Pune Institute of Computer Technology
Department of Computer Engineering
Academic Year 2021-22

DATA MINING & WAREHOUSING
MINI PROJECT REPORT ON

COVID-19 Tweets Sentiment Analysis

Performed by
41247 Aakash Patil
41256 Saket Gupta

Under the guidance of
Prof. Manish Jansari

TABLE OF CONTENTS

1 OBJECTIVE

The objective of this mini project is to classify the sentiment a Tweet conveys about the COVID-19 pandemic into the categories Positive, Negative, and Neutral.

2 PROBLEM STATEMENT

Consider a labeled dataset belonging to an application domain. Apply suitable data preprocessing steps such as handling of null values, data reduction, discretisation. For prediction of class labels of given data instances, build classifier models using different techniques (minimum 3), analyse the confusion matrix and compare these models. Also apply cross validation while preparing the training and testing datasets.

3 OUTCOME

Three classifiers with varying accuracies were used to classify the tweets into Positive, Negative and Neutral sentiment labels. A GUI was developed to test the classifiers.

4.1 SOFTWARE REQUIREMENTS

UNIX/Linux OS, python3, tkinter, jupyter, pandas, numpy, sklearn, matplotlib, seaborn

4.2 HARDWARE REQUIREMENTS

64 bit machine with 8GB RAM, i5 or greater processor, 128GB SSD / 1TB HDD

5 THEORY

Classification is a type of supervised machine learning which solves the problem of identifying to which of a set of categories a new observation belongs, on the basis of a training set of data containing observations whose category membership is known.

The problem of Sentiment Analysis involves 3 categories (or labels) which means we will be applying multi-class (also known as multinomial) classification.

The **Bag-of-words** model was used during preprocessing. It is a simplifying representation used in natural language processing, where text is represented as a multi-set of its words, disregarding word order and grammar, but keeping its multiplicity. The frequency of each word is then used as a feature for training the classifier.

Instead of having just the frequency, a numerical statistic called **TF-IDF** (short for term frequency-inverse document frequency) is used. This is used to reflect how important a word is to a document in a collection. The tf-idf value for a word increases proportionally to the number of times a word appears in the document, and is offset by the number of documents in the collection that contain the word. This helps adjust the fact that some words appear more frequently in general.

The following algorithms were used to perform the classification; they are explained on the next page.

1. Multinomial Naive Bayes:

Naive Bayes classifiers are a family of simple, probabilistic classifiers based on Bayes' Theorem, which describes the probability of a certain event occurring based on prior knowledge of conditions that *might* be related to the event.

Mathematically, the theorem is:

$$P(A \mid B) = \frac{P(B \mid A) \,.\, P(A)}{P(B)}$$

where A, B are the events, P(A|B) is the conditional probability that the event A will occur knowing B is true, P(B|A) is the conditional probability that the event B will occur knowing A is true, and P(A), P(B) are marginal probabilities.

Naive Bayes is a technique for constructing classifiers, which *applies* the above theorem with the *strong (naive) assumption* that the features are largely independent (little to no correlation).

The model assigns the class labels (in this case, Positive, Negative, Neutral) to the problem instances. The model is actually a family of algorithms based on the common principle of the Naive Bayes classifier.

Multinomial Naive Bayes is used when there are more than 2 classification labels; samples (feature vectors) represent the frequencies with which certain events have been generated by k multinomials $(p_1, \ldots, p_n)$ where p_i is the probability that event i occurs. A feature vector is then a histogram, with x_i counting the number of times event i was observed in a particular instance.

2. Linear Support Vector Classification:

A Linear Support Vector Classifier plots each data item in the dataset as a point in n-dimensional space (where n is number of features) with the value of each feature being the value of a particular coordinate. Classification is performed by finding the hyperplane that differentiates the between the classes the best. The "best fit" hyperplane is determined by multiple methods; one such method is the so called function margin (maximising the distances between nearest data point and hyperplane) in which the larger the margin, the lower the generalization error of the classifier.

While SVM is typically a binary classification method, a multi-class classification problem can be solved by reducing it to multiple binary classification problems; this is made easy through sklearn's Linear SVC model.

Mathematically, if the dataset is represented as $(\overrightarrow{x_1}, y), \ldots, (\overrightarrow{x_n}, y)$ where the y_i are either 1 or -1, indicating which class the point $\overrightarrow{x_i}$ belongs.

The objective is to find a *maximum margin hyperplane* that divides the group of points $\overrightarrow{x_i}$ for which $y_i = 1$ from the group of points for which $y_i = -1$, defined such that the distance between the hyperplane and the nearest point $\overrightarrow{x_i}$ from either group is maximised.

3. Random Forest Classification:

Random forests are an ensemble learning method for classification that operate by constructing a multitude of decision trees at training time and outputting the class that is the *mode* of the classes.

They solve the overfitting problem that occurs when using Decision Tree classifiers by averaging multiple deep decision trees trained on different parts of the same training set, with the goal of reducing variance. This is called ***bagging*** *(bootstrap aggregating)*.

Given a training set $X = x_1, \ldots, x_n$with responses $Y = y_1, \ldots, y_n$ bagging repeatedly (B times) selects a *random sample with replacement* of the training set and fits trees to these samples:

For $b = 1, \ldots, B_n$

1. Sample, with replacement, n training examples from X, Y; call these X_b, Y_b.
2. Train a classification or regression tree f_b on X_b, Y_b.

After training, predictions for unseen samples x' can be made by taking the majority vote.

This method leads to better model performance because it decreases the variance of the model, without increasing the bias. This is because while the predictions of a single tree are highly sensitive to noise in its training set, the average of many trees is not.

This however might be affected by correlation among the trees, so Random Forests use a modified tree learning algorithm that selects, at each candidate split in the learning process, a random subset of the features *(feature bagging).*

6 CODE SCREENSHOTS

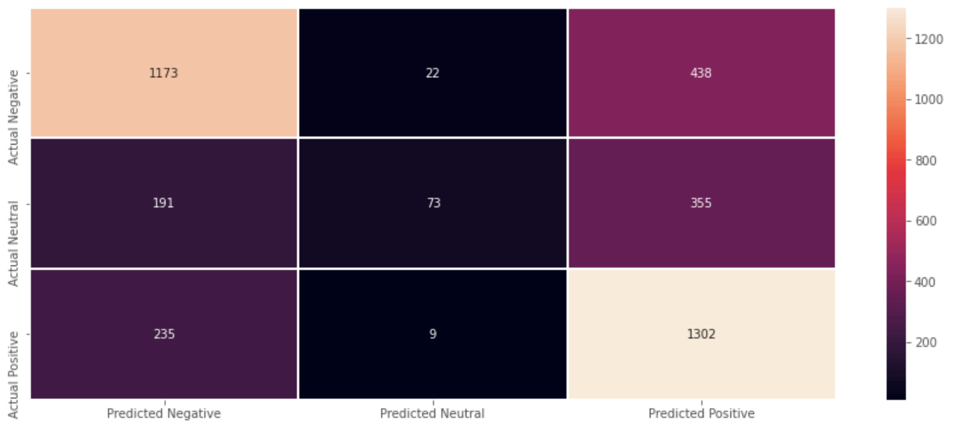
Classifier #1: Multinomial Naive Bayes

```
nb = MultinomialNB()
nb.fit(train_tfidf, train_data.label)
nb_model = nb.predict(test_tfidf)

accuracy_score(test_file.label, nb_model)
```

```
0.6708794102159031
```

```
nb_conf = confusion_matrix(test_file.label, nb_model)
ylabel = ["Actual Negative","Actual Neutral", "Actual Positive"]
xlabel = ["Predicted Negative","Predicted Neutral", "Predicted Positive"]
plt.figure(figsize=(15,6))
sns.heatmap(nb_conf, annot=True, xticklabels = xlabel, yticklabels = ylabel, linecolor='white', linewidths=1, fmt='g
```

```
<AxesSubplot:>
```



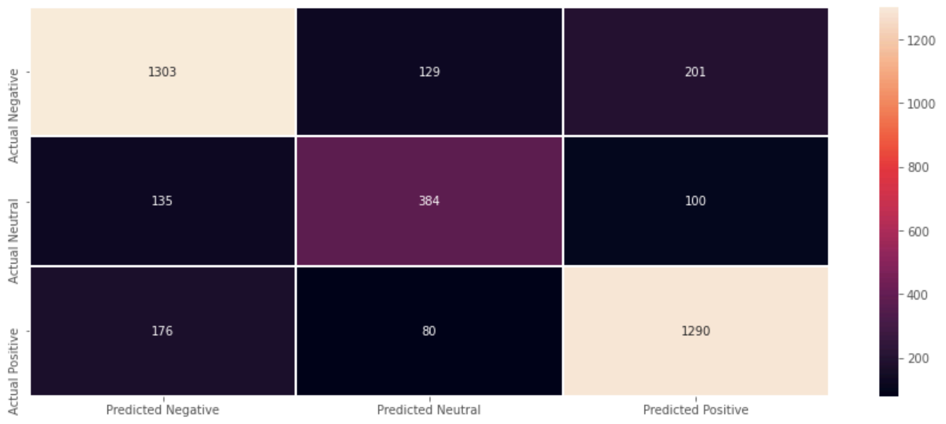
Classifier #2: Linear Support Vector

```
lsvc = LinearSVC()
lsvc.fit(train_tfidf, train_data.label)
lsvc_model = lsvc.predict(test_tfidf)

accuracy_score(test_file.label, lsvc_model)
```

```
0.7838335966298051
```

```
lsv_conf = confusion_matrix(test_file.label, lsvc_model)
ylabel = ["Actual Negative","Actual Neutral", "Actual Positive"]
xlabel = ["Predicted Negative","Predicted Neutral", "Predicted Positive"]
plt.figure(figsize=(15,6))
sns.heatmap(lsv_conf, annot=True, xticklabels = xlabel, yticklabels = ylabel, linecolor='white', linewidths=1, fmt='
```

```
<AxesSubplot:>
```



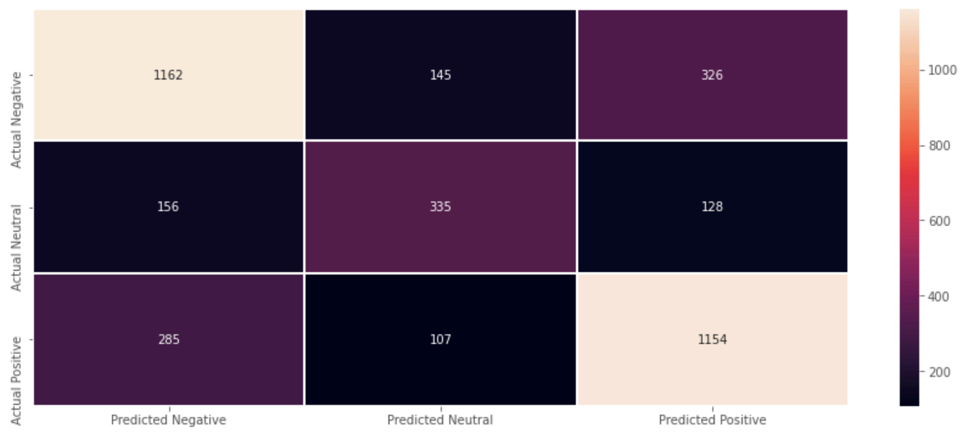
Classifier #3: Random Forest

```
rfc=RandomForestClassifier(n_estimators=100)
rfc.fit(train_tfidf, train_data.label)
rfc_model = rfc.predict(test_tfidf)

accuracy_score(test_file.label, rfc_model)
```

```
0.6979989468141127
```

```
rf_conf = confusion_matrix(test_file.label, rfc_model)
ylabel = ["Actual Negative","Actual Neutral", "Actual Positive"]
xlabel = ["Predicted Negative","Predicted Neutral", "Predicted Positive"]
plt.figure(figsize=(15,6))
sns.heatmap(rf_conf, annot=True, xticklabels = xlabel, yticklabels = ylabel, linecolor='white', linewidths=1, fmt='g
```

```
<AxesSubplot:>
```



```
from tkinter import *
from tkinter import filedialog
```

```
def switchHelper(argument):
    print(argument)
    switcher = {
        "0": "Negative",
        "1": "Neutral",
        "2": "Positive",
    }
    return switcher.get(argument, "Neutral")
```

```
def predictSentence(text):
    text = lower(text)
    text = remove_num(text)
    text = punct_remove(text)
    text = remove_stopwords(text)
    text = remove_mention(text)
    text = remove_hash(text)
    text = remove_space(text)

    tfidfVector = tfidf.transform([text])

    nb_pred_label = nb.predict(tfidfVector)
    return nb_pred_label
```

```
def predictSentenceHelper():
    pred = predictSentence(tweetInput.get())
    pred_class = switchHelper(pred[0])

    labelPredict = Label(root, text=pred_class)
    labelPredict.grid(row=6, column=1)
```

```python
# f = 'Corona_NLP_test.csv'
def predictFile(f):
    test_data = pd.read_csv(f,encoding='latin1')

    #Preprocess test file
    test_data['text'] = test_data.OriginalTweet
    test_data["text"] = test_data["text"].astype(str)
    test_data['label']=test_data['Sentiment'].apply(lambda x:classes_def(x))
    test_data['text_new']=test_data['text'].apply(lambda x:remove_urls(x))
    test_data['text']=test_data['text_new'].apply(lambda x:remove_html(x))
    test_data['text_new']=test_data['text'].apply(lambda x:lower(x))
    test_data['text']=test_data['text_new'].apply(lambda x:remove_num(x))
    test_data['text_new']=test_data['text'].apply(lambda x:remove_mention(x))
    test_data['text']=test_data['text_new'].apply(lambda x:remove_hash(x))
    test_data['text_new']=test_data['text'].apply(lambda x:remove_space(x))
    test_data = test_data.drop(columns=['text_new'])

    tfidf_file = tfidf.transform(test_data.text)
    nb_model = nb.predict(tfidf_file)
    model_acc = [accuracy_score(test_data.label, nb_model)*100]

    lsvc_model = lsvc.predict(tfidf_file)
    model_acc.append(accuracy_score(test_data.label, lsvc_model)*100)

    rfc_model = rfc.predict(tfidf_file)
    model_acc.append(accuracy_score(test_data.label, rfc_model)*100)

    return model_acc

# predictFile(f)
```

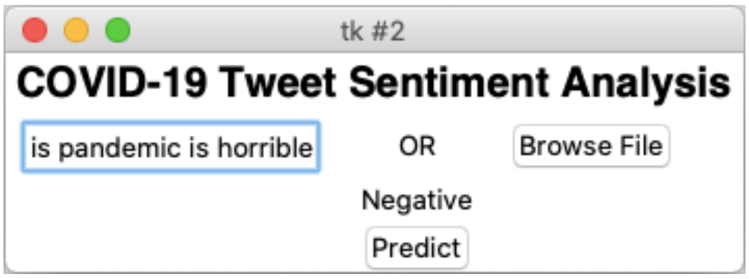
```python
def predictFileHelper():
    filename = filedialog.askopenfilename(initialdir="/env/LP2/DMW_Mini/", title="Select File")
    model_acc = predictFile(filename)

    show_confusion(model_acc)
```

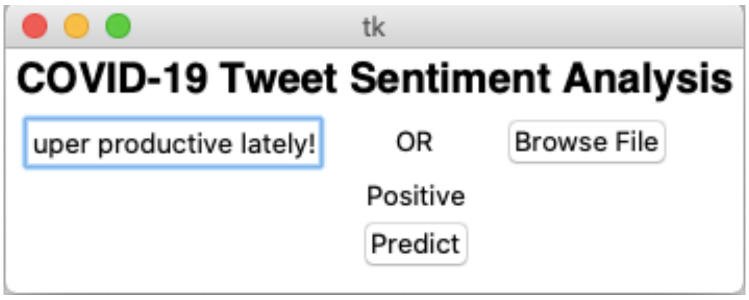
```python
def clear_frame():
    for w in root.winfo_children():
        w.destroy()
```

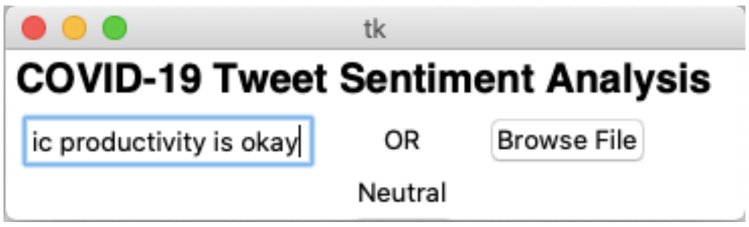
```python
def show_confusion(model_acc):
    clear_frame()

    labelTitle = Label(root, text='COVID-19 Tweet Sentiment Analysis', font='Helvetica 20 bold')
    labelTitle.grid(row=2, columnspan=5)

    Label(root, text='Naive Bayes').grid(row=3,column=1)
    Label(root, text=f'Accuracy: {str(round(model_acc[0],2))}%').grid(row=4,column=1 )
    Label(root, text=f'Confusion Matrix: ').grid(row=5,column=1 )
    Label(root, text=f'{nb_conf}').grid(row=6,column=1 )
    Label(root, text=f'Accuracy after Cross Validation: {str(round(cross_nb,2))}%').grid(row=7,column=1 )

    Label(root, text='').grid(column=2)

    Label(root, text='Linear Support Vector').grid(row=3,column=3)
    Label(root, text=f'Accuracy: {str(round(model_acc[1],2))}%').grid(row=4,column=3 )
    Label(root, text=f'Confusion Matrix: ').grid(row=5,column=3 )
    Label(root, text=f'{lsv_conf}').grid(row=6,column=3 )
    Label(root, text=f'Accuracy after Cross Validation: {str(round(cross_lsv,2))}%').grid(row=7,column=3 )

    Label(root, text='').grid(column=2)

    Label(root, text='Random Forest').grid(row=3,column=5)
    Label(root, text=f'Accuracy: {str(round(model_acc[2],2))}%').grid(row=4,column=5 )
    Label(root, text=f'Confusion Matrix: ').grid(row=5,column=5 )
    Label(root, text=f'{rf_conf}').grid(row=6,column=5 )
    Label(root, text=f'Accuracy after Cross Validation: {str(round(cross_rf,2))}%').grid(row=7,column=5 )
```

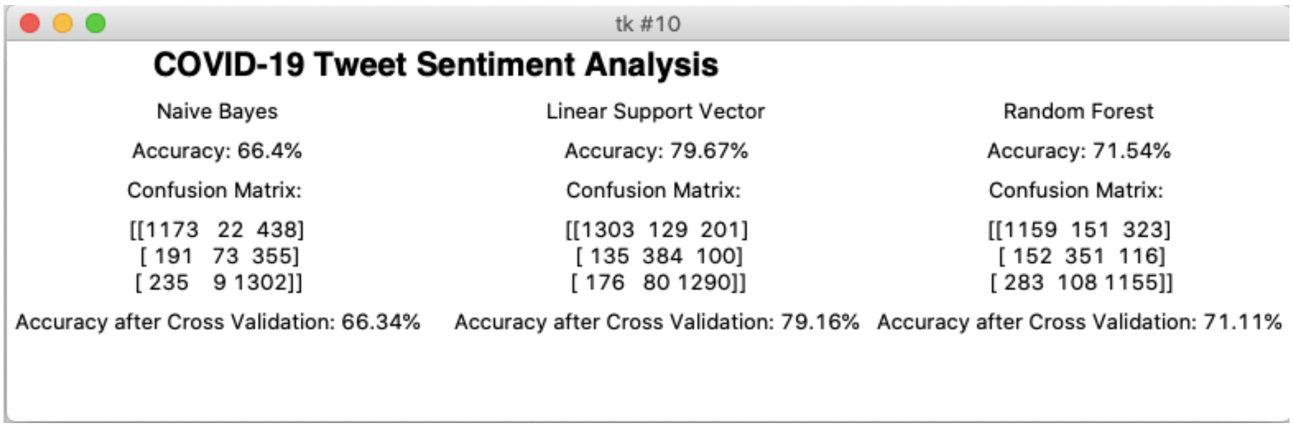
```
root = Tk()

#Custom Input

labelTitle = Label(root, text='COVID-19 Tweet Sentiment Analysis', font='Helvetica 20 bold')
labelTitle.grid(row=1, column=0,columnspan=5, rowspan=1)

tweetInput = Entry(root, width=15)
tweetInput.insert(0, 'Enter Tweet')

tweetInput.grid(row=4,column=0)

predictButton = Button(root, text="Predict", command=predictSentenceHelper)
predictButton.grid(row=7,column=1)

labelOR = Label(root, text="OR")
labelOR.grid(row=4, column=1)

# File Input
button_file_open = Button(root, text="Browse File", command=predictFileHelper)
button_file_open.grid(row=4,column=2)

root.mainloop()
```

7 OUTPUT SCREENSHOTS









```
tk #10
COVID-19 Tweet Sentiment Analysis
        Naive Bayes                      Linear Support Vector                 Random Forest
     Accuracy: 66.4%                       Accuracy: 79.67%                   Accuracy: 71.54%
    Confusion Matrix:                     Confusion Matrix:                  Confusion Matrix:
    [[1173   22  438]                     [[1303  129  201]                  [[1159  151  323]
     [ 191   73  355]                      [ 135  384  100]                   [ 152  351  116]
     [ 235    9 1302]]                     [ 176   80 1290]]                  [ 283  108 1155]]
Accuracy after Cross Validation: 66.34%  Accuracy after Cross Validation: 79.16%  Accuracy after Cross Validation: 71.11%
```

8 TEST CASES

INPUT	MODEL	ACCURACY / OUTPUT	CROSS VALIDATION ACCURACY
COVID-19 Tweets Dataset	Multinomial Naive Bayes	67%	66%
	Random Forest	69%	71%
	Linear Support Vector	78%	79%
This pandemic is horrible!	All	Negative	-
I've been super productive lately!	All	Positive	-
My WFH productivity is okay	All	Neutral	-
I hate covid man, this sucks	All	Negative	-
I'm so happy to be home!	All	Positive	-

9 COMPARISON OF CLASSIFIERS

Initially, after only performing model fitting, the accuracy scores for the 3 classifiers were obtained. Further, cross-validation was performed for all 3 classifiers, and the following results were obtained:

Model	Accuracy	Cross Validation Accuracy
Multinomial Naive Bayes	67.09%	66.33%
Linear Support Vector	78.38%	79.16%
Random Forest	69.72%	71.11%

The accuracy of Random Forest model as well as Linear Support Vector model improved with cross-validation, while the Multinomial Naive Bayes model showed a minor reduction in its accuracy.

It is obvious that the Linear Support Vector model outperforms the other two classifiers.

10 PERFORMANCE AND EXPLANATION

A labelled dataset of Tweets about the COVID-19 pandemic was obtained, having 41157 training entries and 3798 testing entries.

Originally the labels were: Extremely Positive, Positive, Neutral, Negative, and Extremely Negative. However, the classifiers performed poorly and the labels where reduced to Positive (2), Neutral (1) and Negative (0).

Preprocessing the dataset involved:
1. Removing URLs and HTML from the Tweets' text
2. Converting the Tweet text to lowercase
3. Removing numerical values from the Tweet text
4. Removing punctuation and stop words from the Tweet text, and finally
5. Removing hashtags (#), mentions (@) and extra spaces from the Tweet text.

Most classifiers work with *numerical*, not text data, so the TF-IDF method was used to convert the tweet text to a usable feature.

The Multinomial Naive Bayes, Linear Support Vector, and Random Forest models were then fitted using the training data, and predicted the labels for the given test data, with varying accuracies. To better understand the performance of the models, a heat map of the confusion matrix was made for each of the models.

Furthermore, cross-validation was performed on all 3 models using the combined training and testing data, which resulted in improved accuracy. The Linear Support Vector model was found to be the best performing model out of the 3 classifiers used.

A GUI was developed using tkinter, which enables the user to input a file or text to the models. The models are first trained, then used to make a prediction for the user input. For single tweet text inputs, the predicted class for the tweet is displayed. For file (multi-tweet input) the accuracy before and after cross validation, as well as the confusion matrix is displayed.

11 APPLICATIONS

This project has many further applications, some of which are listed below:

- Understanding trio of sentiments (fear-panic-despair) to effectively deploy public service announcements as well as motivational solutions and strategies.
- Corporations and small businesses can benefit through such analyses to better understand consumer sentiment and expectations.

12 CONCLUSION

Sentiment Analysis (Classification) for a given Tweet text was successfully performed using 3 different classifiers namely Multinomial Naive Bayes, Linear Support Vector, and Random Forest.

The different classification algorithms behind the methods were understood, along with the concept of cross-validation.

The Linear Support Vector model was found to be the most accurate model for classifying the tweets.