School of Engineering and Applied Science (SEAS), Ahmedabad University

B.Tech (CSE Semester VI)/M.Tech/PhD:
Machine Learning (CSE 523)

Project Submission #2: Linear Regression

Submission Deadline: March 04, 2020 (11:59 PM)

- **Group No.: 5**
- **Project Area: Natural Language Processing**
- **Project Title: Harnessing Twitter 'Big Data' for Automatic Emotion Identification**
- **Name of the group members :**
  1. Jay Patel(AU1741018)
  2. Manav Shah(AU1741042)
  3. Prima Sanghvi(AU1741045)
  4. Priyanshi Deliwala(AU1741047)

1. **Logistic regression:** Logistic regression is an instance of supervised classification in which the correct label y (either 0 or 1) is known for each observation x. There are two components of logistic regression: first is the metric of the distance between the current label $\hat{y}$ is to the original label y. This distance is loss function or cost function, for logistic regression it is commonly called cross-entropy function. Second is the optimization algorithm for updating the weights to minimize the loss function. The algorithm used for this is gradient descent.

   Cross-entropy loss function:
   $L(\hat{y}, y)$ = how much $\hat{y}$ differs from true value of y
   Loss function prefers the correct class labels of the training examples to be more likely. It is called conditional maximum likelihood estimation: the parameters are w,b that maximize the log probability of the true y labels in the training data given the observations x. The resulting loss function is the negative log likelihood loss, generally called the cross-entropy loss.
   Let us consider, $\hat{y} = P(y = 1/x)$
   $\hat{y}$ is the probability that y=1,given
   $1 - \hat{y} = P(y = 0/x)$
   $p(y/x) = \hat{y}^y(1 - \hat{y})^{1-y}$
   $y = 1 \implies P(y/x) = \hat{y}$
   Taking log likelihood ;
   $log(\hat{y}^y * (1 - \hat{y})^{(1-y)})$
   $ylog\hat{y} + (1 - y)log(1 - \hat{y})$
   $-L(\hat{y}, y)$

$logP(y/x) = -L(\hat{y}, y)$
The negative log likelihood is to maximize the probability by minimizing loss function.

Multi-class Logistic Regression :
The principle underlying logistic-regression doesn't change but increasing the classes means that we must calculate odds ratios for each of the K classes. Consider the odds-ratio for the binary case: We take the ratio of the probability of class A to the probability of the Kth class which would be the second class (B). K classes means we must now consider the calculated probability, p, of class i. Each class has its own specific vector of coefficients (represented as a vector of coefficients with a subscript signifying its class) :

$$\begin{aligned}
&\text{Class 1:} && ln\left(\frac{p_1}{p_K}\right) = \beta_1^T x \\
&\text{Class 2:} && ln\left(\frac{p_2}{p_K}\right) = \beta_2^T x \\
& && \vdots \\
&\text{Class K-1:} && ln\left(\frac{p_{K-1}}{p_K}\right) = \beta_{K-1}^T x \\
&\text{Class K:} && 1 - \sum_{i=1}^{K-1} p_i
\end{aligned}$$

Note that instead of just trying to fit one set of parameters, we now have (K-1) sets of variables which we are trying to fit! Going through the requisite algebra to solve for the probability values yields the equations shown below:

$$\begin{aligned}
&\text{Class 1:} && p_1 = \frac{e^{\beta_1^T x_i}}{1 + \sum_{i=1}^{K-1} e^{\beta_i^T x_i}} \\
&\text{Class 2:} && p_2 = \frac{e^{\beta_2^T x_i}}{1 + \sum_{i=1}^{K-1} e^{\beta_i^T x_i}} \\
& && \vdots \\
&\text{Class K-1:} && p_{K-1} = \frac{e^{\beta_{K-1}^T x_i}}{1 + \sum_{i=1}^{K-1} e^{\beta_i^T x_i}} \\
&\text{Class K:} && p_K = \frac{1}{1 + \sum_{i=1}^{K-1} e^{\beta_i^T x_i}}
\end{aligned}$$

Maximum Likelihood Function :
Since we now are using more than two classes the log of the maximum likelihood function becomes:

$$L(\beta_i^T, X) = \sum_{i=1}^{N} log\big(p_i(x_i, \beta_i^T)\big) = \sum_{i=1}^{N} log\Big(\frac{e^{\beta_i^T x_i}}{1 + e^{\beta_i^T x_i}}\Big)$$

The Gradient :
The derivation of the gradient of the maximum likelihood function below:

$$\begin{aligned}\frac{\partial}{\partial\beta}L(\beta, x_i) &= \sum_{i=1}^{N} \frac{\partial}{\partial\beta} y_i \beta^T x_i - \frac{\partial}{\partial\beta} ln(1 + e^{\beta^T x_i}) \\ &= \sum_{i=1}^{N} y_i x_i - \frac{x_i e^{\beta^T x_i}}{1 + e^{\beta^T x_i}} \\ &= \sum_{i=1}^{N} x_i (y_i - p_i)\end{aligned}$$

Turning this into a matrix equation is more complicated than in the two-class example — we need to form a N(K1)×(p+1)(K1) block-diagonal matrix with copies of X in each diagonal block matrix

2. **Implementation code:**

```
# importing libraries
import nltk
import pandas as pd
import numpy as np
import re
from gensim.models import Word2Vec
from io import StringIO
import matplotlib.pyplot as plt
from sklearn.feature_extraction.text import TfidfVectorizer, CountVectorizer, TfidfTransf
from sklearn.feature_selection import chi2
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import MultinomialNB
import seaborn as sns
from sklearn.svm import LinearSVC
from sklearn import metrics

# importing the stop words
stop_words = set(nltk.corpus.stopwords.words('english'))
```

```
# importing the sample dataset from Emotion_Phrases.csv
data = pd.read_csv("datasets/Emotion_Phrases.csv", header=None)

# adding the coloumns names
data.columns = ['emotions', 'text']

# getting the informations of data
data.info()

# converting the emotions to the ids
data['emo_id'] = data['emotions'].factorize()[0]

# print(data.head(10))
# print(data['emo_id'])

# dropped the duplicates and sorted it
emo_id_df = data[['emotions', 'emo_id']].drop_duplicates().sort_values('emo_id')

# converted it into a dictionary
emo_to_id = dict(emo_id_df.values)

# representing the dataset with a plot
fig = plt.figure()
data.groupby('emotions').text.count().plot.bar(ylim=0)
plt.show()

# converted the data into the dictionary
data_dict = data.to_dict()

sw_rem = []
txt = []

# preprocessed the database:
#       *   Removed the words other than A-Z and a-z and space
#       *   Tokenize the words
#       *   Convert the tokenized words into a list
for k in range(len(data_dict['text'])):
    data_dict['text'][k] = re.sub('([^A-Za-z ])', '', data_dict['text'][k])
    txt.append(nltk.tokenize.word_tokenize(data_dict['text'][k]))
    for j in range(len(txt[k])):
        if txt[k][j] not in stop_words:
            sw_rem.append(txt[k][j])
    data_dict['text'][k] = " ".join(sw_rem)
    sw_rem = []
data = pd.DataFrame.from_dict(data_dict)

# using tfidf vectorizer to convert the words into vectors as words that occur more frequ
tfidf = TfidfVectorizer(sublinear_tf=True, min_df=1, norm='l2', encoding='utf-8', ngram_r
```

```
l_data = data.text.tolist()
fitted_vectorizer=tfidf.fit(l_data)
features=fitted_vectorizer.transform(l_data)
labels = data.emo_id

# split train and test data
X_train, X_test, y_train, y_test, indices_train, indices_test = train_test_split(features

# instantiate the model
model = LinearSVC()

# train the model
model.fit(X_train, y_train)

# used the train model to predict the dataset
y_pred = model.predict(X_test)

#Sigmoid
def sigmoid(input):
        return 1/(1+np.exp(-input))

#Sigmoid Gradient
def sigmoid_grad(x):
        return x*(1-x)

# confusion matrix
conf_mat = metrics.confusion_matrix(y_test, y_pred)
fig, ax = plt.subplots(figsize=(8,6))
sns.heatmap(conf_mat, annot=True, fmt='d', xticklabels=emo_id_df.emotions.values, ytickla
plt.ylabel('Actual')
plt.xlabel('Predicted')
plt.show()

# generated an accuracy report based on the tests
print(metrics.classification_report(y_test, y_pred, target_names=data['emotions'].unique(

# print(clf.predict(count_vect.transform(["Angry Angry Angry Angry"])))
```

3. **URL links :** Datasets and Code links click here

4. **Implementation:**
We have implemented Logistic Regression to identify the emotions based on the phrases or tweets. Logistic Regression being a supervised machine learning it extracts real-value feature from the input. It helps to examine the relationship between one dependent variable and one or more nominal independent variable(joy, fear guilt etc.) comprising of both discrete and continuous variables. It ensures the determination of risk factors as probability is a method that investigates the relationship of the result variables with the independent variables. The method is an alternative to the linear regression as the normality assumption fails in case of

binary categorical or multi-categorical discrete variable. It plays an important role that the model from implementations is mathematically flexible and can be easily interpreted, and results in meaningful implementations.

We have implementing the logistic regression of some dependent variable y on the set of independent variables x_i= (x, ..., x), where $x_i$is the number of predictors(or inputs). In our project we have at first,imported **NumPy** given the choice of **NumPy** and **Pandas**. Imported the respective dataset and represented the same with the bar graph. In the preprocessing part of the our logistic regression, we majorly did three exercises:
1.Removed the words other than A-Z and a-z and space
2.Tokenize the words
3. Convert the tokenized words into a list.
We then converted the dataset into vectors using **TfidfVectorizer** and then split them into the test and train dataset. After the selection of the model-**Linear SVM** we trained the model and predicted and it resulted in 57 percent accuracy. As the data consisted of more than 2 classes, softmax regression was used. Confusion matrix and an aggregated report were generated for the same.

Results :

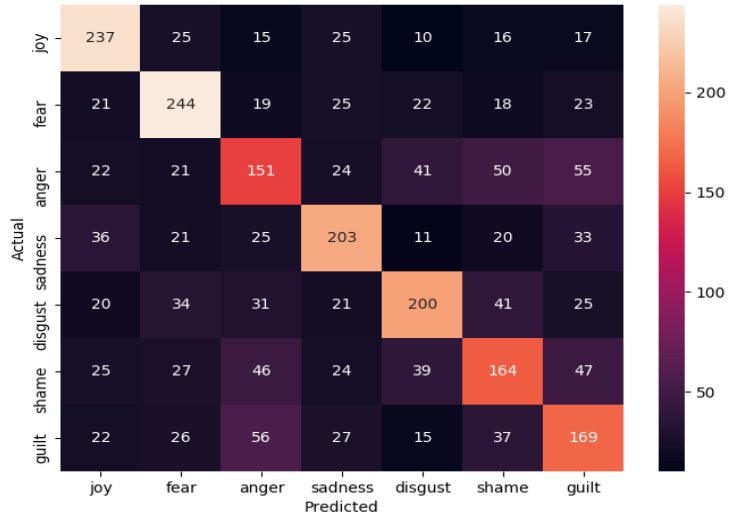

Figure 1: Confusion Matrix

Inference :
The confusion matrix is a table test which describes the performance of the classification model on the test data for which the true values are already known, in order to confusion to evaluate the model. For example: In our model you can see the shaded which represents the actual true positive predictive value of joy with joy and the region with the shade of black represents the false negative- where the predicted value was fear but the actual value was joy.

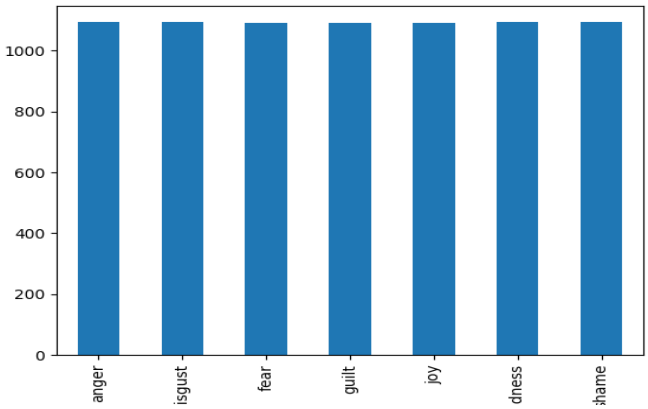

Figure 2: Dataset distribution

```
RangeIndex: 7652 entries, 0 to 7651
Data columns (total 2 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   emotions  7652 non-null   object
 1   text      7652 non-null   object
dtypes: object(2)
memory usage: 119.7+ KB
              precision    recall  f1-score   support

         joy       0.62      0.69      0.65       345
        fear       0.61      0.66      0.63       372
       anger       0.44      0.41      0.43       364
     sadness       0.58      0.58      0.58       349
     disgust       0.59      0.54      0.56       372
       shame       0.47      0.44      0.46       372
       guilt       0.46      0.48      0.47       352

    accuracy                           0.54      2526
   macro avg       0.54      0.54      0.54      2526
weighted avg       0.54      0.54      0.54      2526
```

Figure 3: Accuracy

References

[1] W. Wang, L. Chen, K. Thirunarayan and A. P. Sheth .*W. Wang, L. Chen, K. Thirunarayan and A. P. Sheth, "Harnessing Twitter "Big Data" for Automatic Emotion Identification," 2012 International Conference on Privacy, Security, Risk and Trust and 2012 International Confernece on Social Computing, Amsterdam, 2012, pp. 587-592.*

[2] G. Mishne, *"Experiments with mood classification in blog posts," in Proceedings of ACM SIGIR 2005 Workshop on Stylistic Analysis of Text for Information Access*

[3] Brownlee, Jason. *"A Gentle Introduction to Logistic Regression With Maximum Likelihood Estimation." Machine Learning Mastery, 27 Oct. 2019, machinelearningmastery.com/logistic-regression-with-maximum-likelihood-estimation/.*

[4] Usman Malik *Implementing PCA In Python With Scikit-Learn". Stack Abuse, 2020, https://stackabuse.com/implementing-pca-in-python-with-scikit-learn/.*