Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

School of Computer Science and Engineering

J Component Report

Programme : Integrated M.Tech CSE (Business Analytics)

Course Title : Natural Language Processing

Course Code : SWE1017

Slot : B2

Title : Detecting Signs of Depression from Social Media Text

Team Members : Adithya G | 19MIA1042

Varun Ragul R | 19MIA1085

Faculty: Dr. Sridhar R Sign:

SWE1017 – Natural Language Processing

J Component Report

Detecting Signs of Depression from Social Media Text

By

Varun Ragul.R 19MIA1085

Adithya Gopal 19MIA1042

Submitted to

Dr. Sridhar R

DECLARATION BY THE CANDIDATE

I hereby declare that the report titled "Detecting Signs of Depression from Social Media Text" submitted by Varun Ragul.R 19MIA1085, Adithya Gopal 19MIA1042 to VIT Chennai is a record of bona-fide work undertaken by me under the supervision of Dr. Sridhar. R, Vellore Institute of Technology, Chennai.

ACKNOWLEDGEMENT

We wish to express our sincere thanks and deep sense of gratitude to our project guide, Dr. Sridhar R for his consistent encouragement and valuable guidance offered to us throughout the course of the project work.

We are extremely grateful to Dr. R. Ganesan, Dean, School of Computer Science and Engineering, Vellore Institute of Technology, Chennai, for extending the facilities of the School towards our project and for his unstinting support.

We express our thanks to our Head of the Department for her support throughout the course of this project. We also take this opportunity to thank all the faculty of the School for their support and their wisdom imparted to us throughout the courses.

We thank our parents, family, and friends for bearing with us throughout the course of our project and for the opportunity they provided us in undergoing this course in such a prestigious institution.

ABSTRACT

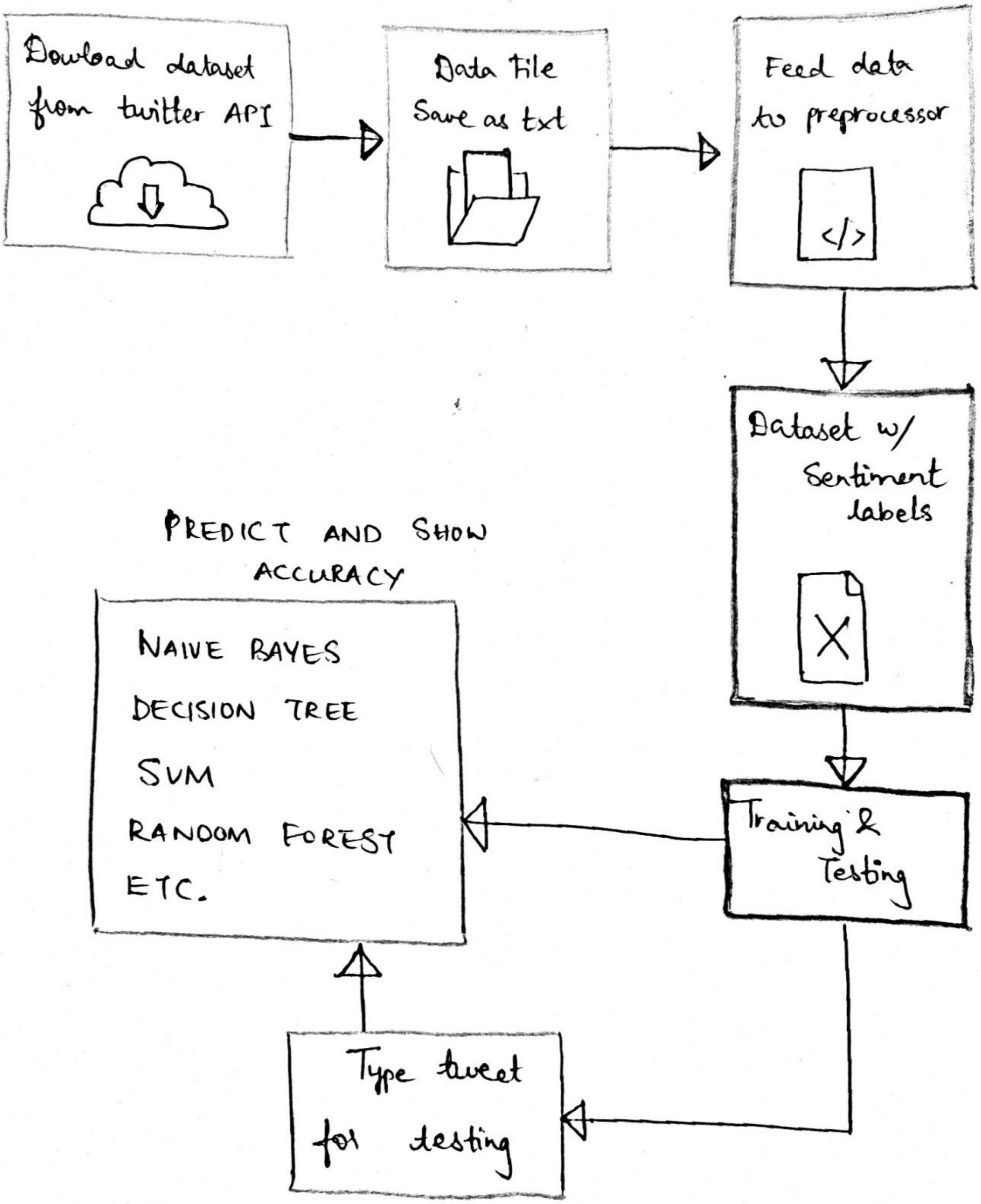
Depression is a constant feeling of sadness and loss of interest, which stops you doing your normal activities. Different types of depression exist, with symptoms ranging from relatively minor to severe. Generally, depression does not result from a single event, but from a mix of events and factors. Depression is a common mental illness that involves sadness and lack of interest in all daily activities. Detecting depression is important since it has to be observed and treated at an early stage to avoid severe consequences.

Social media is considered as a major part of the human culture, which affects the mental health of its user directly. The aim of this review is to detect the signs of depression of a person from their social media postings where people share their feelings and emotions and to identify studies that complement the assessment of mental illness with measures of well being.

PROBLEM STATEMENT

Our project aims to detect the signs of depression of a person from their social media postings where they've shared their feelings and emotions. We intend to classify the text provided in the dataset into different labels that indicate the levels of depression by training different deep learning models and comparing them based on results from development and training dataset.

The aim of the project is to predict early signs of depression through Social Media text mining. Below are the steps to run the python codes using the data sets uploaded in the repositories:


Dowload dataset from twitter API
Data File Save as txt
Feed data to preprocessor
</>
Dataset w/ Sentiment labels
Training & Testing
PREDICT AND SHOW ACCURACY
NAIVE BAYES
DECISION TREE
SVM
RANDOM FOREST
ETC.
Type tweet for testing

DATASET

The dataset comprises training, development and test set. The data files are in Tab Separated Values (tsv) format with three columns namely posting_id (pid), text data and label. The sample instances are as follows:

	Unnamed: 0	text	class
0	2	Ex Wife Threatening SuicideRecently I left my ...	severely depressed
1	3	Am I weird I don't get affected by compliments...	not depressed
2	4	Finally 2020 is almost over... So I can never ...	not depressed
3	8	i need helpjust help me im crying so hard	severely depressed
4	9	I'm so lostHello, my name is Adam (16) and I'v...	severely depressed

Reviewing dataset:

The dataset contains a collection of postings and comments from social media users in English and we are required to model a system based on the train data to classify the test and development data into two labels:

- Not Depressed
- Severly Depressed

The csv files have a target column set to 1 by default, and we manually set the non-depressive entries to have target of 0, and also removed non-English tweets from the file. The csv files contain roughly 50-50 split of depressive and non-depressive tweets. This is a good resource to train our model on, as all of the tweets originally had depressive hashtags, so by distinguishing the tweets based on its content rather than tags, we are training the model to be more sensitive to the content and more precise in its predictions.

As the dataset is made of English sentences and paragraphs, we are required to preprocess the text down into a format the computer can easily understand.

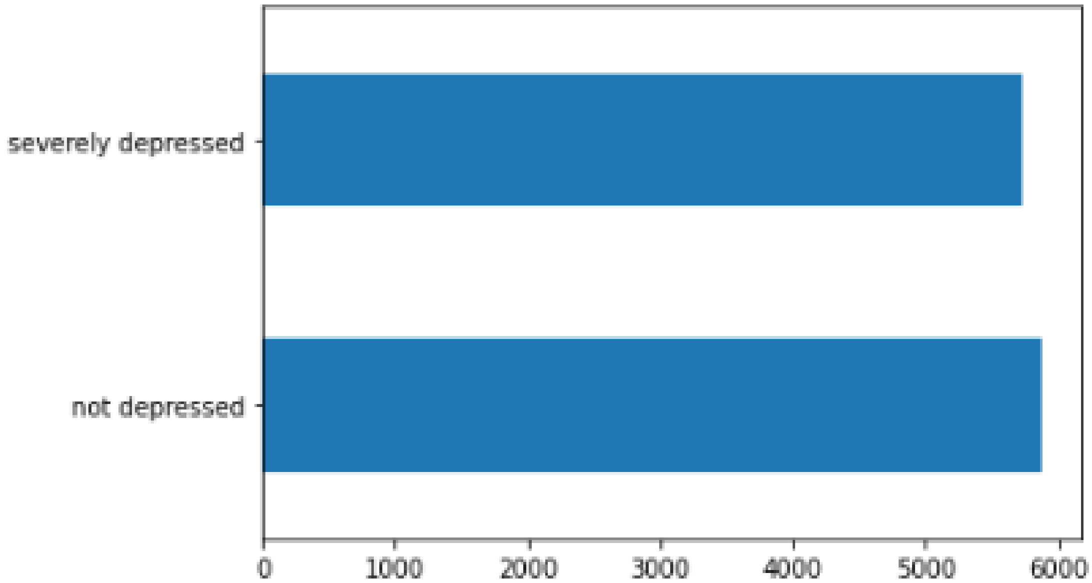
Missing Value Analysis:

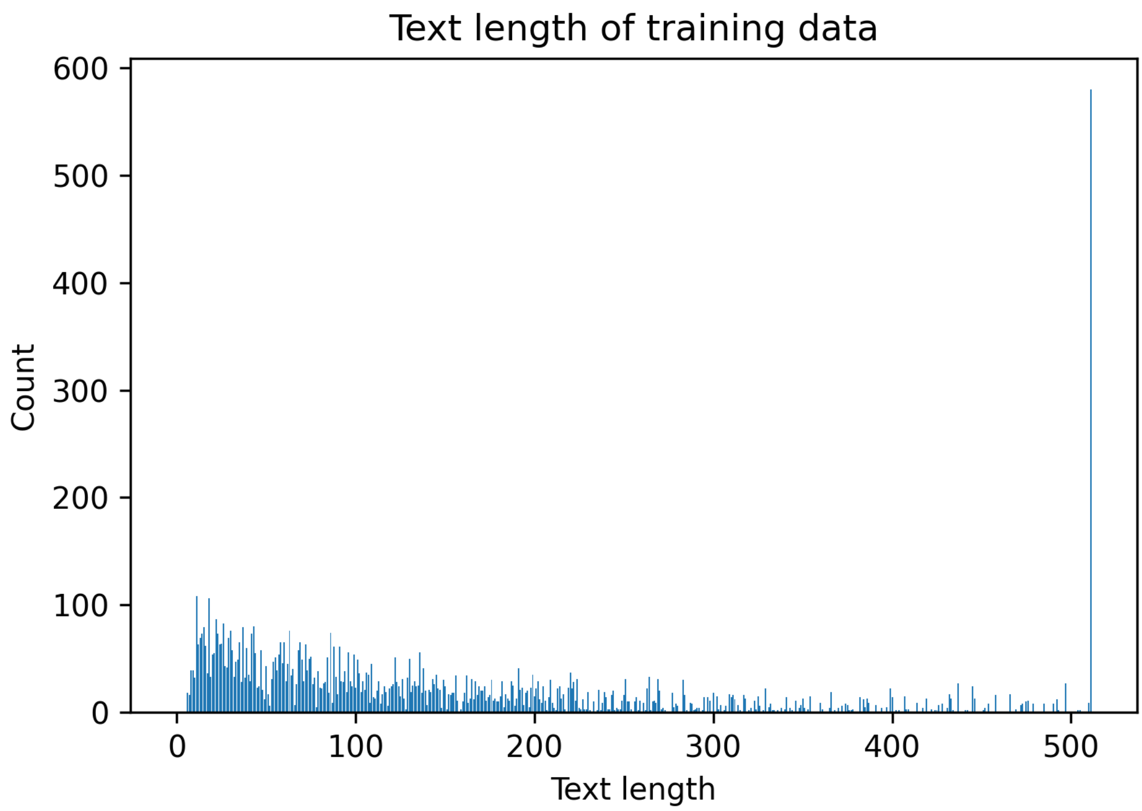
```
df_train = train.to_pandas()
df_valid = valid.to_pandas()
df_train.isnull().sum()
df_valid.isnull().sum()
```

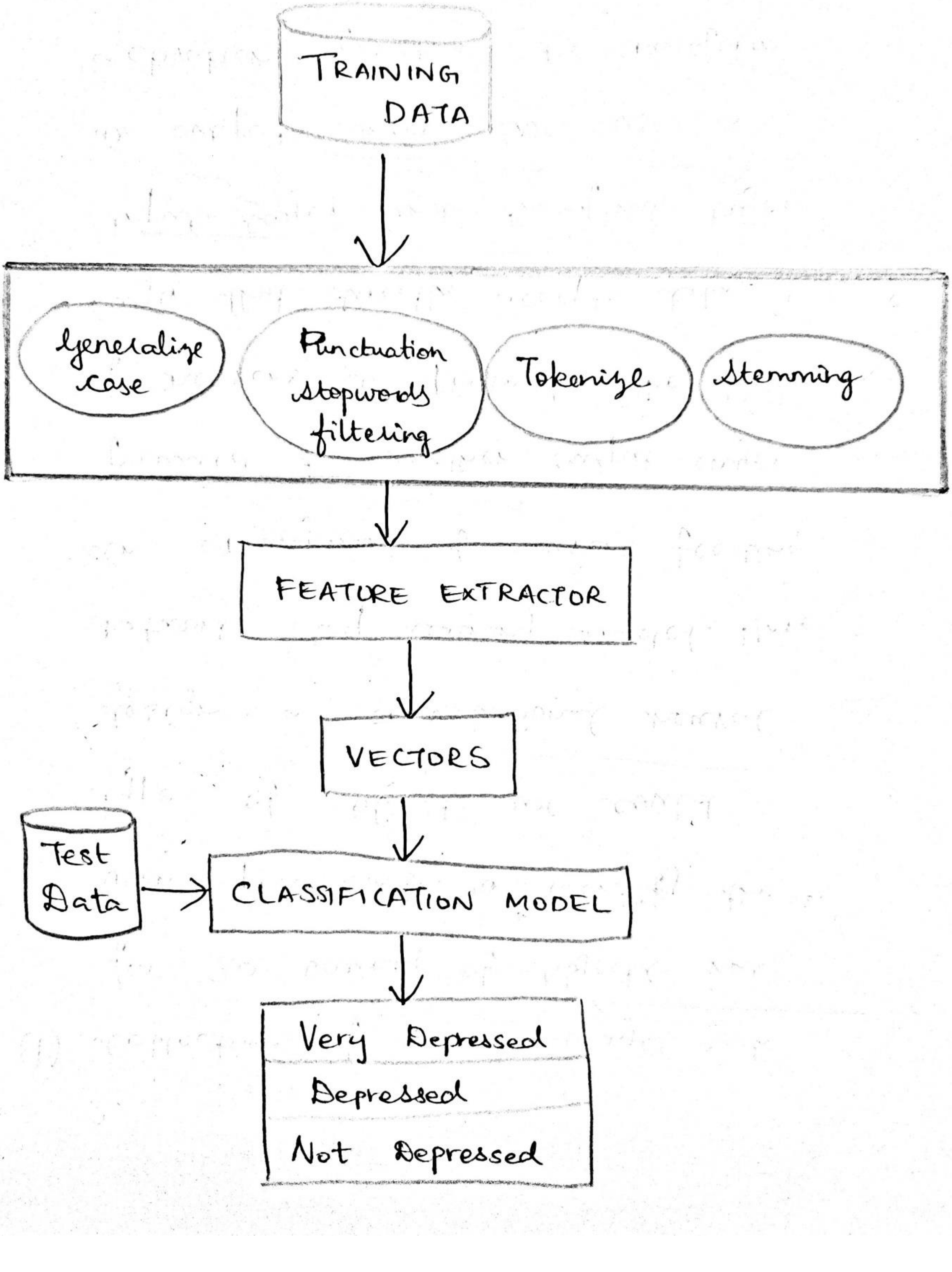
```
PID          0
Text_data    0
Label        0
dtype: int64
```

```
1 df.target.value_counts().plot.barh()
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f17f6c6e9d0>
```






TRAINING DATA
Generalize case
Punctuation stopwords filtering
Tokenize
Stemming
FEATURE EXTRACTOR
VECTORS
Test Data
CLASSIFICATION MODEL
Very Depressed
Depressed
Not Depressed

PROPOSED METHODOLOGY

First we perform text cleaning and preparation this involves:

- Generalizing Case, where we convert all letters to lower case
- Ignoring Punctuation, we can do this by keeping everything in the text that does not belong to string. punctuation (a pre initialized string that contains all sets of punctuations)
- Ignoring stop words that don’t provide any information. Here we import a list of most frequently used words from the nltk. corpus and remove them as they reduce the predictive power of the model.
- Comparing Stemming and Lemmatizing: Stemming, where we reduce words to their stem by cutting off prefixes and ending of words but sometimes the new word could lose its actual meaning.
- Lemmatizing maps common words into one base. Therefore we would like to compare the two algorithms to find which one works better for this dataset and apply it. This aims to remove inflectional endings only and to return the base or dictionary form of a word, which is known as the lemma.

```
def _preprocess_text(self, text):
    return self._lemmatize(self._leave_letters_only(self._clean(text)))

def _clean(self, text):
    bad_symbols = '!"#%&\'*+,-<=>?[\\]^_`{|}~'
    text_without_symbols = text.translate(str.maketrans('', '', bad_symbols))

    text_without_bad_words = ''
    for line in text_without_symbols.split('\n'):
        if not line.lower().startswith('from:') and not line.lower().endswith('writes:'):
            text_without_bad_words += line + '\n'

    clean_text = text_without_bad_words
    email_regex=r'([a-zA-Z0-9_.+-]+@[a-zA-Z0-9-]+\.[a-zA-Z0-9-.]+)'
    regexes_to_remove=[email_regex, r'Subject:', r'Re:']
    for r in regexes_to_remove:
        clean_text = re.sub(r,'', clean_text)

    return clean_text

def _leave_letters_only(self, text):
    text_without_punctuation = text.translate(str.maketrans('', '', string.punctuation))
    return ' '.join(re.findall("[a-zA-Z]+", text_without_punctuation))

def _lemmatize(self, text):
    doc = nlp(text)
    words = [x.lemma_ for x in [y for y in doc if not y.is_stop and y.pos_ != 'PUNCT'
                                  and y.pos_ != 'PART' and y.pos_ != 'X']]
    return ' '.join(words)

def fit(self, *_):
    return self
```

Natural language processing tasks require input text in numerical representation rather than raw words meaning that each word must be encoded in order to be processed by machines. The vectors are derived from the textual data so that they retain the various linguistic properties of the text. We will use :

- Scikit learn's Countvectorizer to transform a given text into a vector on the basis of the frequency of each word that occurs in the entire text

- TF-IDF which means Term Frequency Inverse Document Requency. This is based on the frequency of a word in the data but also provides a numerical representation of how important a word is for statistical analysis. We would compare these algorithms to choose the best one to work with.

When training classifiers on large collections of documents, both the time and memory requirements connected with processing of these vectors may be huge. This calls for using a feature selection method, not only to reduce the number of features but also to increase the sparsity of document vectors. We propose a feature selection method based on linear Support Vector Machines (SVMs).

Functions to print confusion matrix andmodel evaluation:

```
def print_confusion_matrix(confusion_matrix,
                           class_names,
                           figsize = (15,15),
                           fontsize=12,
                           ylabel='True label',
                           xlabel='Predicted label'):
    df_cm = pd.DataFrame(
        confusion_matrix, index=class_names, columns=class_names,
    )
    fig = plt.figure(figsize=figsize)
    try:
        heatmap = sns.heatmap(df_cm, annot=True, fmt="d")
    except ValueError:
        raise ValueError("Confusion matrix values must be integers.")
    heatmap.yaxis.set_ticklabels(heatmap.yaxis.get_ticklabels(), rotation=0, ha='right',
    heatmap.xaxis.set_ticklabels(heatmap.xaxis.get_ticklabels(), rotation=45, ha='right'
    plt.ylabel(ylabel)
    plt.xlabel(xlabel)
```

```python
def evaluate_model(model, X, y, X_test, y_test, target_names=None):
    scores = cross_val_score(model, X, y, cv=5, scoring='accuracy')
    scores_test = cross_val_score(model, X_test, y_test, cv=5, scoring='accuracy')

    print("Accuracy: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std()))
    print("Accuracy test: %0.2f (+/- %0.2f)" % (scores_test.mean(), scores_test.std()))

    print("Test classification report: ")
    if target_names is None:
        target_names = model.classes_
    print(classification_report(y_test, model.predict(X_test), target_names=target_names))
    print("Test confusion matrix: ")
    print_confusion_matrix(confusion_matrix(y_test, model.predict(X_test)), class_names=target_names)
```

Predictive Models with evaluation results:

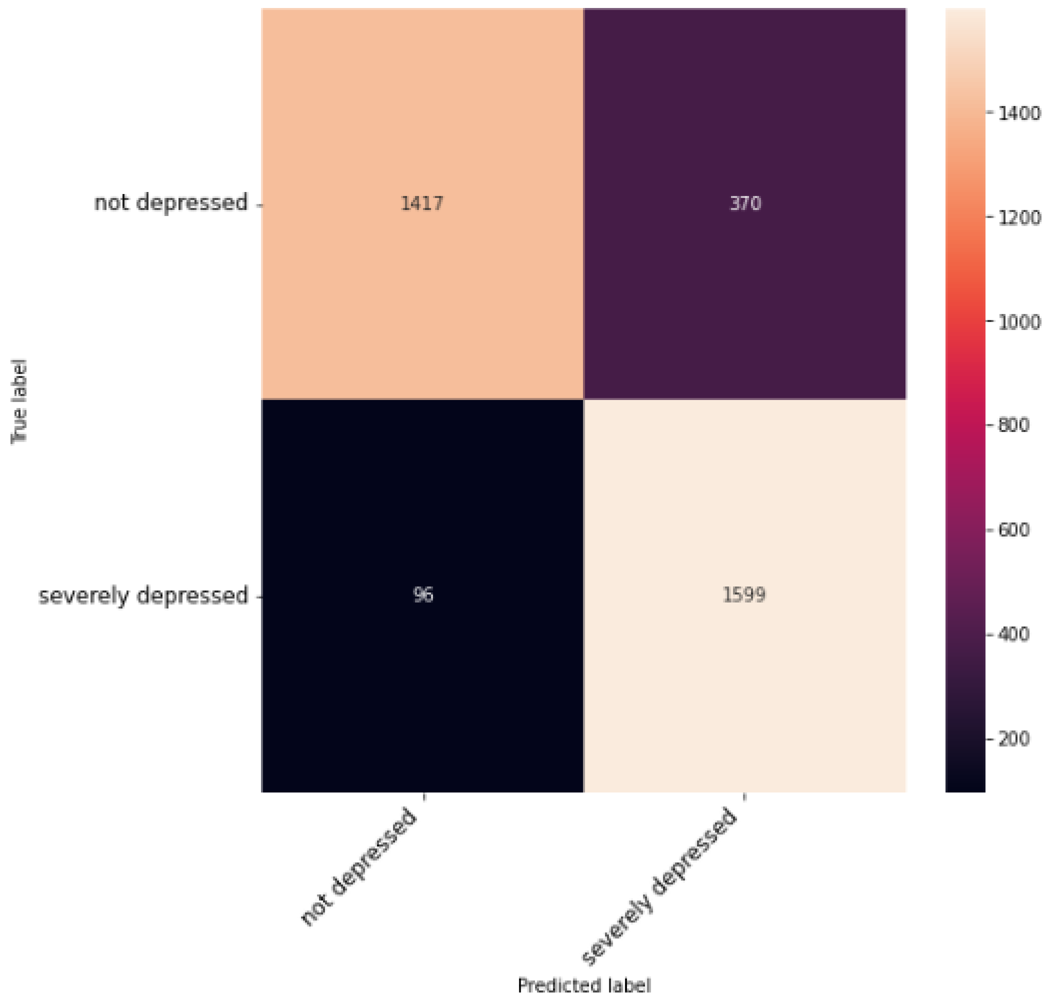
MultiNomial Naive Bayes

```
Accuracy: 0.90 (+/- 0.01)
Accuracy test: 0.85 (+/- 0.01)
Test classification report:
                    precision    recall  f1-score   support

    not depressed       0.94      0.79      0.86      1787
severely depressed       0.81      0.94      0.87      1695

         accuracy                           0.87      3482
        macro avg       0.87      0.87      0.87      3482
     weighted avg       0.88      0.87      0.87      3482
```

Test confusion matrix:



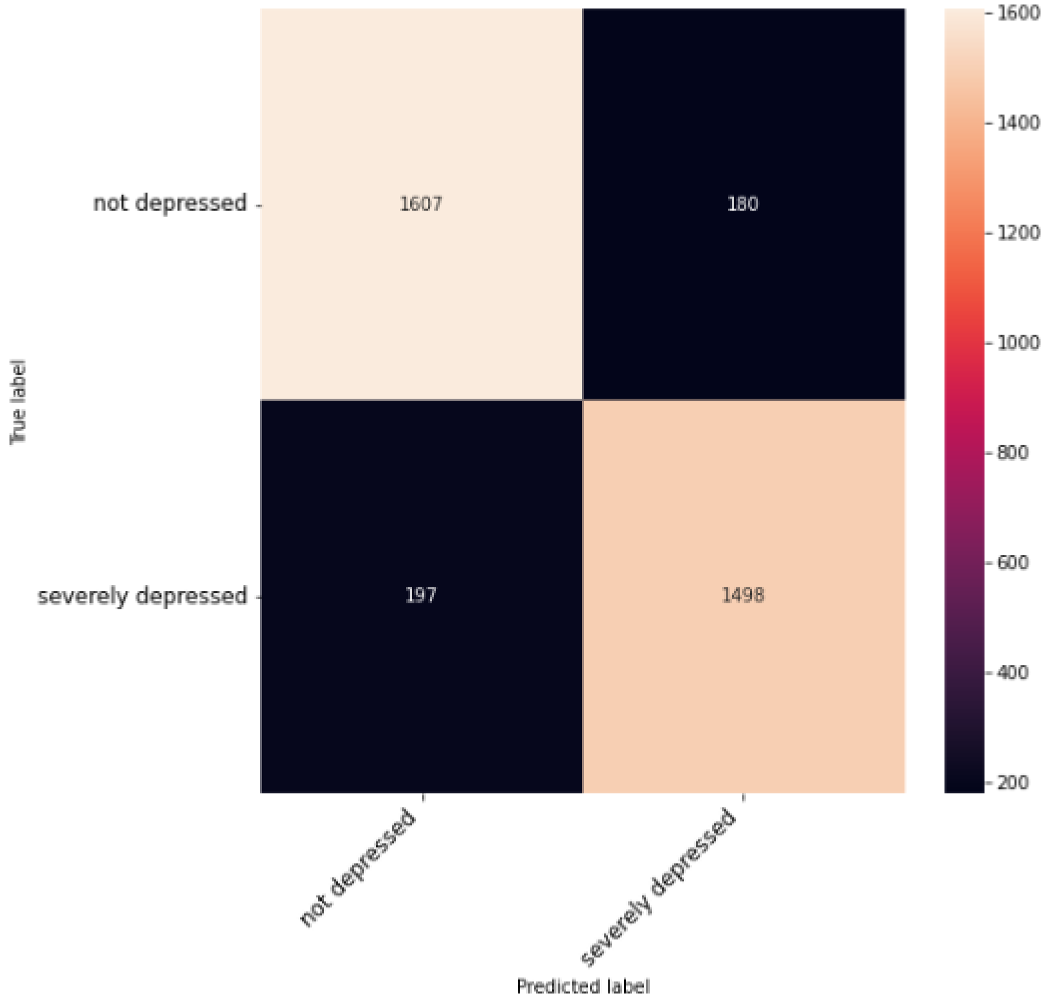
Logistic Regression

```
Accuracy: 0.92 (+/- 0.01)
Accuracy test: 0.88 (+/- 0.01)
Test classification report:
                    precision    recall  f1-score   support

     not depressed       0.89      0.90      0.90      1787
severely depressed       0.89      0.88      0.89      1695

          accuracy                           0.89      3482
         macro avg       0.89      0.89      0.89      3482
      weighted avg       0.89      0.89      0.89      3482
```

```
Test confusion matrix:
```



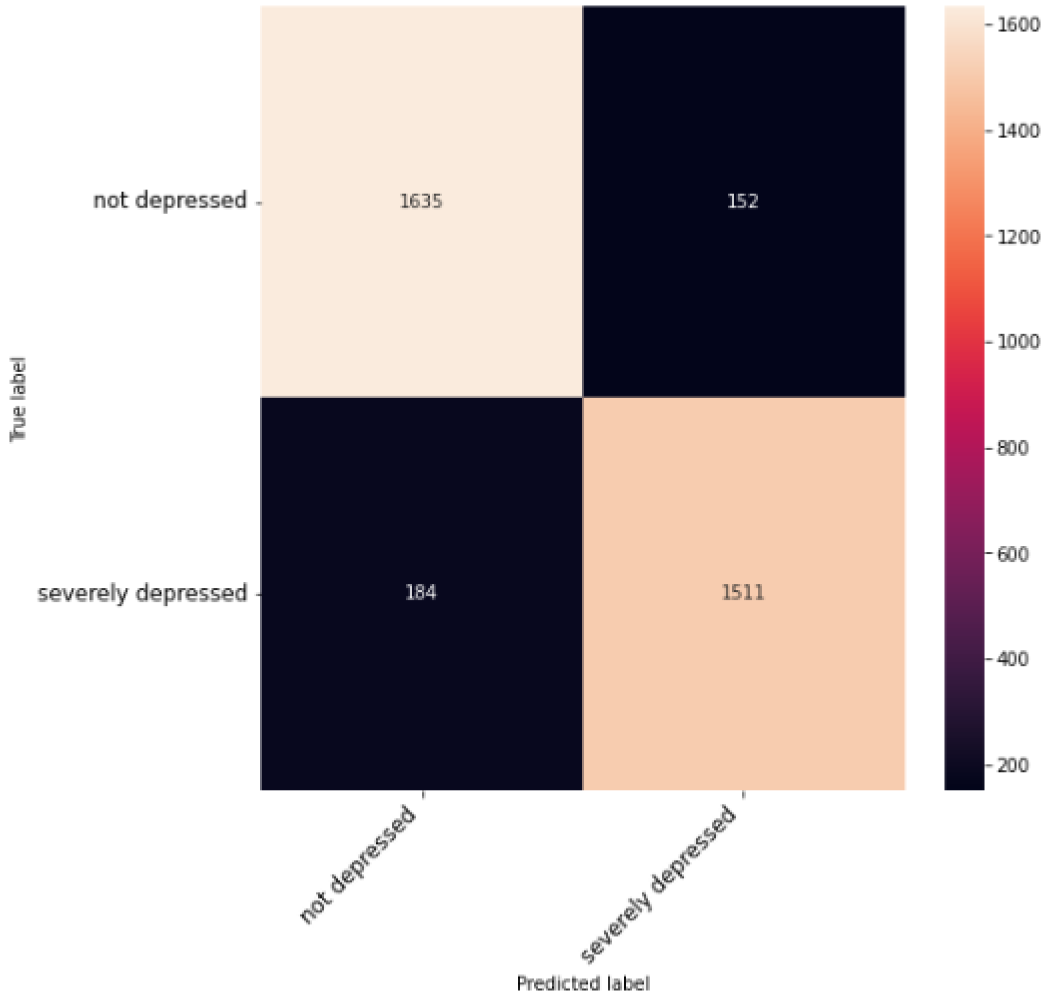
SGDClassifier

```
Accuracy: 0.91 (+/- 0.01)
Accuracy test: 0.89 (+/- 0.01)
Test classification report:
                    precision    recall  f1-score   support

     not depressed       0.90      0.91      0.91      1787
severely depressed       0.91      0.89      0.90      1695

          accuracy                           0.90      3482
         macro avg       0.90      0.90      0.90      3482
      weighted avg       0.90      0.90      0.90      3482
```

Test confusion matrix:



Soft Voting:

A Voting Classifier is a machine learning model that trains on an ensemble of numerous models and predicts an output based on their highest probability of chosen class as the output. In soft voting, every individual classifier provides a probability value that a specific data point belongs to a particular target class. The predictions are weighted by the classifier's importance and summed up. Then the target label with the greatest sum of weighted probabilities wins the vote.

```
Accuracy: 0.92 (+/- 0.01)
Accuracy test: 0.89 (+/- 0.00)
Test classification report:
                    precision    recall  f1-score   support

    not depressed       0.91      0.90      0.91      1787
severely depressed       0.90      0.91      0.90      1695

         accuracy                           0.90      3482
        macro avg       0.90      0.90      0.90      3482
     weighted avg       0.90      0.90      0.90      3482
```

Creating a pipeline with all the functions:

The purpose of the pipeline is to assemble several steps that can be cross validated together while setting different parameters. Pipelines function by allowing a linear series of data transforms to be linked together, resulting in a measurable modeling process.

```
#Pipeline

text_classification_pipeline = Pipeline([
    ('text_preprocessor', TextPreprocessor(text_attribute='text')),
    ('vectorizer', TfidfVectorizer(analyzer = "word", max_features=10000)),
    ('todense_converter', DenseTransformer()),
    ('scaler', MinMaxScaler()),
    ('classifier', VotingClassifier(estimators=[
                         ('lr', LogisticRegression(multi_class='ovr', solver = 'liblinear', C=10, penalty = 'l2')),
                        ('mb', MultinomialNB()),
                        ('sgd', SGDClassifier(alpha=.0001, max_iter=50, loss='log', penalty="elasticnet"))
                        ],
                                   voting='soft', n_jobs=-1))
     ])
```

Prediction for user generated test cases:

	Unnamed: 0	text
11604	17436	I haven't done or been in remote classes and I...
11605	17437	Let's mess up the billionaires And since we le...
11606	17439	Why do people think every mistake can be fixed...
11607	17442	Sweden, a tool to further your political agend...
11608	17444	My Mother is suicidal and has a plan, not sure...

```
1 y_test_pred
```

```
array(['severely depressed', 'not depressed', 'severely depressed', ...,
       'severely depressed', 'not depressed', 'not depressed'],
      dtype=object)
```

RESULTS

From the above confusion matrices and classification report we can conclude that the machine learning models have a test accuracy of:

Multinomial naïve bayes:	0.85
Logistic regression:	0.88
SGD Classifier:	0.89
Soft Voting:	0.89

We can observe that SGD Classifier reports the highest test accuracy. Further, the model has been tested on user generated input and has produced accurate results for these test cases.

CONCLUSION

This project component defines a binary classification problem as identifying whether a person is depressed or not based on his twitter postings and social media commens. Different machine learning algorithms were exploited and different feature datasets are explored. Many preprocessing steps were performed, including data preparation and data labeling, and feature extraction and selection.

This study can be considered as a step toward building a complete social media based platform for analyzing and predicting mental and psychological issues and recommending solutions for these users. It also makes use of a rich, diversified feature set that includes both tweet content and user behavioural patterns. This research can be expanded in the future by evaluating other machine learning models that are less prone to overfit the data and finding a more reliable technique to measure the impact of different features.

REFERENCES

- https://monkeylearn.com/blog/text-cleaning/#:~:text=Text%20cleaning%20can%20be%20performed,words%20to%20their%20root%20form.&text=You'd%20need%20to%20perform,Removing%20Stopwords
- https://machinelearningmastery.com/voting-ensembles-with-python/
- https://scikit-learn.org/stable/modules/sgd.html
- https://www.analyticsvidhya.com/blog/2017/01/ultimate-guide-to-understand-implement-natural-language-processing-codes-in-python/
- https://neptune.ai/blog/exploratory-data-analysis-natural-language-processing-tools
- **Literture Survey:** Predicting Anxiety, Depression and Stress in Modern Life using Machine Learning Algorithms By AnuPriyaa|ShrutiGarga|Neha PrernaTiggaa https://www.sciencedirect.com/science/article/pii/S1877050920309091

APPENDIX

(Libraries and important functions)

```
import re
import string
from sklearn.base import TransformerMixin, BaseEstimator
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.calibration import CalibratedClassifierCV
from imblearn.under_sampling import InstanceHardnessThreshold
```

```
from sklearn.svm import LinearSVC
from sklearn.preprocessing import MinMaxScaler
from sklearn.feature_selection import SelectFromModel
from sklearn.linear_model import LogisticRegression, SGDClassifier
from sklearn.ensemble import VotingClassifier
from sklearn.naive_bayes import MultinomialNB, ComplementNB
from sklearn.svm import LinearSVC
from sklearn.model_selection import cross_val_score
from sklearn.metrics import classification_report
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
from sklearn.utils.multiclass import unique_labels
from sklearn.feature_selection import SelectFromModel
from imblearn.pipeline import Pipeline
import pickle
import spacy
nlp = spacy.load("en_core_web_lg")
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns


class TextPreprocessor(TransformerMixin):
    def __init__(self, text_attribute):
        self.text_attribute = text_attribute

    def transform(self, X, *_):
        X_copy = X.copy()
        X_copy[self.text_attribute]=X_copy[self.text_attribute].apply(s
elf._preprocess_text)
        return X_copy

    def _preprocess_text(self, text):
        return self._lemmatize(self._leave_letters_only(self._clean(tex
t)))

    def _clean(self, text):
        bad_symbols = '!"#%&\'*+,-<=>?[\\]^_`{|}~'
        text_without_symbols = text.translate(str.maketrans('', '', bad
_symbols))

        text_without_bad_words = ''
        for line in text_without_symbols.split('\n'):
            if not line.lower().startswith('from:') and not line.lower(
).endswith('writes:'):
                text_without_bad_words += line + '\n'
```

```
        clean_text = text_without_bad_words
        email_regex=r'([a-zA-Z0-9_.+-]+@[a-zA-Z0-9-]+\.[a-zA-Z0-9-.]+)'
        regexes_to_remove=[email_regex, r'Subject:', r'Re:']
        for r in regexes_to_remove:
            clean_text = re.sub(r,'', clean_text)

        return clean_text

    def _leave_letters_only(self, text):
        text_without_punctuation = text.translate(str.maketrans('', '',
 string.punctuation))
        return ' '.join(re.findall("[a-zA-
Z]+", text_without_punctuation))

    def _lemmatize(self, text):
        doc = nlp(text)
        words = [x.lemma_ for x in [y for y in doc if not y.is_stop and
 y.pos_ != 'PUNCT'
                                    and y.pos_ != 'PART' and y.pos_ != 'X']]
        return ' '.join(words)

    def fit(self, *_):
        return self


text_preprocessor = TextPreprocessor(text_attribute='text')
df_preprocessed = text_preprocessor.transform(df)
train, test = train_test_split(df_preprocessed, test_size=0.3)
tfidf_vectorizer=TfidfVectorizer(analyzer="word",max_features=10000)
X_tfidf_train=tfidf_vectorizer.fit_transform(train['text'])
X_tfidf_test=tfidf_vectorizer.transform(test['text'])
y = train['target']
y_test = test['target']
X, y = X_tfidf_train, y
X_test, y_test = X_tfidf_test, y_test
scaler = MinMaxScaler()
X_norm = scaler.fit_transform(X.toarray())
X_test_norm = scaler.transform(X_test.toarray())
lsvc = LinearSVC(C=100, penalty='l1', max_iter=500, dual=False)
lsvc.fit(X_norm, y)
fs = SelectFromModel(lsvc, prefit=True)
X_sel = fs.transform(X_norm)
X_test_sel = fs.transform(X_test_norm)
from IPython.display import Markdown, display
```

```
def show_top10_features(classifier, feature_names, categories):
    for i, category in enumerate(categories):
        top10 = np.argsort(classifier.coef_[0, i])[-100:]
        display(Markdown("**%s**: %s" % (category, ", ".join(feature_na
mes[top10]))))

from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
from sklearn.utils.multiclass import unique_labels
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

def print_confusion_matrix(confusion_matrix,
                           class_names,
                           figsize = (8,8),
                           fontsize=12,
                           ylabel='True label',
                           xlabel='Predicted label'):
    df_cm = pd.DataFrame(
        confusion_matrix, index=class_names, columns=class_names,
    )
    fig = plt.figure(figsize=figsize)
    try:
        heatmap = sns.heatmap(df_cm, annot=True, fmt="d")
    except ValueError:
        raise ValueError("Confusion matrix values must be integers.")
    heatmap.yaxis.set_ticklabels(heatmap.yaxis.get_ticklabels(), rotati
on=0, ha='right',fontsize=fontsize)
    heatmap.xaxis.set_ticklabels(heatmap.xaxis.get_ticklabels(), rotati
on=45, ha='right',fontsize=fontsize)

def evaluate_model(model, X, y, X_test, y_test, target_names=None):
    scores = cross_val_score(model, X, y, cv=5, scoring='accuracy')
    scores_test = cross_val_score(model, X_test, y_test, cv=5, scoring=
'accuracy')

    print("Accuracy: %0.2f (+/-
 %0.2f)" % (scores.mean(), scores.std()))
    print("Accuracy test: %0.2f (+/-
 %0.2f)" % (scores_test.mean(), scores_test.std()))

    print("Test classification report: ")
    if target_names is None:
        target_names = model.classes_
```

```
    print(classification_report(y_test, model.predict(X_test), target_n
ames=target_names))
    print("Test confusion matrix: ")
    print_confusion_matrix(confusion_matrix(y_test, model.predict(X_tes
t)), class_names=target_names)

mb = MultinomialNB()
mb.fit(X_sel, y)
evaluate_model(mb, X_sel, y, X_test_sel, y_test)
pickle.dump(mb, open("MultinomialNB_Text_classification",'wb')) #90%

lr = LogisticRegression(multi_class='ovr', solver = 'liblinear', C=10,
penalty = 'l2')
lr.fit(X_sel, y)
evaluate_model(lr, X_sel, y, X_test_sel, y_test)
pickle.dump(lr, open("LogisticRegression_Text_classification", 'wb'))
sgd = SGDClassifier(alpha=.0001, max_iter=50, loss='log',
                                    penalty="elasticnet", n_jobs=-1)
sgd.fit(X_sel, y)
evaluate_model(sgd, X_sel, y, X_test_sel, y_test)
pickle.dump(sgd, open("SGDClassifier_Text_classification", 'wb'))

vclf_sgd = VotingClassifier(estimators=[
         ('lr', LogisticRegression(multi_class='ovr', solver = 'libline
ar', C=10, penalty = 'l2')),('mb', MultinomialNB()),
        ('sgd', SGDClassifier(alpha=.0001, max_iter=50, loss='log', pen
alty="elasticnet"))
], voting='soft', n_jobs=-1)
vclf_sgd.fit(X_sel, y)
evaluate_model(vclf_sgd, X_sel, y, X_test_sel, y_test)

pickle.dump(vclf_sgd, open("VotingClassifier_Text_classification", 'wb'
)) #93%

#Pipeline
text_classification_pipeline = Pipeline([
    ('text_preprocessor', TextPreprocessor(text_attribute='text')),
    ('vectorizer', TfidfVectorizer(analyzer = "word", max_features=1000
0)),
    ('todense_converter', DenseTransformer()),
    ('scaler', MinMaxScaler()),
    ('classifier', VotingClassifier(estimators=[('lr', LogisticRegressi
on(multi_class='ovr', solver = 'liblinear', C=10, penalty = 'l2')),('mb
', MultinomialNB()),('sgd', SGDClassifier(alpha=.0001, max_iter=50,loss
='log',penalty="elasticnet"))],voting='soft', n_jobs=-1)) ])
```