BUAN 6341 APPLIED MACHINE LEARNING ASSIGNMENT 1

Project Report
Linear and logistic regression & Experimentations

By Animesh Kansal (axk169531)

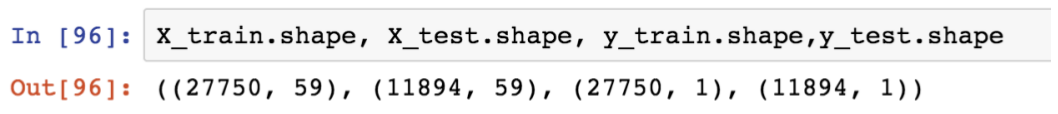
News Sharing dataset available for download at Dataset
I have used Python as scripting language

Tasks:

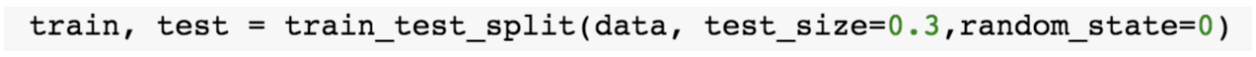
1. Divide the dataset into train and test sets sampling randomly. Use only predictive attributes and the target variable (do not use non-predictive attributes).

from sklearn.model_selection import train_test_split

I have divided the dataset into train and test sets sampling randomly by importing Scikit-learn's train_test_split library. I have split the train and test into 70/30 ratio. So the resulting X_train , X_test,y_train and y_test has shape as following, (we have dropped the 2 columns 'url' & ' timedelta':



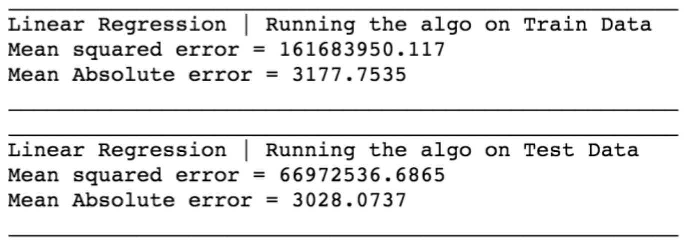
```
In [96]: X_train.shape, X_test.shape, y_train.shape,y_test.shape
Out[96]: ((27750, 59), (11894, 59), (27750, 1), (11894, 1))
```



```
train, test = train_test_split(data, test_size=0.3,random_state=0)
```

2. Use linear regression to predict the number of shares. Report and compare your train and test error/accuracy metrics. You can pick any metrics you like (e.g. mean squared error, mean absolute error, etc.)

I've created Linear regression on my own and created a cost function and gradient descent for the same, these are my findings, when run on all features:



```
Linear Regression | Running the algo on Train Data
Mean squared error = 161683950.117
Mean Absolute error = 3177.7535

Linear Regression | Running the algo on Test Data
Mean squared error = 66972536.6865
Mean Absolute error = 3028.0737
```

3. Convert this problem into a binary classification problem. The target variable should have two values (large or small number of shares).

I've converted this Linear regression problem into a binary classification problem. Now shares has value 1 or 0 and I divided 1 and 0 such that they both are 50-50 (used median as the parameter)

```
median = data[' shares'].median()
data[' shares'] = np.where(data[' shares']>=median, 1, 0)

# 1: represents Large
# 0: represents Small
```

4. Implement logistic regression to carry out classification on this data set. Report accuracy/error metrics for train and test sets.

I've created Logistic regression on my own and created a cost function and gradient descent for the same, Used scipy.optimize.fmin_tnc to minimize the cost function of logistic regression

scipy.optimize.fmin_tnc : Doc String:

Minimize a function with variables subject to bounds, using

gradient information in a truncated Newton algorithm. This

method wraps a C implementation of the algorithm.

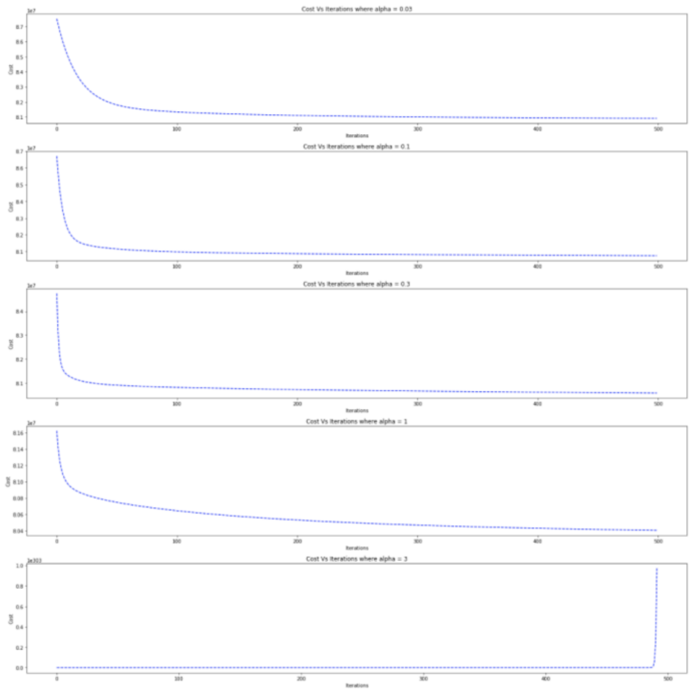
These are my findings, when run:

```
________________________________________________
Logistic Regression | Running the algo on Test set
Accuracy is 64.32 %
Confusion Matrix =
[[3232 2353]
 [1891 4418]]
Mean Absolute error = 0.3568
________________________________________________
________________________________________________
Logistic Regression | Running the algo on Train set
Accuracy is 65.81 %
Confusion Matrix =
[[ 7820  5085]
 [ 4404 10441]]
Mean Absolute error = 0.3419
________________________________________________
```

Experimentation:

- ***Experiment 1* - Changing learning rate (alpha)**

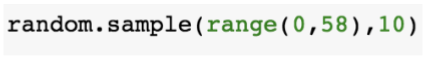
I have plotted cost vs Iterations using different learning rates:



As we can see from these plots, as alpha increases, the curve becomes steeper and able to reach lowest cost in less iterations but as alpha becomes greater than 1, cost goes to infinity because of too high alpha.

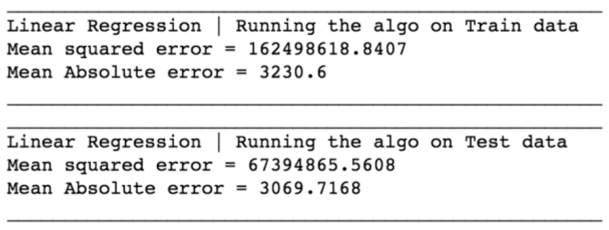
- ***Experiment 2* - Implement linear regression with 10 random features**

I have used random 10 numbers using random sample function



```
random.sample(range(0,58),10)
```

These are my findings:



```
______________________________________________________
Linear Regression | Running the algo on Train data
Mean squared error = 162498618.8407
Mean Absolute error = 3230.6
______________________________________________________
______________________________________________________
Linear Regression | Running the algo on Test data
Mean squared error = 67394865.5608
Mean Absolute error = 3069.7168
______________________________________________________
```

- ***Experiment 3* – Parameters that I think are best suited to predict the output**

I feel below mentioned 10 features will be able to help us predict the output better.

Because I think it matters that if weekend is there, people have more time and people will share more, Also Number of words in the title, If too big title, people might avoid the article, Number of images and videos so as to keep them interested and subjectivity and sentiment is very important as per current affairs. If content is positive, then people will feel like sharing it.

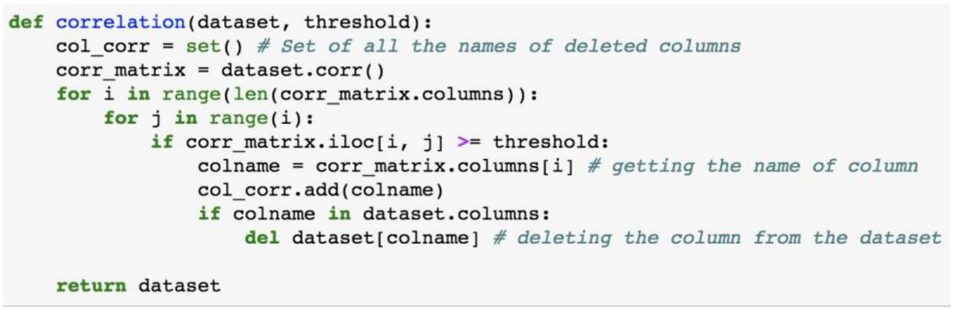
' is_weekend': Was the article published on the weekend?
' n_tokens_title': Number of words in the title
' num_imgs': Number of images
' num_videos': Number of videos
' global_subjectivity': Text subjectivity
' global_sentiment_polarity': Text sentiment polarity
' global_rate_positive_words': Rate of positive words in the content
' title_subjectivity': Title subjectivity
' title_sentiment_polarity': Title polarity
' abs_title_subjectivity': Absolute subjectivity level

These are my findings after running the linear regression:

```
____________________________________________________
Linear Regression | Running the algo on Train Data
Mean squared error = 163613361.021
Mean Absolute error = 3267.2118
____________________________________________________
____________________________________________________
Linear Regression | Running the algo on Test Data
Mean squared error = 68073309.3791
Mean Absolute error = 3112.5479
____________________________________________________
```

- ***Experimentation 4* - Drop Highly Correlated Features (greater than 0.75)**

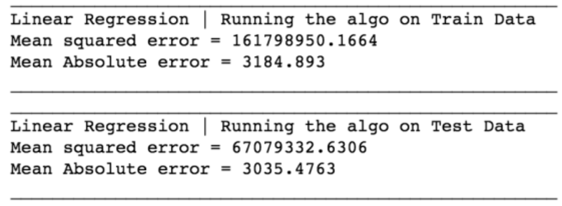
Multicollinearity exists whenever an independent variable is highly correlated with one or more of the other independent variables in a multiple regression equation. Multicollinearity is a problem because it undermines the statistical significance of an independent variable**.**
For these reason, we need to remove highly correlated features and it effects a regression model if you are having highly correlated features.



```python
def correlation(dataset, threshold):
    col_corr = set() # Set of all the names of deleted columns
    corr_matrix = dataset.corr()
    for i in range(len(corr_matrix.columns)):
        for j in range(i):
            if corr_matrix.iloc[i, j] >= threshold:
                colname = corr_matrix.columns[i] # getting the name of column
                col_corr.add(colname)
                if colname in dataset.columns:
                    del dataset[colname] # deleting the column from the dataset

    return dataset
```

These are my findings:



```
Linear Regression | Running the algo on Train Data
Mean squared error = 161798950.1664
Mean Absolute error = 3184.893
___________________________________________________
___________________________________________________
Linear Regression | Running the algo on Test Data
Mean squared error = 67079332.6306
Mean Absolute error = 3035.4763
___________________________________________________
```

Final Words:

These are the scenarios and are rated best to worst:

1) **Linear regression all features:**
```
Mean Absolute Error Train Data = 3177.7535
Mean Absolute Error Test Data = 3028.0737
```

2) **Linear regression when dropped Highly Correlated Features:**
```
Mean Absolute Error Train Data = 3184.893
Mean Absolute Error Test Data = 3035.4763
```

3) **Linear regression with random 10 features:**
```
Mean Absolute Error Train Data = 3230.6
Mean Absolute Error Test Data = 3069.7168
```

4) **Linear regression with selected 10 features:**
```
Mean Absolute Error Train Data = 3267.2118
Mean Absolute Error Test Data = 3112.5479
```

Logistic regression results:
```
Mean Absolute error = 0.3568
Accuracy is 64.32 %
```

What do you think matters the most for predicting the number and class of shares?

Every problem in Machine Learning cannot be solved with just one learning algorithm. We need to apply different learning algorithms depending on whether the relationship between the independent variables and the predictor variable is linear or non-linear. As we have seen in our analysis, Linear Regression and Classification did not have plausible accuracy for this particular dataset. There might be certain nonlinear relationships existing between the independent and the target variables in this dataset which might be better captured by learning algorithms like the neural network which are better suited to modeling non-linear relationships. Hence the most important thing that matters is the learning algorithm chosen.

What other steps you could have taken with regards to modeling to get better results?

I could have handled this problem would by first trying to reduce the dimensionality of the feature vectors by using PCA, and then use neural networks to model the predictions.