to perform feature transformation and extraction in sequence and provide to learning algorithm

We have used TF-IDF, Word2Vec, Tokenizer, StopWordRemover, StringIndexer algorithms for sentence based data.

Documentation **:** [link](https://spark.apache.org/docs/2.1.0/api/scala/index.html#org.apache.spark.ml.classification.LogisticRegression)

#### 

#### 

#### 2.c ML algorithm Implementation

*Apply Spark MLLib algorithm with puppy info alone on each set of data prepared in (1.b). List the prediction outcome vs. feature size in a table like the one shown below.*

***Software*** *: Scala (2.11.8),* ***Tool*** *: Spark (2.1.0)*

***Platform*** *used: Ubuntu*

**Executing the code (LBFGS) :**

* Run **2c/*PuppyTrainerLBFGS.scala*** file from spark shell.
* This code executes for dataSet with 5 features. For other dataSets (9, 10, 15, 19, 22) please replace 5 with these values respectively.



Dataset used to run the code : ***2c/datasets/dataSet5.csv***

We tested the data set using ***LogisticRegressionWithLBFGS()*** with random split as well as fixed split as below.

With random split:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Feature Set** | **5-feature set** | **9-feature set** | **10-feature set** | **15-feature set** | **19-feature set** | **22-feature set** |
| Training data prediction rate | 57.99% | 61.40% | 67.09% | 68.22% | 66.71% | 67.78% |
| Testing data prediction rate | 63.64% | 68.93% | **63.89%** | 61.80% | **62.72%** | 61.31% |

*Offer explanation on your observation.*

The **5-feature** set has only 213 samples. Though, the prediction is not relatively bad, the rate is not reliable. The size of feature set may have influenced the outcome.

The **9-feature** set use the data in milestone 1 which might not be as accurate as the dataset in milestone 2, which we regenerate based on more understanding and refinement of the data. We don’t take this outcome to consideration in milestone 2.

***Note: Due to the same reason, in future analysis, we will focus on other 5 data-set.***

**10-feature** set has the highest prediction rate, which give us the sense that those 10 features might be good features.

**15-feature** set generated by adding 5 more features by human judgement. The prediction rate decreases. The reason might be that some of the 5 newly added features are not good features.

**19-feature** set generated by human judgment, including many features not in the 10-feature set, however, it still has good prediction rate. There are some good features in the 19-feature set, but not in 10-feature set.

**22-feature** set suffers overfitting. It has good prediction rate on training set, but, poorer prediction rate on testing set, which indicates it suffers overfitting.

There might be around 18 features that are optimal. Some features come from the 10-feature set, and others from features in the 19-features set. Without multiple experiments, we cannot conclude anything without doubt.

**Executing the code (Fixed Split):**

* Run **2c/*PuppyTrainerFixedSplit.scala*** file from spark shell.
* For running with other dataSets, replace “5” as mentioned above.

With **fixed split** (top 80% as train data, the rest 20% as test data)

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Feature Set | 5-feature set | 9-feature set | 10-feature set | 15-feature set | 19-feature set | 22-feature  set |
| Training data prediction rate | 54.12% | 55.61% | 62.05% | 63.13% | 62.90% | 62.51% |
| Testing data prediction rate | 72.09% | 86.28% | **84.80%** | 83.96% | 79.61% | 82.23% |

During the presentation, one group raised the doubt as to why testing data prediction is better than training data.

To answer the question:

First, we checked our code, and we did not find a bug in our code.

Next we went through various materials online to find the solution and from the machine learning class offered in Coursera, the professor mentioned that the order of records sometime has certain rule embedded, and that he recommends to randomize the data before using it to train the model.

To validate if this might be the case with us, we conducted an experiment.

We changed the training data set and test data set. We use the top 20% as test data, and other 80% as train data. With this setting, the training prediction gets better than testing prediction for all datasets except the 5-feature dataset (which is not reliable anyway).

We did not have enough time to experiment more, however, this experient helped us to conclude that the anomaly pointed out above, is related to the order of the data.

***Note:*** *As stated above, the prediction rate of testing is better than training data is possible, because some relationship is beard in the order of the data. We will restate this when the phenomena happens in following sections.*

With fixed split (top 20% as test data, other 80% as train data)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Feature Set** | **5-feature set** | **10-feature set** | **15-feature set** | **19-feature set** | **22-feature**  **set** |
| Training data prediction rate | 58.48% | 71.37% | 69.93% | 72.21% | 69.47% |
| Testing data prediction rate | 61.90% | 47.46% | 48.99% | 40.15% | 51.54% |

# 

#### **2.**dPrediction on randomly split and non-randomly split data

*Repeat each cell of table 2.c 10 times. Do you see consistent prediction rate for the same feature set of data when it is randomly split? How about when the data is not randomly split?Fill in each trial result and the average of trials in the table as shown follows.*

**Executing the code (LBFGS) :**

* We are using same code from **2c**.
* Run **2c/*PuppyTrainerLBFGS.scala*** file from spark shell.
* This code executes for dataSet with 5 features. For other dataSets (9, 10, 15, 19, 22) please replace 5 with these values respectively.

We ran both algorithms with random and fixed split, and the result was consistent when ran on single machine. But when ran on a different machine we saw inconsistency in the results.

**On single Machine**

* With LogisticRegressionWithLBFGS on random and fixed split. The outcome is consistent. Below are the results:

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 5-feature set | 1st trial | 2nd trial | 3rd trial | 4th trial | 5th trial | 6th trial | 7th trial | 8th trial | 9th trial | 10th trial | average |
| training-randomly | 57.99% | 57.99% | 57.99% | 57.99% | 57.99% | 57.99% | 57.99% | 57.99% | 57.99% | 57.99% | 57.99% |
| test-randomly | 63.64% | 63.64% | 63.64% | 63.64% | 63.64% | 63.64% | 63.64% | 63.64% | 63.64% | 63.64% | 63.64% |
| training-fixed | 54.12% | 54.12% | 54.12% | 54.12% | 54.12% | 54.12% | 54.12% | 54.12% | 54.12% | 54.12% | 54.12% |
| test-fixed | 72.09% | 72.09% | 72.09% | 72.09% | 72.09% | 72.09% | 72.09% | 72.09% | 72.09% | 72.09% | 72.09% |

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 9-feature set | 1st trial | 2nd trial | 3rd trial | 4th trial | 5th trial | 6th trial | 7th trial | 8th trial | 9th trial | 10th trial | average |
| training-randomly | 61.92% | 61.92% | 61.92% | 61.92% | 61.92% | 61.92% | 61.92% | 61.92% | 61.92% | 61.92% | 61.92% |
| test-randomly | 68.93% | 68.93% | 68.93% | 68.93% | 68.93% | 68.93% | 68.93% | 68.93% | 68.93% | 68.93% | 68.93% |
| training-fixed | 55.61% | 55.61% | 55.61% | 55.61% | 55.61% | 55.61% | 55.61% | 55.61% | 55.61% | 55.61% | 55.61% |
| test-fixed | 86.28% | 86.28% | 86.28% | 86.28% | 86.28% | 86.28% | 86.28% | 86.28% | 86.28% | 86.28% | 86.28% |

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 10-feature set | 1st trial | 2nd trial | 3rd trial | 4th trial | 5th trial | 6th trial | 7th trial | 8th trial | 9th trial | 10th trial | average |
| training-randomly | 67.09% | 67.09% | 67.09% | 67.09% | 67.09% | 67.09% | 67.09% | 67.09% | 67.09% | 67.09% | 67.09% |
| test-randomly | 63.89% | 63.89% | 63.89% | 63.89% | 63.89% | 63.89% | 63.89% | 63.89% | 63.89% | 63.89% | 63.89% |
| training-fixed | 62.05% | 62.05% | 62.05% | 62.05% | 62.05% | 62.05% | 62.05% | 62.05% | 62.05% | 62.05% | 62.05% |
| test-fixed | 84.80% | 84.80% | 84.80% | 84.80% | 84.80% | 84.80% | 84.80% | 84.80% | 84.80% | 84.80% | 84.80% |

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 15-feature set | 1st trial | 2nd trial | 3rd trial | 4th trial | 5th trial | 6th trial | 7th trial | 8th trial | 9th trial | 10th trial | average |
| training-randomly | 68.22% | 68.22% | 68.22% | 68.22% | 68.22% | 68.22% | 68.22% | 68.22% | 68.22% | 68.22% | 68.22% |
| test-randomly | 61.80% | 61.80% | 61.80% | 61.80% | 61.80% | 61.80% | 61.80% | 61.80% | 61.80% | 61.80% | 61.80% |
| training-fixed | 63.13% | 63.13% | 63.13% | 63.13% | 63.13% | 63.13% | 63.13% | 63.13% | 63.13% | 63.13% | 63.13% |
| test-fixed | 83.96% | 83.96% | 83.96% | 83.96% | 83.96% | 83.96% | 83.96% | 83.96% | 83.96% | 83.96% | 83.96% |

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 19-feature set | 1st trial | 2nd trial | 3rd trial | 4th trial | 5th trial | 6th trial | 7th trial | 8th trial | 9th trial | 10th trial | average |
| training-randomly | 66.71% | 66.71% | 66.71% | 66.71% | 66.71% | 66.71% | 66.71% | 66.71% | 66.71% | 66.71% | 66.71% |
| test-randomly | 62.72% | 62.72% | 62.72% | 62.72% | 62.72% | 62.72% | 62.72% | 62.72% | 62.72% | 62.72% | 62.72% |
| training-fixed | 62.90% | 62.90% | 62.90% | 62.90% | 62.90% | 62.90% | 62.90% | 62.90% | 62.90% | 62.90% | 62.90% |
| test-fixed | 79.61% | 79.61% | 79.61% | 79.61% | 79.61% | 79.61% | 79.61% | 79.61% | 79.61% | 79.61% | 79.61% |

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 22-feature set | 1st trial | 2nd trial | 3rd trial | 4th trial | 5th trial | 6th trial | 7th trial | 8th trial | 9th trial | 10th trial | average |
| training-randomly | 67.78% | 67.78% | 67.78% | 67.78% | 67.78% | 67.78% | 67.78% | 67.78% | 67.78% | 67.78% | 67.78% |
| test-randomly | 61.31% | 61.31% | 61.31% | 61.31% | 61.31% | 61.31% | 61.31% | 61.31% | 61.31% | 61.31% | 61.31% |
| training-fixed | 62.51% | 62.51% | 62.51% | 62.51% | 62.51% | 62.51% | 62.51% | 62.51% | 62.51% | 62.51% | 62.51% |
| test-fixed | 82.23% | 82.23% | 82.23% | 82.23% | 82.23% | 82.23% | 82.23% | 82.23% | 82.23% | 82.23% | 82.23% |

* With LogisticRegressionWithSGD on random and fixed split. The outcome is consistent. Below are the results:

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| *10-feature set* | *1st trial* | *2nd trial* | *3rd trial* | *4th trial* | *5th trial* | *6th trial* | *7th trial* | *8th trial* | *9th trial* | *10th trial* | *average* |
| *training-randomly* | *66.92%* | *66.92%* | *66.92%* | *66.92%* | *66.92%* | *66.92%* | *66.92%* | *66.92%* | *66.92%* | *66.92%* | *66.92%* |
| *test-randomly* | *64.93%* | *64.93%* | *64.93%* | *64.93%* | *64.93%* | *64.93%* | *64.93%* | *64.93%* | *64.93%* | *64.93%* | *64.93%* |
| *training-fixed* | *62.13%* | *62.13%* | *62.13%* | *62.13%* | *62.13%* | *62.13%* | *62.13%* | *62.13%* | *62.13%* | *62.13%* | *62.13%* |
| *test-fixed* | *84.12%* | *84.12%* | *84.12%* | *84.12%* | *84.12%* | *84.12%* | *84.12%* | *84.12%* | *84.12%* | *84.12%* | *84.12%* |

*Summary table for feature set with feature size = 5*

**On a different Machine**

* For dataset with 5 features, we ran same algorithm on different machines and got different results for random split as shown below :

1st Machine :

***Software*** *: Scala (2.11.8),* ***Tool*** *: Spark (2.1.0)*

***Platform*** *used: MAC OS*

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 5-feature set | 1st trial | 2nd trial | 3rd trial | 4th trial | 5th trial | 6th trial | 7th trial | 8th trial | 9th trial | 10th trial | average |
| training-randomly | 57.99% | 57.99% | 57.99% | 57.99% | 57.99% | 57.99% | 57.99% | 57.99% | 57.99% | 57.99% | 57.99% |
| test-randomly | 63.64% | 63.64% | 63.64% | 63.64% | 63.64% | 63.64% | 63.64% | 63.64% | 63.64% | 63.64% | 63.64% |
| training-fixed | 54.12% | 54.12% | 54.12% | 54.12% | 54.12% | 54.12% | 54.12% | 54.12% | 54.12% | 54.12% | 54.12% |
| test-fixed | 72.09% | 72.09% | 72.09% | 72.09% | 72.09% | 72.09% | 72.09% | 72.09% | 72.09% | 72.09% | 72.09% |

2nd Machine :

***Software*** *: Scala (2.11.8),* ***Tool*** *: Spark (2.1.0)*

***Platform*** *used: Ubuntu*

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 5-feature set | 1st trial | 2nd trial | 3rd trial | 4th trial | 5th trial | 6th trial | 7th trial | 8th trial | 9th trial | 10th trial | average |
| training-randomly | 61.14% | 61.14% | 61.14% | 61.14% | 61.14% | 61.14% | 61.14% | 61.14% | 61.14% | 61.14% | 61.14% |
| test-randomly | 50.00% | 50.00% | 50.00% | 50.00% | 50.00% | 50.00% | 50.00% | 50.00% | 50.00% | 50.00% | 50.00% |
| training-fixed | 54.12% | 54.12% | 54.12% | 54.12% | 54.12% | 54.12% | 54.12% | 54.12% | 54.12% | 54.12% | 54.12% |
| test-fixed | 72.09% | 72.09% | 72.09% | 72.09% | 72.09% | 72.09% | 72.09% | 72.09% | 72.09% | 72.09% | 72.09% |

# 

# 

#### 2.e Optimal number of features

*What is the optimal number of features in your that offers best prediction rate based on 2.e?*

* 9 features and 10 features are optimal among the datasets we tested based on the test result. However, based upon the analysis on the outcome of datasets in 2.a (please refer to 2.a’s analysis), there might be around 18 features that are optimal. Some comes from the 10-feature set, and some comes from other features in 19-features set. However, without plenty of experiment, we can not conclude anything without doubt. More information, please refer to the explanation in 2.a.
* We cannot have too many features in our feature -set as having more features than requires can harm our learning algorithm as follows:  
  - It can lead to increased processing time for processing irrelevant features that doesn’t add value to our prediction rate  
  - If there are too many features in our dataset,it can lead to overfitting
* Likewise, we cannot take too less number of features because it can cause underfitting in two ways  
  a) It has less data  
  b) It has less feature. If those features are not at all adding value to our prediction rate, then that feature -set might have very poor predictive performance.

*What features are in the dataset? Is it consistent with your 2.c result?*

Below are the features in dataset :

**"Age", "Health", "RespondsToCommandKennel", "GoodWStrangers", "TrafficFear", "StaysOnCommand", "BehavesWellClass", "Sex", "Breed"**

The result is consistent with the 2.c result :

Training Accuracy : 61.40%

Testing Accuracy : 68.93%

#### 

#### 2.f List of Algorithms and Parameters used.

*List in a table the choice of parameters. e.g., maxdepth, choice of gradient descent vs. stochastic gradient descent, that your algorithm provides. Check on the options you tried.*

**1) RDD based API -**

We tried below algorithms :

* LogisticRegressionWithLBFGS
* LogisticRegressionWithSGD

*Explain which parameters worked better than others. Highlight in bold the prediction rate and the parameters used for your best prediction trial.*

*Indicate in your report what program(s) and data to run and how to run to get this best prediction rate*

***Software*** *: Scala (2.11.8),* ***Tool*** *: Spark (2.1.0)*

***Platform*** *used: Ubuntu*

**Executing the code (SGD and LBFGS):**

* Run **2f/*PuppyTrainerSGD.scala*** file from spark shell.
* Run **2f/*PuppyTrainerLBFGS.scala*** file from spark shell.
* For running with other dataSets (Here it’s 10), replace “5” as mentioned above.

Dataset used to run the code : ***2f/datasets/2fdataset.txt***

For both files “**setIntercept = true**”

Please replace “true” with “false” if needed to run without intercept.

*Provide a weblink to an online algorithm parameter choice overview you follow.*

* We followed spark [documentation](https://spark.apache.org/docs/2.1.0/api/scala/index.html#org.apache.spark.mllib.classification.LogisticRegressionWithLBFGS) for all available parameters.
* Spark [MLlib](http://spark.apache.org/docs/latest/mllib-guide.html) documentation

*Show the prediction rate, the parameter, and data used for your best prediction trial.*

Parameters used :

* **setIntercept**
* **setValidateData** (Results were same with true and false)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 10-feature set | LogisticRegressionWithLBFGS | LogisticRegressionWithLBFGS  (setIntercept = true) | LogisticRegressionWithSGD | LogisticRegressionWithSGD  (setIntercept = true) |
| training | 67.30% | 67.05% | 67.05% | 59.75% |
| test | 64.96% | **65.33%** | 62.77% | **63.14%** |

*Explain why you use those parameters/algorithms and not others.*

* LogisticRegressionWithLBFGS can often get the better solution than the other methods with less iteration.
* It approximates BFGS algorithm using a limited amount of computer memory and recommended over SGD.
* SGD does better when the data is large. This is slower than LBFGS.
* With setIntercept = true, the algorithm will add intercept and this gives better results when there is consistent 0 in data, which is not the case in our dataset. That’s why setIntercept doesn’t help much here.

**2) DataFrame based API -**

*Explain which parameters worked better than others. Highlight in bold the prediction rate and the parameters used for your best prediction trial.*

*Indicate in your report what program(s) and data to run and how to run to get this best prediction rate*

**Executing the code :**

* Run **2f/*PuppyInfoParams.scala*** file from spark shell.

Dataset used to run the code : ***2f/datasets/2fdataset.csv***

*Provide a weblink to an online algorithm parameter choice overview you follow.*

* We followed spark [documentation](http://spark.apache.org/docs/latest/api/scala/index.html#org.apache.spark.ml.classification.LogisticRegression) for all available parameters.

For [LogisticRegression](http://spark.apache.org/docs/latest/api/scala/index.html#org.apache.spark.ml.classification.LogisticRegression) Algorithm we tried below parameters :

* **setMaxIter**
* **setRegParam**
* **setFitIntercept**
* **setStandardization**
* **setThreshold**
* **setTol**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Algorithm  /Values | **setMaxIter** | **setRegParam** | **setFitIntercept** | **setStandardization** | **setThreshold** | **setTol** |
| 1.value | 5 | 0.0 | false | false | Default | Default |
| 2.value | 5 | 0.3 | true | true | 0.6 | 0.3 |
| **3.value** | **10** | **0.0** | **false** | **false** | **Default** | **Default** |
| 4.value | 10 | 0.5 | true | true | 0.4 | 0.001 |
| 5.value | 100 | 0.001 | false | false | Default | Default |
| 6.value | 500 | 0.001 | false | false | 0.55 | 0.0 |
| 7.value | 50 | 0.0 | true | true | 0.55 | 0.00001 |
| 8.value | 50 | 0.0 | true | false | Default | Default |
| 9.value | 100 | 0.0 | false | true | 0.55 | Default |
| 10.value | 100 | 0.0 | false | false | Default | Default |

**Results for above 10 cases** :

|  |  |  |
| --- | --- | --- |
| Output  /Values | **Training Result** | **Testing Result** |
| 1.value | 62.87% | **62.78%** |
| 2.value | 57.16% | 57.46% |
| **3.value** | **64.30%** | **61.15%** |
| 4.value | 53.74% | 49.78% |
| 5.value | 63.90% | 61.15% |
| 6.value | 63.40% | 61.74% |
| 7.value | 63.47% | 61.89% |
| 8.value | 63.93% | 61.30% |
| 9.value | 63.47% | 61.89% |
| 10.value | 64.07% | 61.30% |

*Explain why you use those parameters/algorithms.*

* We see that 3rd case gives the best result.
* **setMaxIter** will specify the number of iterations the algorithm will use while running. If we specify it less than default value the we saw better results.
* If **setTol** value is more than we are getting less accuracy. Setting it to default gives better results.
* If **setFitIntercept** than we do not see much difference in prediction. But for some cases is little more because intercept is added while running.
* Setting **setRegParam** is responsible for setting regularization parameter and less value (0.0) gives better results.
* If setThreshold value is high, than most probability the prediction will be 0 and low threshold encourages the model to predict 1 more often. So we get better results if value is 0.5 or if near to this.

**3) TF parameters**

**Executing the code :**

* Run **2f/*PuppyInfoTFParams.scala*** file from spark shell.

Online Resource : [documentation](https://spark.apache.org/docs/2.1.0/api/scala/index.html#org.apache.spark.ml.feature.HashingTF)

Dataset used to run the code : ***2f/datasets/2fdataset.csv***

* Here we are using some sentence based columns from PuppyInfo for prediction
* Parameters of TF used :

#### setNumFeatures

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| setNumFeatures Value | 20 | 1000 | 2000 | 3000 |
| training | 64.30% | 82.03% | **84.16%** | 83.36% |
| test | 61.15% | 79.17% | **80.80%** | 80.80% |

* Setting this feature to 2000 gives us better results.
* This value states the number of features the algorithm should consider to extract features.
* Our input sentences contain many words and so setting this value to large number gives better results.
* But after 2000 the prediction accuracy decreases as seen for 3000.

# 

# PART 3 Prediction model based on trainer info

*Normalization and Feature Extraction of text data*

#### 3.a 1) **Data Preparation**

To create model using trainer info, we need to extract the data with label(training outcome) and DayInLife. This procedure is done by following steps.

***Software*** *: Python 2.7.13,* ***Tool*** *: Microsoft excel*

***Platform*** *used: MAC OS*

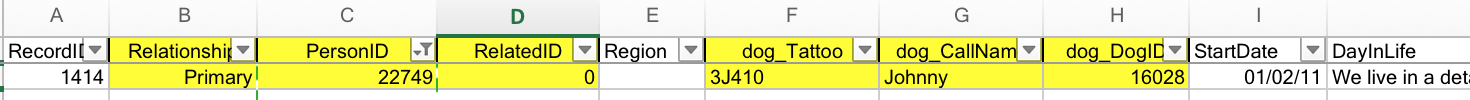
**Step1: Analyze Data**

Analyze data in ***PuppyTrainerOutcome.xlsx*** and ***TrainerInfo.xlsx***.

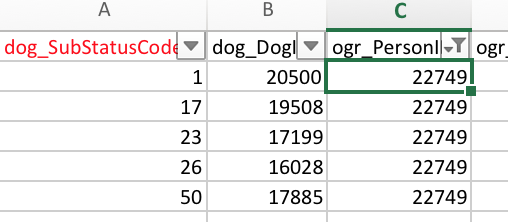
In *TrainerInfo.xlsx*, there is no data to indicate the outcome of this trainer. Considering using “PersonID” and “dog\_DogID” pair to find the outcome from *PuppyTrainerOutcome.xlsx*.

Ex:

For trainer with person id 22749 in *trainerInfo.xlsx*



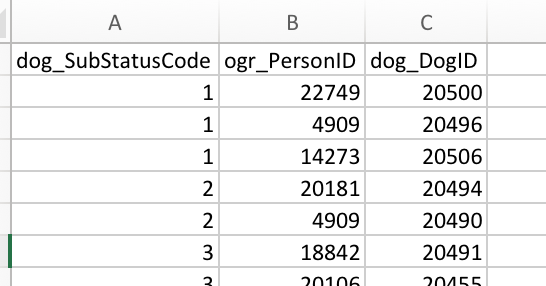
There are many outcomes of this trainer paired with different dogs.



Based on the analysis, The pair of “PersonID” and “dog\_DogID” should be used to identify the outcome associated with the “DayInLife” description.

**Step2: Preprocess Outcome Data**

Manually clean “***PuppyTrainerOutcome.xlsx****”* to contains only “dog\_SubStatusCode”, “ogr\_PersonID” and “dog\_DogID”. Refer to “**3a/dataset/*PuppyTrainerOutcome-after-step2.csv”***



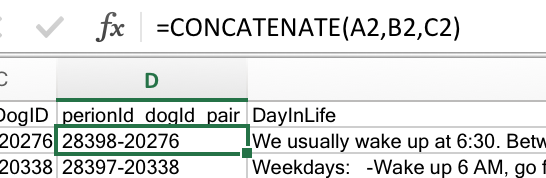
**Step3: Preprocess Trainer Data**

Manually clean “***PuppyTrainerOutcome.xlsx****”* to contains only “PersonID”, “dog\_DogID”, and “DayInLife”.

Using concatenate function in Microsoft excel to generate “perionId\_dogId\_pair”. Refer to “***TrainerInfo-after-step3.xlsx***” sheet “PreplacementQuizApplicantRaiser”

Clean the data again to contains only “perionId\_dogId\_pair” and “DayInLife”. Refer to “***TrainerInfo-after-step3.xlsx***” sheet “CleanedData”.

Then generate csv file, refer to “***TrainerInfo-after-step3.csv***”



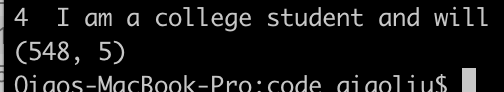
**Step4: Generate Data with Label and DayInLife**

This Step is done by python code ***process\_data.py***

***To run this code, first “cd” to the folder of the code, which is [YOUR FOLDER]/3a. Then run command “python process\_data.py”***

There are 548 records without duplicates.

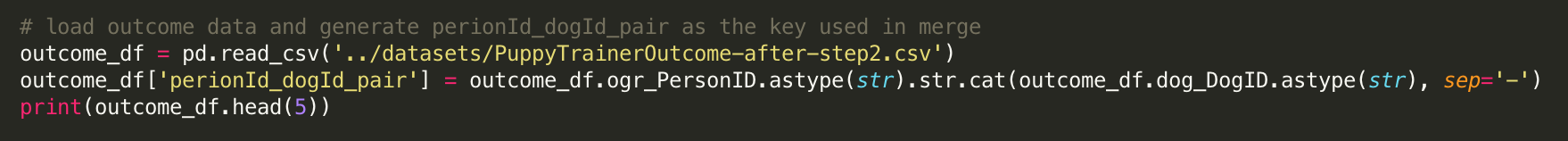
After removing the 1 column which is null, there are 547 records.



Explanation of the code is as below

4.1

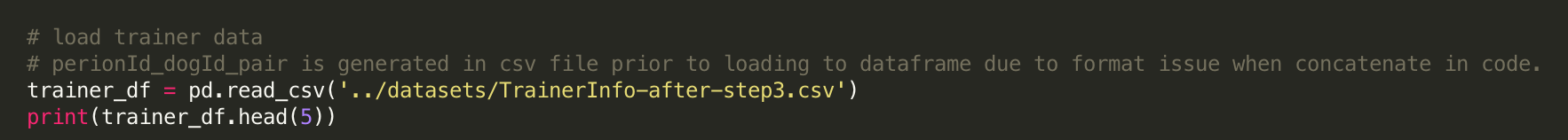
Load outcome csv file, and generate perionId\_dogId\_pair, which will be used as the key in merge operation.



4.2

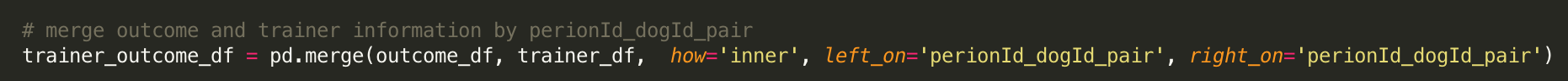
Load trainer data

*Note: trainer data already has perionId\_dogId\_pair column in Step3.*



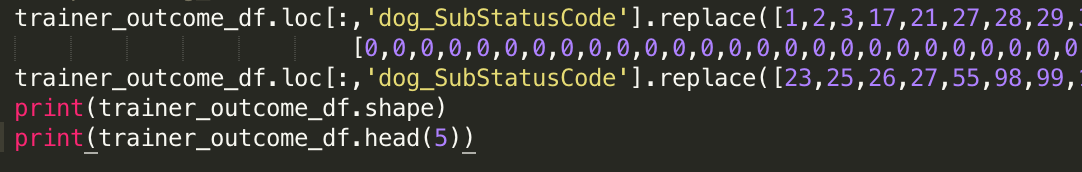
4.3

Merge outcome and trainer information by perionId\_dogId\_pair



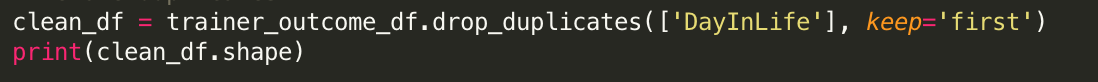
4.4

Works on data consistency of statuscode



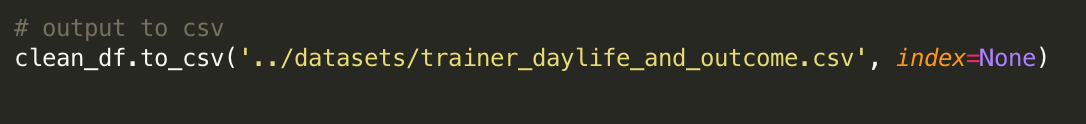
4.5

Remove duplicates by “DayInLife” which is the feature in your model



4.6

Output dataframe to csv file.



4.7

Select dog\_SubStatusCode and DayInLife column from ***trainer\_daylife\_and\_outcome.csv***, and remove the column, which is null in DayInLife to generate the final dataset.

Refer file for final dataset : **3a/datasets/*3adataset.csv***

#### 3.a 2) Text Normalization and Feature Transformation

or **DayInLife** column in ***3adataset.csv*** :

*Show in a list of what normalization steps you did and with what code/library:*

**Java normalizer library** is used to normalize text.

Import Library using : *import java.text.Normalizer*

Online Resource referred for code : [javaNormalizer](http://www.programcreek.com/java-api-examples/java.text.Normalizer)

***Software*** *: Scala (2.11.8),* ***Tool*** *: Spark (2.1.0)*

***Platform*** *used: Ubuntu*

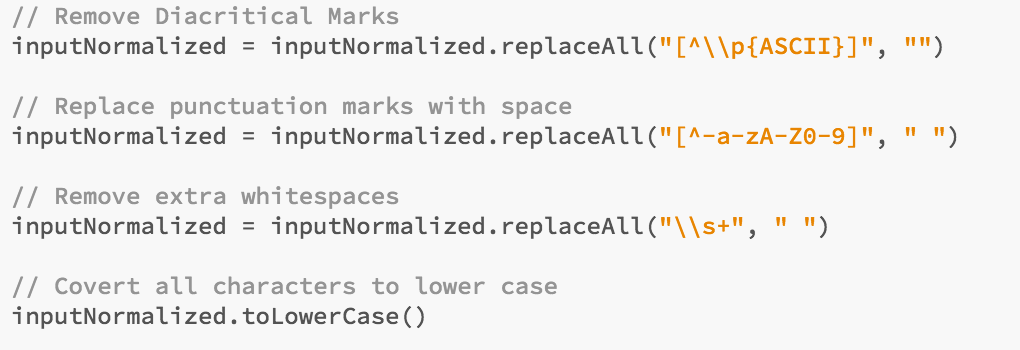
**Executing the code :**

* Run the file ***3a/TextNormalize3a.scala*** from spark shell

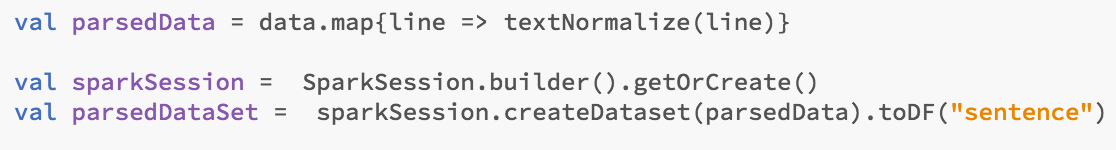
Dataset used to run the code : ***3a/datasets/3adataset.txt***

**Text normalization** is done by making following changes -

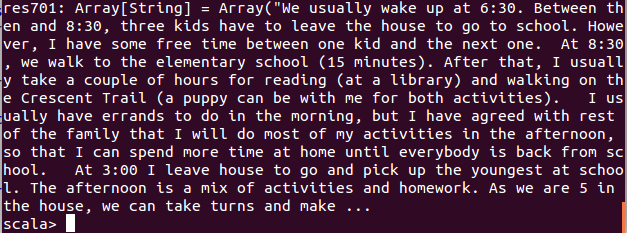
* Replacing diacritical marks with ASCII
* Removing punctuations by allowing only alphanumeric characters
* Removing extra spaces
* Converting all characters to lowercase



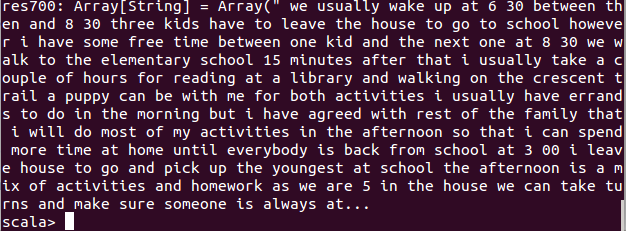
* Here *textNormalize()* function will run to normalize data in above fashion.
* Create data set using *SparkSession* and set column name to “*sentence*”



Below is the **input** for 1st row :

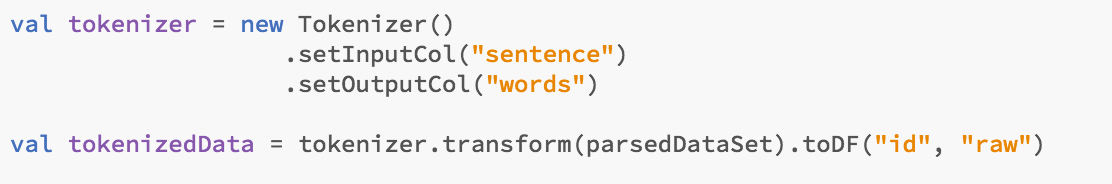


**Output** after text normalization is :



**Feature transformation** is done by using spark algorithms mentioned below -

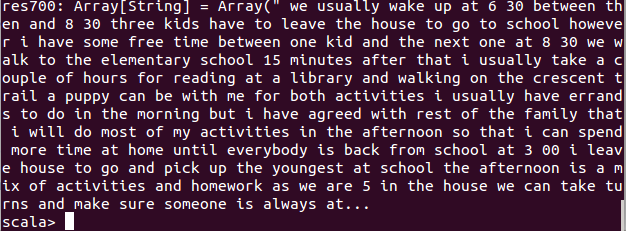
* Tokenizer
  + This is the process where text (sentence) is broken down into individual terms (words)



* StopWordsRemover
  + This algorithm will remove words which appear very frequently and don’t carry as such meaning.
  + This takes input as sequence of strings (words), here it’s output of tokenizer.



**Input** given to tokenizer is output from text normalization :



**Output** of stopWordRemover for 1st row :

#### Screen Shot 2017-05-12 at 8.43.45 AM.png

#### 

#### 

#### 3.b Feature Extraction

*Explain what feature extraction in Spark MLlib are useful for this purpose.Show what feature extraction you did and describe what code/library was used.*

Online Resource : [spark-ML-features](https://spark.apache.org/docs/2.1.0/ml-features.html)

***Software*** *: Scala (2.11.8),* ***Tool*** *: Spark (2.1.0)*

***Platform*** *used: Ubuntu*

**Executing the code (Same code from 3a has this section covered) :**

* Run the file ***3a/TextNormalize3a.scala*** from spark shell

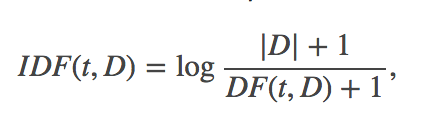
Dataset used to run the code : ***3a/datasets/3adataset.txt***

Spark provides few algorithms for feature extraction :

* TF-IDF
* Word2Vec
* CountVectorizer

We used **TF-IDF** algorithm for feature extraction :

* It reflects the importance of a term (word) in a document.
* TF(t, d) is number of times the term occurs in document d. We need to apply IDF also because using only TF can cause over-emphasizing the importance of the frequently occurring term which carries very less importance.
* IDF is represented as below. Here |D| is total number of documents in corpus.

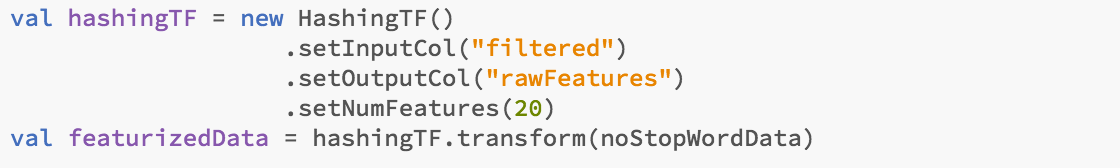


* The TF-IDF measure is the product of TF and IDF:

Screen Shot 2017-05-12 at 3.29.23 PM.png

* This is implemented as shown below for our problem :

TF -



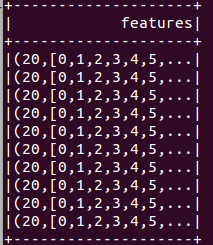
IDF -



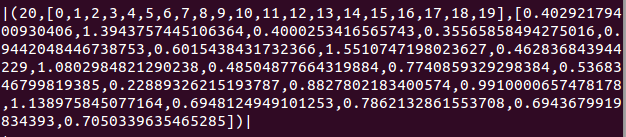
**Input** for TF-IDF is output from stopWordRemover :

#### Screen Shot 2017-05-12 at 8.43.45 AM.png

Final **output** for first few rows :



**Output** for 1st row :



We also used **Word2Vec** Feature Extaction

* Word2Vec is an Estimator which takes sequences of words in the documents and trains a Word2VecModel.
* The model maps each word to a unique fixed-size vector.
* The Word2VecModel transforms each document into a vector using the average of all words in the document; this vector can then be used for training and prediction.

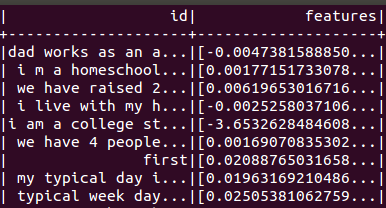


* Here “filtered” column is output of StopWordRemover.
* Output of Word2Vec is stored in “features”.

**Input** for TF-IDF is output from stopWordRemover :

#### Screen Shot 2017-05-12 at 8.43.45 AM.png

Final **output** for first few rows :



**Output** for 1st row :

# Screen Shot 2017-05-20 at 4.35.32 PM.png

# 

#### 3.c Prediction Rate

*Build and train your model based on the data in 3.b.*

***Software*** *: Scala (2.11.8),* ***Tool*** *: Spark (2.1.0)*

***Platform*** *used: Ubuntu / MAC OS*

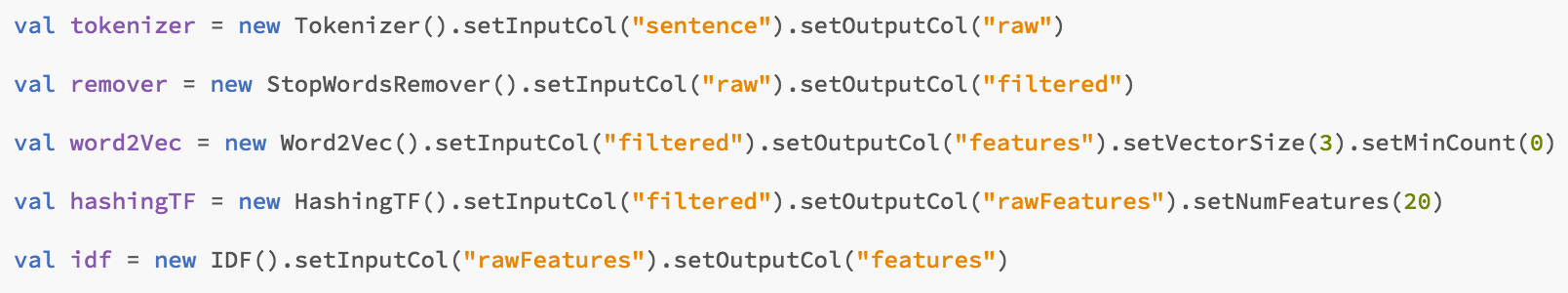
**Executing the code :**

* Run the file ***3c/TextNormalize3c.scala*** from spark shell

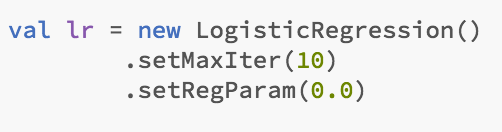
Dataset used to run the code : ***3c/datasets/3cdataset.csv***

*Explain how you make use of your extracted features to build your model.*

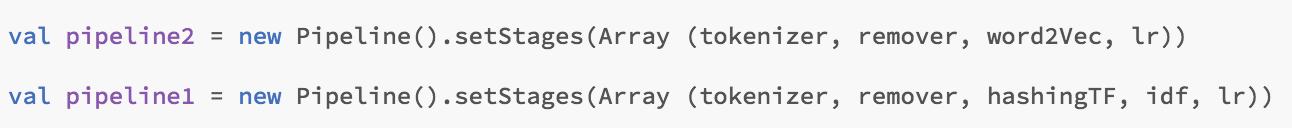
* We are using spark **pipeline** algorithm to do feature extraction and prediction.
* Create definitions for algorithms (e,g tokenizer, tf-idf, word2Vec)



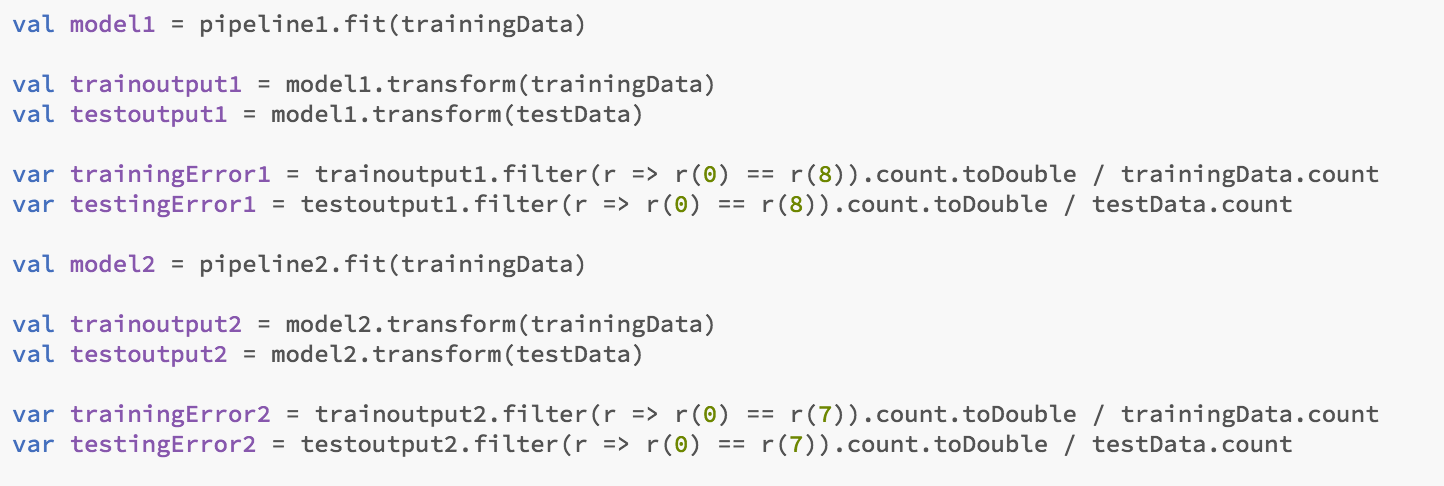
* Create instance of LogisticRegression algorithm



* Create 2 pipelines. One for TF-IDF and another for Word2Vec



* Train both to create Models (mode1, mode2).
* This training model will be used to test the data using “.transform()” method.



* Compare “label” (Expected status) and “prediction” (Predicted status) column to find the accuracy.
* More detailed code with multiple features is explained in **“B.3” part.**

*Do you see consistent prediction rate when the data is randomly split?*

We see consistent prediction rate when data is randomly split :

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Data Random split | 1st trail | 2nd trial | 3rd trial | 4th trial | 5th trial |
| Training  (TF-IDF) | 67.51% | 67.51% | 67.51% | 67.51% | 67.51% |
| Test  (TF-IDF) | 61.82% | 61.82% | 61.82% | 61.82% | 61.82% |
| Training  (word2Vector) | 66.82% | 66.82% | 66.82% | 66.82% | 66.82% |
| Testing  (word2Vector) | 64.55% | 64.55% | 64.55% | 64.55% | 64.55% |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Data Random split | 6th trail | 7th trial | 8th trial | 9th trial | 10th trial |
| Training  (TF-IDF) | 67.51% | 67.51% | 67.51% | 67.51% | 67.51% |
| Test  (TF-IDF) | 61.82% | 61.82% | 61.82% | 61.82% | 61.82% |
| Training  (word2Vector) | 66.82% | 66.82% | 66.82% | 66.82% | 66.82% |
| Testing  (word2Vector) | 64.55% | 64.55% | 64.55% | 64.55% | 64.55% |

|  |  |
| --- | --- |
| Data Random split | Average |
| Training  (TF-IDF) | 67.51% |
| Test  (TF-IDF) | 61.82% |
| Training  (word2Vector) | 66.82% |
| Testing  (word2Vector) | 64.55% |

*How about when the data is not randomly split? Fill in each trial result and the average of trials in the table as shown follows. (??%)*

* The result of fixed split, with top 80% Training, other 20% testing, is consistent

Run ***3cFixed.scala*** file in 3c folder from spark shell.

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Data Fixed Split | 1st trial | 2nd trial | 3rd trial | 4th trial | 5th trial | 6th trial | 7th trial | 8th trial | 9th trial | 10th trial | average |
| Training (TF-IDF) | 99.08% | 99.08% | 99.08% | 99.08% | 99.08% | 99.08% | 99.08% | 99.08% | 99.08% | 99.08% | **99.08%** |
| Testing  (TF-IDF) | 83.64% | 83.64% | 83.64% | 83.64% | 83.64% | 83.64% | 83.64% | 83.64% | 83.64% | 83.64% | **83.64%** |
| Training  (word2Vector) | 62.01% | 62.01% | 62.01% | 62.01% | 62.01% | 62.01% | 62.01% | 62.01% | 62.01% | 62.01% | **62.01%** |
| Testing  (word2Vector) | 85.45% | 85.45% | 85.45% | 85.45% | 85.45% | 85.45% | 85.45% | 85.45% | 85.45% | 85.45% | **85.45%** |

* The result of fixed split, with top 20% Testing, other 80% testing, is consistent

Run **3cFixed-Switched.scala** file in 3c folder from spark shell.

In the previous table of fixed training (with top 80% Training, other 20% testing), we get better prediction rate of testing data on both TF-IDF and Word2Vector than randomly split. And, you will see In the switched fixed (split with top 20% Testing, other 80% testing) in the table below, the prediction of test data is very poor. From this observation, we can see the data at the beginning of the dataset has more useful information.

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Data Fixed Split | 1st trial | 2nd trial | 3rd trial | 4th trial | 5th trial | 6th trial | 7th trial | 8th trial | 9th trial | 10th trial | average |
| Training (TF-IDF) | 97.95% | 97.95% | 97.95% | 97.95% | 97.95% | 97.95% | 97.95% | 97.95% | 97.95% | 97.95% | **97.95%** |
| Testing  (TF-IDF) | 28.44% | 28.44% | 28.44% | 28.44% | 28.44% | 28.44% | 28.44% | 28.44% | 28.44% | 28.44% | **28.44%** |
| Training  (word2Vector) | 75.34% | 75.34% | 75.34% | 75.34% | 75.34% | 75.34% | 75.34% | 75.34% | 75.34% | 75.34% | **75.34%** |
| Testing  (word2Vector) | 28.44% | 28.44% | 28.44% | 28.44% | 28.44% | 28.44% | 28.44% | 28.44% | 28.44% | 28.44% | **28.44%** |

# 

# 

# 

# BONUS Features

*The ideal prediction model should take into account of both the puppy and the trainer info in deciding the training outcome.*

#### B.1 Prediction using both Trainer and Puppy

*If you blend both the trainer and puppy info used in 2.e in building/training a new model, is the prediction rate better or worse? Why?*

According to 2e, we selected 9 features as optimal based on the result test. These are the recommended features given by RFE.  
  
But when we combined it with trainer data, our predicate rate is worse. This could be cause the having irrelevant features can harm our learning algorithm as follows:.  
-If those features are not at all adding value to our prediction rate, then that feature -set might have very poor predictive performance

- It can lead to increased processing time for processing irrelevant features that doesn’t add value to our prediction rate

# 

#### B.2 Prediction with Random and Fixed Split

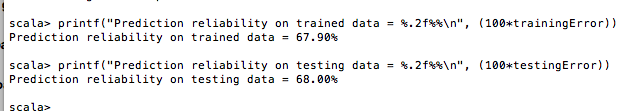
*Do you see consistent prediction rate when the data is randomly split?*

**Software** : Scala (2.11.8), **Tool** : Spark (2.1.0)

**Platform** used: MAc OS

**Executing the code :**

* Run **b2/TextNormalizerb2.scala** file from spark shell.

**Data Used** to run the code : **b2/datasets/b2dataset.csv**We are seeing consistent prediction when data is randomly split.   
  


|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Data Fixed Split | 1st trial | 2nd trial | 3rd trial | 4th trial | 5th trial | 6th trial | 7th trial | 8th trial | 9th trial | 10th trial | average |
| Training (TF-IDF) | 67.90% | 67.90% | 67.90% | 67.90% | 67.90% | 67.90% | 67.90% | 67.90% | 67.90% | 67.90% | 67.90% |
| Testing  (TF-IDF) | 68.00% | 68.00% | 68.00% | 68.00% | 68.00% | 68.00% | 68.00% | 68.00% | 68.00% | 68.00% | 68.00% |

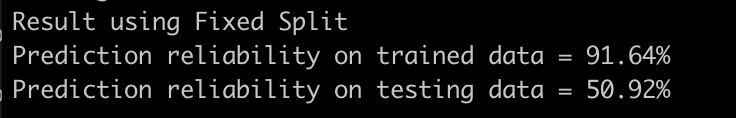
*How about when the data is not randomly split? Fill in each trial result and the average of trials in the table as shown follows.   
There is a little consistency when data is randomly split.*

***Software*** *: Scala (2.11.8),* ***Tool*** *: Spark (2.1.0)*

***Platform*** *used: MAC OS*

* The result of fixed split, with top 80% Testing, other 20% testing, is consistent

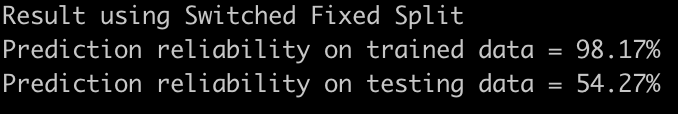
Run **b2Fixed.scala** file in b2 folder from spark shell.



|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Data Fixed Split | 1st trial | 2nd trial | 3rd trial | 4th trial | 4th trial | 4th trial | 4th trial | 4th trial | 4th trial | 10th trial | average |
| Training (TF-IDF) | 91.64% | 91.64% | 91.64% | 91.64% | 91.64% | 91.64% | 91.64% | 91.64% | 91.64% | 91.64% | 91.64% |
| Testing  (TF-IDF) | 50.92% | 50.92% | 50.92% | 50.92% | 50.92% | 50.92% | 50.92% | 50.92% | 50.92% | 50.92% | 50.92% |

* The result of fixed split, with top 20% Testing, other 80% testing, is consistent

Run **b2Fixed-Switched.scala** file in b2 folder from spark shell.



When using fixed split, we get good rate of training data, but bad rate of testing data. The model is overfitting to training data, it might because of the model or the training data is not generalized. We tried many parameter, the one in the code is the best one we get. So, most likely, the data is fixed split is not general data, which can **not** present the feature of the data.

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Data Fixed Split | 1st trial | 2nd trial | 3rd trial | 4th trial | 4th trial | 4th trial | 4th trial | 4th trial | 4th trial | 10th trial | average |
| Training (TF-IDF) | 95.97% | 95.97% | 95.97% | 95.97% | 95.97% | 95.97% | 95.97% | 95.97% | 95.97% | 95.97% | 95.97% |
| Testing  (TF-IDF) | 55.92% | 55.92% | 55.92% | 55.92% | 55.92% | 55.92% | 55.92% | 55.92% | 55.92% | 55.92% | 55.92% |

#### B.3 Prediction Rate with Puppy Info text/sentence based

*Select few columns of puppy info that is text based and in sentences. Perform normalization and feature extraction on these selected columns. Describe with screenshots and code how you did these.*

***Software*** *: Scala (2.11.8),* ***Tool*** *: Spark (2.1.0)*

***Platform*** *used: Ubuntu*

**Executing the code :**

* Run **b3/*TextNormalizerb3.scala*** file from spark shell.

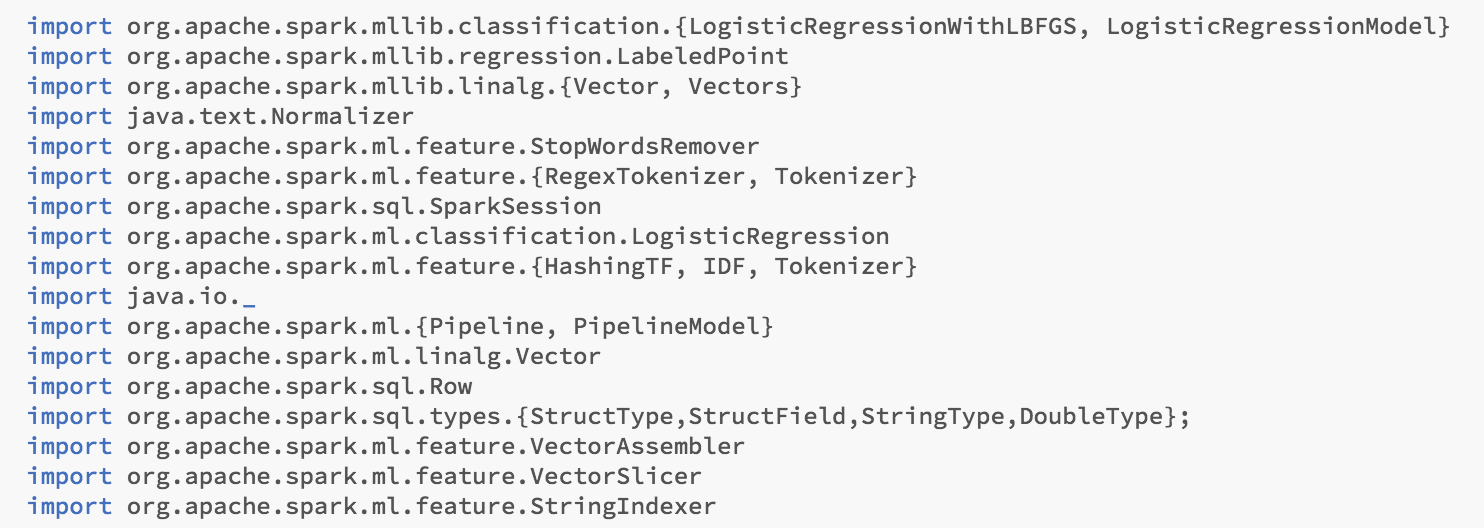
**Data Used** to run the code : **b3/datasets/b3dataset.csv**

The **dataset** contains following features :

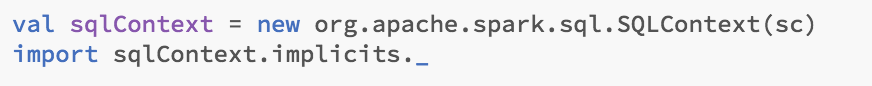
* + "GoodAppetite" : *Text*
  + "FoodType" : *Sentence*
  + "ExerciseType" : *Sentence*
  + "dog\_Sex" : *Text*
  + "dbc\_DogBreedDescription" : *Text*

Code is explained below :

* Import All Libraries required -

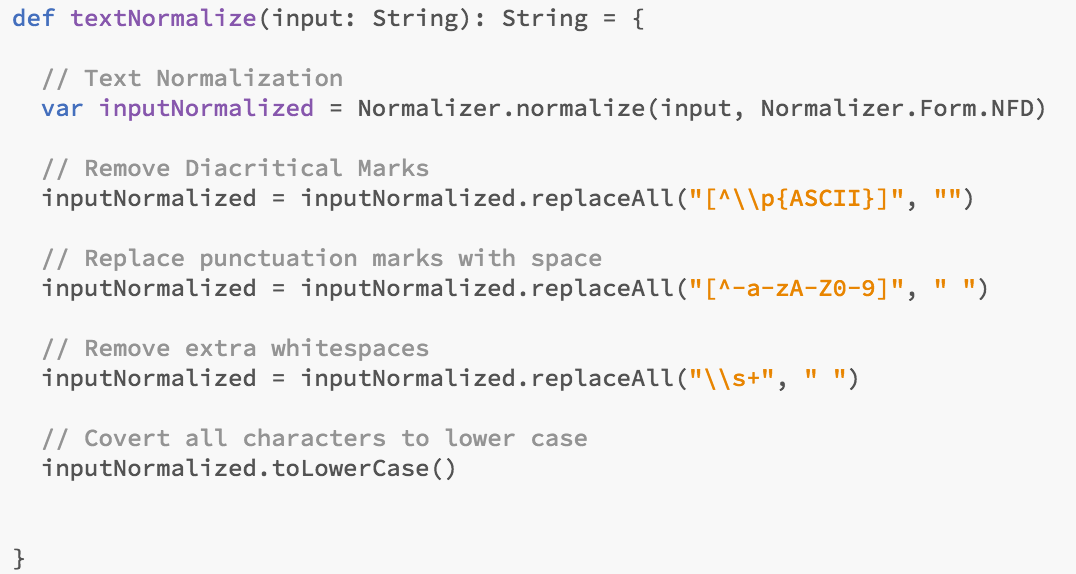


* Parse the dataset using SQLContext and set “header” to true

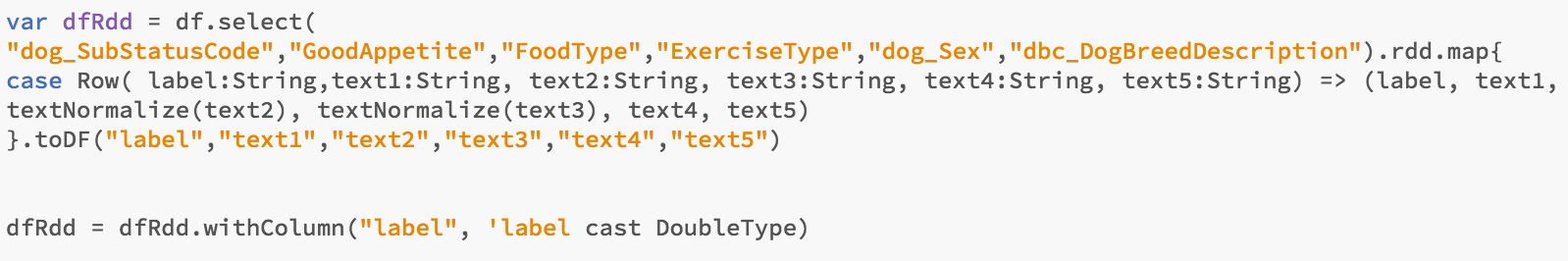




* Text Normalization function definition is show below.
* All steps are explained in 3.a) section



* Above “**textNormalize()**” method is used for “sentence” based columns
* Use **map** transformation to apply above method to 2 sentence based columns



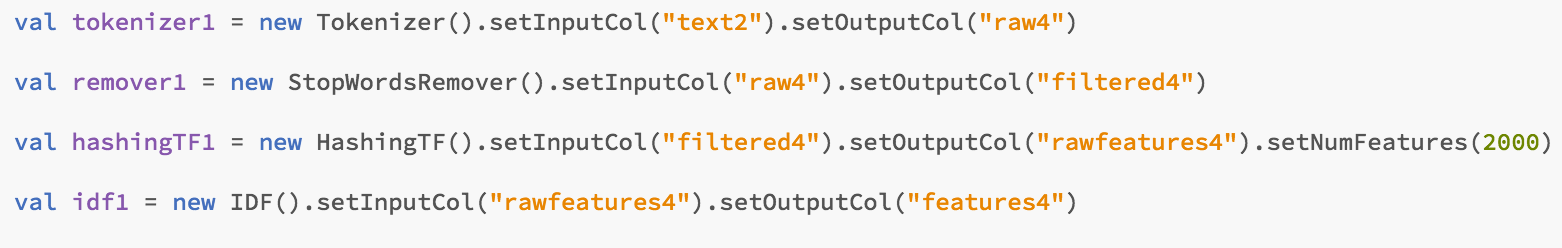
* Here Column names are changed to “**text1**”, “**text2**”...etc for simplicity. And “**dog\_SubStatusCode**” name is changed to “**label**”
* And “dog\_SubStatusCode” column is parsed as a “String”. So **convert it to Double** type using “**.withColumn()**” method.
* Split the above created dataset into 80-20 %.



* We have some text based columns which are 1 or 2 words. We are using [StringIndexer](https://spark.apache.org/docs/2.1.0/ml-features.html#stringindexer) to convert them into indices.



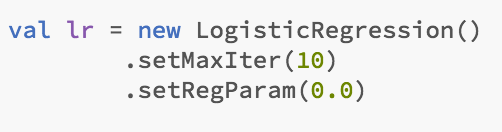
* For long sentence based columns we used below feature extraction and transformation methods :
  + Tokenizer
  + StopWordRemover
  + TF-IDF



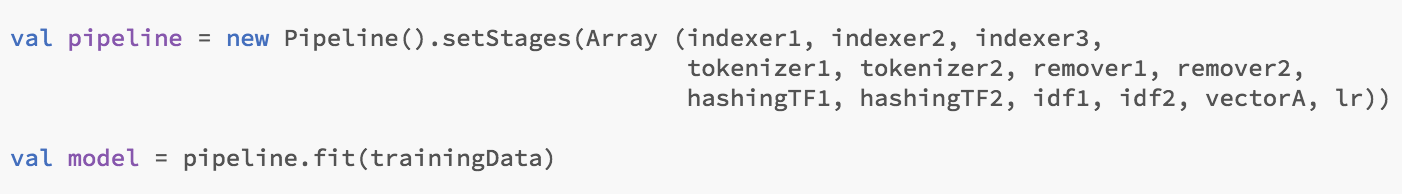
* Now we need to combine all features extracted from above 5 columns using [VectorAssembler](https://spark.apache.org/docs/2.1.0/ml-features.html#interaction).
* Here the Output column “features” will be given as input to training the data using Logistic Regression algorithm.



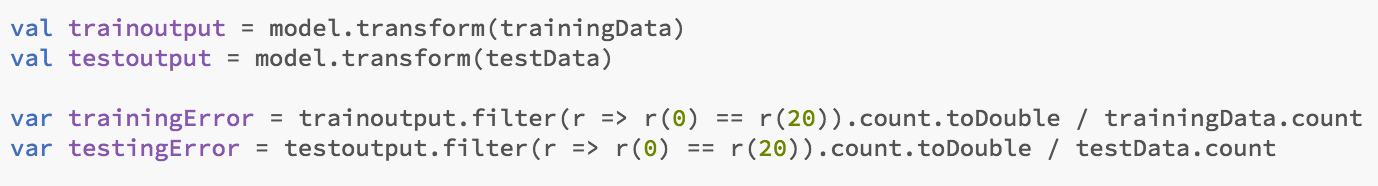
* Below is the Logistic regression algorithm definition. We have set below parameters to get better results.
* All Parameters tried are explained in **2.f) part**
* By default :
  + “label” column is expected label column.
  + “features” is Input column.
  + “prediction” is predicted result column



* Now use pipeline to execute all above steps sequentially (e.g All stringIndexer, All tokenizers, All StopWordRemovers, All TF-IDFs, VectorAssembler, LogisticRegression)
* After training, model will be created and it will be used for testing.



* Now use “.transform()” method to test the model and find prediction rate.
* In our dataset “label” (Expected Status) is 1st column.
* And “prediction” (Actual predicted Status) is by default last column after transform.
* So compare 1st and last column to find the prediction rate.

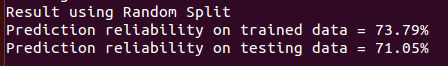


* Here we see that our “trainoutput” and “testoutput” have 21 columns. So Compare 0th and 20th column.

# Screen Shot 2017-05-21 at 7.02.16 PM.png

* Print the Result :





Below are the results for 10 trials:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Dataset | 1st trail | 2nd trial | 3rd trial | 4th trial | 5th trial |
| training | 73.79% | 73.79% | 73.79% | 73.79% | 73.79% |
| test | 71.05% | 71.05% | 71.05% | 71.05% | 71.05% |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 12-feature set | 6th trail | 7th trial | 8th trial | 9th trial | 10th trial |
| training | 73.79% | 73.79% | 73.79% | 73.79% | 73.79% |
| test | 71.05% | 71.05% | 71.05% | 71.05% | 71.05% |

|  |  |
| --- | --- |
| 12-feature set | Average |
| training | 73.79% |
| test | 71.05% |

# 

#### B.4 **Prediction Rate with Puppy Info numeric and text/sentence based**

*Add few other selected columns with numeric data to these transformed text columns and build/train a model before applying it for your prediction.*

*Do you see consistent prediction rate for the same feature set of data when it is randomly split?*

**Software** : Scala (2.11.8), **Tool** : Spark (2.1.0)

**Platform** used: Ubuntu

**Executing the code :**

* Run **b4/TextNormalizerb4.scala** file from spark shell.

**Data Used** to run the code : **b4/datasets/b4dataset.csv**

We have selected below columns from PuppyInfo :

**dog\_SubStatusCode", "Health", "EnergyLevel", "RespondsToCommandKennel", "GoodWStrangers", "BehavesWellClass", "NailCutting", "GoodAppetite", "FoodType", "ExerciseType", "Sex", "Breed"**

* Above columns include numeric, text and sentences type.
* We tried other columns as well, but above columns gave us best prediction rate.

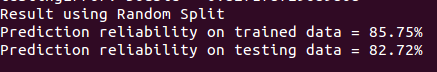
*Do you see consistent prediction rate for the same feature set of data when it is randomly split?*

* We got consistent result throughout .

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 12-feature set | 1st trail | 2nd trial | 3rd trial | 4th trial | 5th trial |
| training | 85.75% | 85.75% | 85.75% | 85.75% | 85.75% |
| test | 82.72% | 82.72% | 82.72% | 82.72% | 82.72% |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 12-feature set | 6th trail | 7th trial | 8th trial | 9th trial | 10th trial |
| training | 85.75% | 85.75% | 85.75% | 85.75% | 85.75% |
| test | 82.72% | 82.72% | 82.72% | 82.72% | 82.72% |

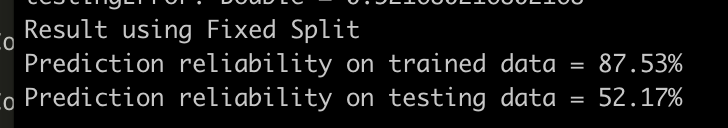
|  |  |
| --- | --- |
| 12-feature set | Average |
| training | **85.75%** |
| test | **82.72%** |



*How about when the data is not randomly split? Fill in each trial result and the average of trials in the table as shown follows.*

* The result of fixed split, with top 80% Testing, other 20% testing, is consistent

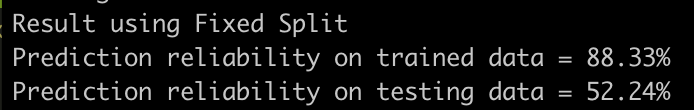
Run **b4Fixed.scala** file in b4 folder from spark shell.



|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Data Fixed Split | 1st trial | 2nd trial | 3rd trial | 4th trial | 4th trial | 4th trial | 4th trial | 4th trial | 4th trial | 10th trial | average |
| Training (TF-IDF) | 87.53% | 87.53% | 87.53% | 87.53% | 87.53% | 87.53% | 87.53% | 87.53% | 87.53% | 87.53% | 87.53% |
| Testing  (TF-IDF) | 52.17% | 52.17% | 52.17% | 52.17% | 52.17% | 52.17% | 52.17% | 52.17% | 52.17% | 52.17% | 52.17% |

* The result offixed split, with top 20% testing, other 80% training, is consistent

Run **b4Fixed-Switched.scala** file in b4 folder from spark shell.



|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Data Fixed Split | 1st trial | 2nd trial | 3rd trial | 4th trial | 4th trial | 4th trial | 4th trial | 4th trial | 4th trial | 10th trial | average |
| Training (TF-IDF) | 88.33% | 88.33% | 88.33% | 88.33% | 88.33% | 88.33% | 88.33% | 88.33% | 88.33% | 88.33% | 88.33% |
| Testing  (TF-IDF) | 52.24% | 52.24% | 52.24% | 52.24% | 52.24% | 52.24% | 52.24% | 52.24% | 52.24% | 52.24% | 52.24% |

#### 

For randomly split, we get quite good prediction rate on both training and testing data. However, it has poor predict ratio of testing data in above two fixed splits. The reason might because, the useful sample is randomly distributed. By randomly selecting the training data, the model learn the pattern well. But it could **not** learn all useful information by fixed split.

#### B.5 Prediction Rate with Puppy and Trainer : numeric and text/sentence based

*With the transformed trainer info and selected puppy info from both text and numeric columns, build/train a model before applying it for your prediction.*

**Software** : Scala (2.11.8), **Tool** : Spark (2.1.0)

**Platform** used: Ubuntu

**Executing the code :**

* Run **b5/TextNormalizerb5.scala** file from spark shell.

**Data Used** to run the code : **b5/datasets/b5dataset.csv**

We have selected below columns from both Trainer and PuppyInfo :

**"dog\_SubStatusCode", "StoolFirm", "EnergyLevel", "RespondsToCommandKennel", "GoodWStrangers", "BehavesWellClass", "RaidsGarbage", "GoodAppetite", "FoodType", "ExerciseType", "Sex", "Breed", "DayInLife"**

* Above columns include numeric, text and sentences type.
* We tried other columns as well, but above columns gave us best prediction rate.

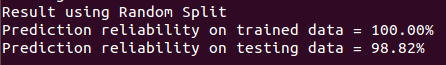
*Do you see consistent prediction rate for the same feature set of data when it is randomly split?*

* We got consistent result throughout .
* We got best result for this final dataset prepared.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 12-feature set | 1st trail | 2nd trial | 3rd trial | 4th trial | 5th trial |
| training | 100.00% | 100.00% | 100.00% | 100.00% | 100.00% |
| test | 98.82% | 98.82% | 98.82% | 98.82% | 98.82% |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 12-feature set | 6th trail | 7th trial | 8th trial | 9th trial | 10th trial |
| training | 100.00% | 100.00% | 100.00% | 100.00% | 100.00% |
| test | 98.82% | 98.82% | 98.82% | 98.82% | 98.82% |

|  |  |
| --- | --- |
| 12-feature set | Average |
| training | **100.00%** |
| test | **98.82%** |



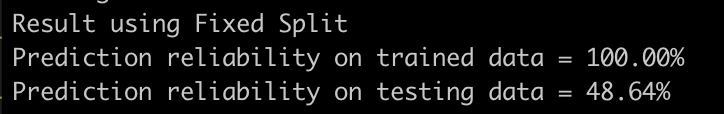
*How about when the data is not randomly split?*

**Software** : Scala (2.11.8), **Tool** : Spark (2.1.0)

**Platform** used: MAC OS

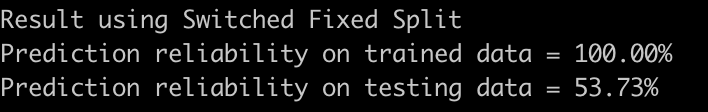
* The result of fixed split, with top 80% Testing, other 20% testing, is consistent

Run **b5Fixed.scala** file in b5 folder from spark shell.



|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Data Fixed Split | 1st trial | 2nd trial | 3rd trial | 4th trial | 4th trial | 4th trial | 4th trial | 4th trial | 4th trial | 10th trial | average |
| Training (TF-IDF) | 100.00% | 100.00% | 100.00% | 100.00% | 100.00% | 100.00% | 100.00% | 100.00% | 100.00% | 100.00% | 100.00% |
| Testing  (TF-IDF) | 48.64% | 48.64% | 48.64% | 48.64% | 48.64% | 48.64% | 48.64% | 48.64% | 48.64% | 48.64% | 48.64% |

* The result offixed split, with top 20% testing, other 80% training, is consistent

Run **b5Fixed-Switched.scala** file in b5 folder from spark shell.  


|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Data Fixed Split | 1st trial | 2nd trial | 3rd trial | 4th trial | 4th trial | 4th trial | 4th trial | 4th trial | 4th trial | 10th trial | average |
| Training (TF-IDF) | 100.00% | 100.00% | 100.00% | 100.00% | 100.00% | 100.00% | 100.00% | 100.00% | 100.00% | 100.00% | 100.00% |
| Testing  (TF-IDF) | 53.73% | 53.73% | 53.73% | 53.73% | 53.73% | 53.73% | 53.73% | 53.73% | 53.73% | 53.73% | 53.73% |

For randomly split, we get quite good prediction rate on both training and testing data. However, it has poor predict ratio of testing data in above two fixed splits. The reason might because, the useful sample is randomly distributed. By randomly selecting the training data, the model learn the pattern well. But it could **not** learn all useful information by fixed split.

# 

# 

# REFERENCE

1. Logistic Regression tutorial in Apache Spark:

<http://technobium.com/logistic-regression-using-apache-spark/>

1. Apache Spark documentation for MLlib:

<https://spark.apache.org/docs/2.1.0/mllib-data-types.html>

1. Machine Learning course by Andrew Ng:  
   <https://www.coursera.org/learn/machine-learning/lecture/wlPeP/classification>
2. Logistic Regression Tutorial for Machine Learning<http://machinelearningmastery.com/logistic-regression-tutorial-for-machine-learning/>
3. Feature Selection For Machine Learning in Python  
   <http://machinelearningmastery.com/feature-selection-machine-learning-python/>