

EEP 596A Advanced Topics in Signal and Image Processing

Mini Project 2 Report

Introduction and Overview

This assignment is about Twitter Sentiment Analysis. In this project, we are asked to classify the sentiment of tweets from a real dataset by building and training some practical machine learning models. Sentiment analysis is meant to classify the emotion and perspective expressed from original text sources. It is usually that a few sentiment data details appeared by analyzing tweets. This data is contributing to learning about the opinions of people. Therefore, we are about to develop an automatic machine learning model to judge the sentiments of tweets. However, it is difficult to be developed because of some noise, which could be called 'useless characters' like punctuations and URLs. So, in this article, we are about to develop a pipeline to analyze the Twitter dataset. It includes three classifiers: Logistic Regression, Bernoulli Naive Bayes, and SVM along with using Term Frequency- Inverse Document Frequency (TF-IDF). The criterion for judging models is accuracy and F1-score.

Preliminary Work

In this section, we imported various necessary dependent libraries, read and loaded the dataset, visualization, and built a word cloud to show the most common words in both positive and negative datasets. Finally, we preprocessed the dataset, including converting it to lowercase, cleaning the datasets from stopwords, punctuations, URLs, numbers, and common words. We did all the things above to decrease the useless attributes of datasets and make them cleaner than before. We supposed that in this way, the models would be trained in a better condition and finally arrived at better accuracies and F1-scores.

Importing Necessary Dependencies

```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
from nltk.tokenize import word_tokenize
import nltk
from nltk import pos_tag
# nltk.download('punkt')
# nltk.download('stopwords')
# nltk.download('wordnet')
# nltk.download('averaged_perceptron_tagger')
from wordcloud import WordCloud
from nltk.corpus import stopwords
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.naive_bayes import BernoulliNB
from sklearn.metrics import confusion_matrix, classification_report
import seaborn as sns
from sklearn.svm import LinearSVC
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import LabelEncoder
from flair.models import TextClassifier
from flair.data import Sentence
from sklearn.metrics import accuracy_score
from flair.data import Corpus
from flair.datasets import TREC_6
from flair.embeddings import WordEmbeddings, FlairEmbeddings, DocumentRNNEmbeddings
from flair.trainers import ModelTrainer
from sklearn.metrics import mean_squared_error
from torch.optim.adam import Adam
import re, spacy
from flair.embeddings import TransformerDocumentEmbeddings
from flair.data import Corpus
from flair.datasets import CSVClassificationCorpus
```

Picture 2- 1

In this assignment, we applied several libraries, including Pandas, NumPy, Sklearn, Torch, NLTK and so on to support our whole work.

Reading and Loading the Dataset

```
df = pd.read_csv('twitter_sentiments_data.csv', low_memory=False)
df = df[['target', 'tweet']]
df.sample(5)
```

	target	tweet
453684	0	Omg!!! Gran torino is so sad. Just saw the sa...
552333	0	@leezukor it's good to know they're ok, but h...
732190	1	watch this if you need a smile http://www.im...
1167333	1	Quote: "No Matter What".....Always...
30728	0	@Senorita_Emz Global warming you need some ch...

Picture 2- 2

We read the dataset file by the function `read_csv` in Pandas and just used two columns of it: `target` and `tweet`, to remove other useless information.

```

data_pos = df[df['target'] == 1]
data_neg = df[df['target'] == 0]
data_pos_small = data_pos.iloc[:100000]
data_neg_small = data_neg.iloc[:100000]

dataset = pd.concat([data_neg, data_pos])
dataset = pd.concat([data_neg_small, data_pos_small])
data_targets = dataset.drop('tweet',axis = 1)
dataset = dataset.drop('target',axis = 1)

```

Picture 2- 3

Then divided the dataset into two sections: a positive one and a negative one. Chose ten thousand sentences for each set since the original dataset is too large to be completed quickly.

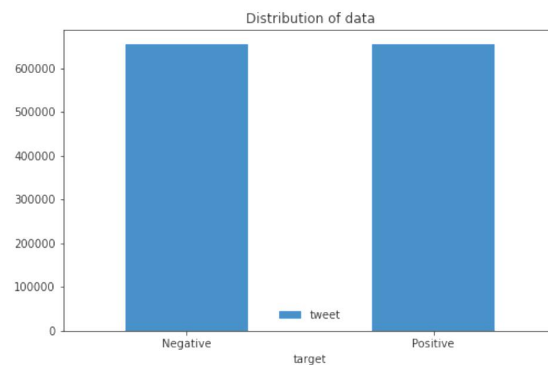
Data Visualization of Target Variables

```

ax = df.groupby('target').count()
ax = ax.plot(kind='bar', figsize = (8,5),title='Distribution of data')
ax.set_xticklabels(['Negative','Positive'], rotation=0)

text, target = list(df['tweet']), list(df['target'])

```

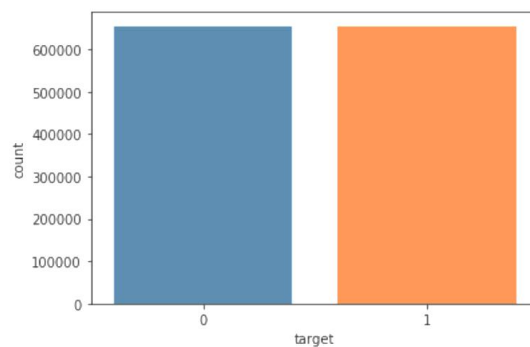


Picture 2- 4

```

sns.countplot(x='target', data=df)
<AxesSubplot:xlabel='target', ylabel='count'>

```

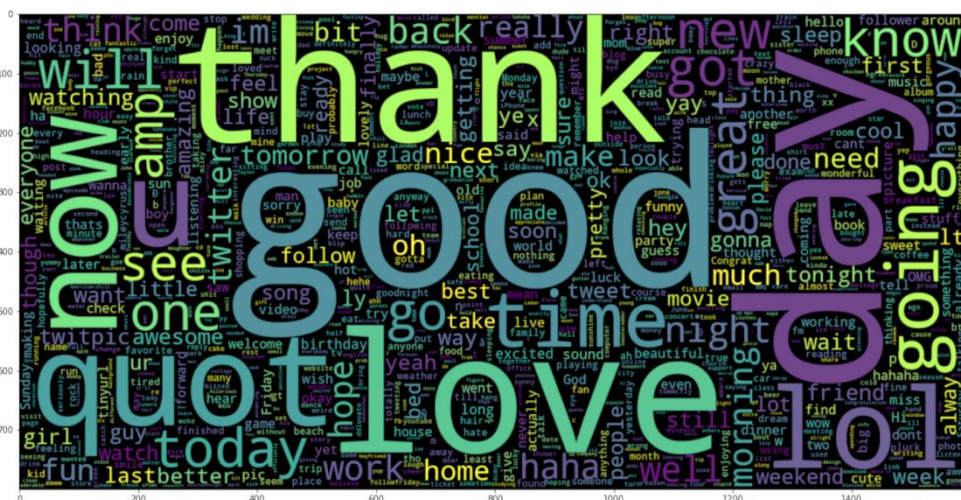


Picture 2- 5

Word cloud

```
data_pos_cloud = df['tweet'][len(data_neg):]
plt.figure(figsize = (20,20))
wc = WordCloud(max_words = 1000 , width = 1600 , height = 800,
               collocations=False).generate(" ".join(data_pos_cloud))
plt.imshow(wc)
```

<matplotlib.image.AxesImage at 0x1c6087596a0>



Picture 2- 7

A word cloud can demonstrate the common words in the dataset, so we build two clouds, one is for negative and another is for positive. In these two pictures, we realized that people usually say ‘day’, ‘work’, ‘now’, ‘go’, ‘time’, and ‘today’, etc. when they feel not good. By contrast, ‘good’, ‘thank’, ‘love’, ‘day’, ‘lol’, ‘now’ are some predictors of positive sentiment. However, we could learn that some words like ‘day’, ‘today’, ‘going’ occurred on both sides, so cleaning the dataset would be what we should do for the next step.

Data Processing

Converting to Lowercase

```
dataset['tweet']=dataset['tweet'].str.lower()
dataset['tweet'].head()

0    is upset that he can't update his facebook by ...
1    @kenichan i dived many times for the ball. man...
2    @nationwideclass no, it's not behaving at all....
3                                     @kwesidei not the whole crew
4                                     need a hug
Name: tweet, dtype: object
```

Picture 3- 1

We converted all the letters to lowercase by lower() function.

Defining and Cleaning Stopwords

```
stopwords_new =['i', 'me', 'my', 'myself', 'we', 'our', 'ours', \
'ourselves', 'you', "you're", "you've", "you'll", \
"you'd", 'your', 'yours', 'yourself', 'yourselves', \
'he', 'him', 'his', 'himself', 'she', "she's", 'her', \
'hers', 'herself', 'it', "it's", 'its', 'itself', 'they', \
'them', 'their', 'theirs', 'themselves', 'what', 'which', \
'who', 'whom', 'this', 'that', "that'll", 'these', 'those', \
'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', \
'have', 'has', 'had', 'having', 'do', 'does', 'did', \
'doing', 'a', 'an', 'the', 'and', 'if', 'on', 'because', \
'as', 'of', 'at', 'by', 'for', 'with', 'about', 'between', 'into', \
'through', 'during', 'before', 'after', 'above', 'below', \
'to', 'from', 'in', 'out', 'on', 'off', \
'over', 'under', 'again', 'further', 'then', 'once', \
'here', 'there', 'when', 'where', 'all', 'any', 'both', \
'each', 'few', 'more', 'most', 'other', 'some', 'such', 'own', \
'same', 'so', 'than', 'too', 'very', 's', \
'will', 'just', 'now', 'd', 'll', 'm', 'o', 're', 've', 'y']

def cleaning_stopwords(text):
    return " ".join([word for word in str(text).split() if word not in stopwords_new])
dataset['tweet'] = dataset['tweet'].apply(lambda text: cleaning_stopwords(text))
dataset['tweet'].head()

0    upset can't update facebook texting it... migh...
1    @kenichan dived many times ball. managed save ...
2    @nationwideclass no, not behaving all. i'm mad...
3                                     @kwesidei not whole crew
4                                     need hug
Name: tweet, dtype: object
```

Picture 3- 2

We defined the stopwords library and removed the words in that library from data to eliminate the influence made by stopwords.

Removing Names

```
def cleaning_names(text):
    return " ".join([word for word in str(text).split() if '@' not in word])
dataset['tweet'] = dataset['tweet'].apply(lambda text: cleaning_names(text))
dataset['tweet'].head()

0    upset can't update facebook texting it... migh...
1    dived many times ball. managed save 50% rest g...
2    no, not behaving all. i'm mad. why here? can't...
3                                     not whole crew
4                                     need hug
Name: tweet, dtype: object
```

Picture 3- 3

We removed names that were cued by the label '@'. We supposed that nicknames were not attributed to classification.

Removing URLs

```
def cleaning_URLs(data):
    return re.sub('((www.[^s]+)|(https?://[^\s]+))', ' ', data)
dataset['tweet'] = dataset['tweet'].apply(lambda x: cleaning_URLs(x))
dataset['tweet'].head()

0    upset can't update facebook texting it... migh...
1    dived many times ball. managed save 50% rest g...
2    no, not behaving all. i'm mad. why here? can't...
3                                     not whole crew
4                                     need hug
Name: tweet, dtype: object
```

Picture 3- 4

URLs are non-sentiment parts of sentences, so we removed them.

Removing Numeric Numbers

```
def cleaning_numbers(data):
    return re.sub('[0-9]+', '', data)
dataset['tweet'] = dataset['tweet'].apply(lambda x: cleaning_numbers(x))
dataset['tweet'].head()

0    upset can't update facebook texting it... migh...
1    dived many times ball. managed save % rest go ...
2    no, not behaving all. i'm mad. why here? can't...
3                                     not whole crew
4                                     need hug
Name: tweet, dtype: object
```

Picture 3- 5

We removed numbers from 0 to 9.

Removing Punctuations

```
english_punctuations = string.punctuation
punctuations_list = english_punctuations
def cleaning_punctuations(text):
    translator = str.maketrans('', '', punctuations_list)
    return text.translate(translator)
dataset['tweet'] = dataset['tweet'].apply(lambda x: cleaning_punctuations(x))
dataset['tweet'].head()

0    upset cant update facebook texting it might cr...
1    dived many times ball managed save rest go bo...
2    no not behaving all im mad why here cant see t...
3                                         not whole crew
4                                         need hug
Name: tweet, dtype: object
```

Picture 3- 6

Removing punctuations is beneficial for training and obtaining a higher accuracy.

Tokenizing Sentences

```
def Tokenizer(data):
    sentence = [getattr(row, 'tweet') for row in data.itertuples()]
    sentence_tokenized = [word_tokenize(sentence[i]) for i in range(len(sentence))]
    return sentence_tokenized

sentence_tokenized = Tokenizer(dataset)
sentence_tokenized

[['upset',
 'cant',
 'update',
 'facebook',
 'texting',
 'it',
 'might',
 'cry',
 'result',
 'school',
 'today',
 'also',
 'blah'],
 ['dived', 'many', 'times', 'ball', 'managed', 'save', 'rest', 'go', 'bounds'],
 ['no',
 'not',
 'behaving',
 'all',
 'im',
 'mad',
 'why',
 'here',
 'cant',
 'see',
 'there'],
```

Picture 3- 7

We perform a word tokenizer on all sentences using custom functions and let every single word be a string in the list.

Lemmatizing words

```
lm = nltk.WordNetLemmatizer()
for sen in sentence_tokenized:
    for word,tag in pos_tag(sen):
        if tag in ['NN','NNS','NNP','NNPS']:#noun
            sen[sen.index(word)] = lm.lemmatize(word, pos='n')
        elif tag in ['VB','VBD','VBG','VBN','VBP','VBZ']:#verb
            sen[sen.index(word)] = lm.lemmatize(word, pos='v')
        elif tag in ['JJ','JJR','JJS']:#adjective
            sen[sen.index(word)] = lm.lemmatize(word, pos='a')
        elif tag in ['RB','RBR','RBS']:#adverb
            sen[sen.index(word)] = lm.lemmatize(word, pos='r')
        else: continue
sentence_tokenized

[['upset',
 'cant',
 'update',
 'facebook',
 'texting',
 'it',
 'might',
 'cry',
 'result',
 'school',
 'today',
 'also',
 'blah'],
 ['dive', 'many', 'time', 'ball', 'manage', 'save', 'rest', 'go', 'bound'],
 ['no',
 'not',
 'behave',
 'all',
 'im',
 'mad',
 'why',
 'here',
 'cant',
 'see',
 'there'],
 ['not', 'whole', 'crew'],
 ['need', 'hug'],
 ['nope', 'didnt'],
```

Picture 3- 8

We recognized the tag of each word like ‘NN’ for none, and ‘JJ’ for adjective words by pos_tag () function, and lemmatized the words depending on the result of pos_tag, through WordNetLemmatizer () function in the nltk library.

Removing the most common words

```
from collections import Counter
positive_counts = Counter()
negative_counts = Counter()
total_counts = Counter()
for review, label in zip(dataset_processed.tweet.values,data_targets.target.values):
    words = review.split(' ')
    total_counts.update(words)
    if label == 1:
        positive_counts.update(words)
    else:
        negative_counts.update(words)
```

Picture 3- 9


```

positive_counts.most_common(20)

[('get', 9628),
 ('im', 9058),
 ('go', 8372),
 ('good', 8350),
 ('day', 7460),
 ('love', 6463),
 ('but', 5947),
 ('up', 5017),
 ('like', 4946),
 ('not', 4231),
 ('time', 4175),
 ('thanks', 4156),
 ('well', 4025),
 ('see', 3918),
 ('lol', 3851),
 ('today', 3718),
 ('make', 3510),
 ('think', 3473),
 ('u', 3469),
 ('work', 3448)]

negative_counts.most_common(20)

[('get', 12278),
 ('go', 12123),
 ('im', 12121),
 ('but', 9842),
 ('not', 9335),
 ('work', 7724),
 ('day', 6730),
 ('up', 6475),
 ('no', 6256),
 ('cant', 5454),
 ('dont', 5318),
 ('miss', 5287),
 ('like', 4991),
 ('want', 4834),
 ('today', 4833),
 ('back', 4623),
 ('good', 4167),
 ('feel', 4162),
 ('time', 4019),
 ('think', 3761)]

```

Picture 3- 10

```

def cleaning_commen(text):
    return " ".join([word for word in str(text).split() if word not in ['get', 'go', 'im', 'like', 'up']])
dataset_processed['tweet'] = dataset_processed['tweet'].apply(lambda text: cleaning_commen(text))
dataset_processed['tweet'].head()

0    upset cant update facebook texting it might cr...
1          dive many time ball manage save rest bound
2          no not behave all mad why here cant see there
3                                not whole crew
4                                need hug
Name: tweet, dtype: object

```

Picture 3- 11

In this part, we realized that some high-frequency words, like ‘get’, ‘go’, on both sides will affect the model's judgment. Thus, we checked them and removed them.

Supervising learning models

First of all, split the dataset into two parts, for train and test. Then, we built three models: Bernoulli Naïve Bayes, SVM, and Logistic Regression to train the dataset. The idea behind choosing these models is that we want to try all the classifiers on the dataset ranging from simple ones to complex models and then try to find out the one which gives the best performance among them. After training the model we then applied the evaluation measures to check how the model is performing. Accordingly, we used the following evaluation parameters to check the performance of the models respectively: Accuracy Score, Confusion Matrix with Plot, and ROC-AUC Curve.

Splitting Datasets and Vectorizing

We split the dataset into two parts: X_train and X_test by train_test_split function, which accounts for 80% and 20% separately. And then vectorized all the words to 3000 features by TfidfVectorizer() function.

```
X = dataset_processed.tweet
y = data_targets.target

X_train, X_test, y_train, y_test = train_test_split(X,y,test_size = 0.2, random_state =990628)
vectoriser = TfidfVectorizer(ngram_range=(1,2),max_features = 3000)
vectoriser.fit(X_train)
print('No. of feature_words: ', len(vectoriser.get_feature_names()))
X_train_tf = vectoriser.transform(X_train)
X_test_tf = vectoriser.transform(X_test)

No. of feature_words: 3000

from scipy.sparse import hstack
feat_comb_tr = hstack([X_train_tf,X_train_cv])
feat_comb_te = hstack([X_test_tf,X_test_cv])

X_train_tf.toarray().shape
(160000, 3000)
```

Picture 4- 1

Building evaluation function

The evaluation model function was defined by us. We made the predictions of the test module Besides, in this function, we integrated some other functions like confusion_matrix () to show the confusion matrix and show the classification report by classification_report ().

```
def model_Evaluate(model):
    y_pred = model.predict(X_test_tf)
    print(classification_report(y_test, y_pred))
    cf_matrix = confusion_matrix(y_test, y_pred)
    categories = ['Negative', 'Positive']
    group_names = ['True Neg', 'False Pos', 'False Neg', 'True Pos']
    group_percentages = ['{0:.2%}'.format(value) for value in cf_matrix.flatten() / np.sum(cf_matrix)]
    labels = [f'{v1}\n{v2}' for v1, v2 in zip(group_names, group_percentages)]
    labels = np.asarray(labels).reshape(2,2)
    sns.heatmap(cf_matrix, annot = labels, cmap = 'Blues', fmt = '',
    xticklabels = categories, yticklabels = categories)
    plt.xlabel("Predicted values", fontdict = {'size':14}, labelpad = 10)
    plt.ylabel("Actual values", fontdict = {'size':14}, labelpad = 10)
    plt.title ("Confusion Matrix", fontdict = {'size':18}, pad = 20)
```

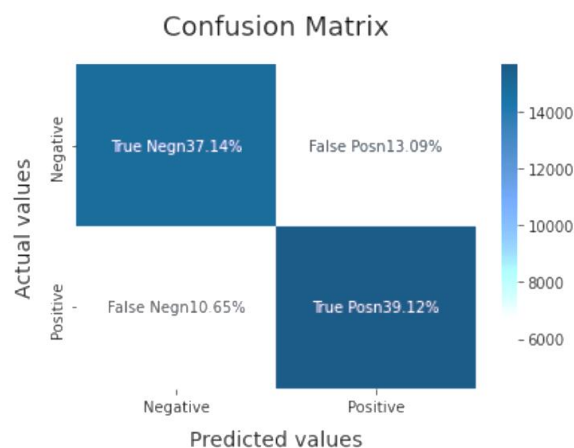
Picture 4- 2

Bernoulli Naïve Bayes

We used the Bernoulli Naïve Bayes model and show the confusion matrix and ROC curve graph below. We could see that the Bayes model did not perform well enough, which just had 0.76 precision and F1-score.

```
BNBmodel = BernoulliNB()
BNBmodel.fit(X_train_tf, y_train)
model_Evaluate(BNBmodel)
y_pred1 = BNBmodel.predict(X_test_tf)
```

	precision	recall	f1-score	support
0	0.78	0.74	0.76	20092
1	0.75	0.79	0.77	19908
accuracy			0.76	40000
macro avg	0.76	0.76	0.76	40000
weighted avg	0.76	0.76	0.76	40000

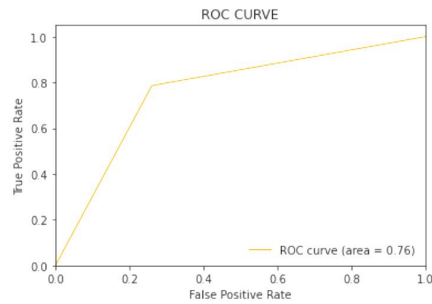


Picture 4- 3

```

from sklearn.metrics import roc_curve, auc
y_test_int = [int(num) for num in y_test]
y_pred1_int = [int(num) for num in y_pred1]
fpr, tpr, thresholds = roc_curve(y_test_int, y_pred1_int)
roc_auc = auc(fpr, tpr)
plt.figure()
plt.plot(fpr, tpr, color='darkorange', lw=1, label='ROC curve (area = %0.2f)' % roc_auc)
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC CURVE')
plt.legend(loc="lower right")
plt.show()

```



Picture 4- 4

Support Vector Machine

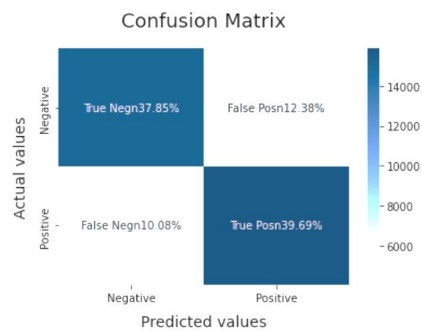
Then, we chose the SVM model. Similarly, we showed the confusion matrix and ROC. It worked better than Bayes that it had 0.78 precision. However, its accuracy was still not high enough.

```

SVCmodel = LinearSVC()
SVCmodel.fit(X_train_tf, y_train)
model_Evaluate(SVCmodel)
y_pred2 = SVCmodel.predict(X_test_tf)

```

	precision	recall	f1-score	support
0	0.79	0.75	0.77	20092
1	0.76	0.80	0.78	19908
accuracy			0.78	40000
macro avg	0.78	0.78	0.78	40000
weighted avg	0.78	0.78	0.78	40000



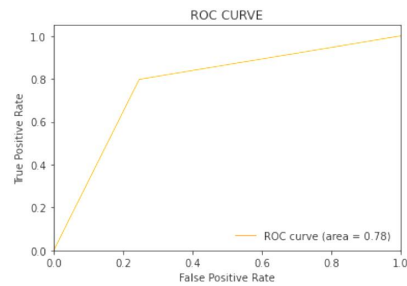
Picture 4- 5

```

from sklearn.metrics import roc_curve, auc

y_pred2_int = [int(num) for num in y_pred2]
fpr, tpr, thresholds = roc_curve(y_test_int, y_pred2_int)
roc_auc = auc(fpr, tpr)
plt.figure()
plt.plot(fpr, tpr, color='darkorange', lw=1, label='ROC curve (area = %0.2f)' % roc_auc)
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC CURVE')
plt.legend(loc="lower right")
plt.show()

```



Picture 4- 6

Logistic Regression

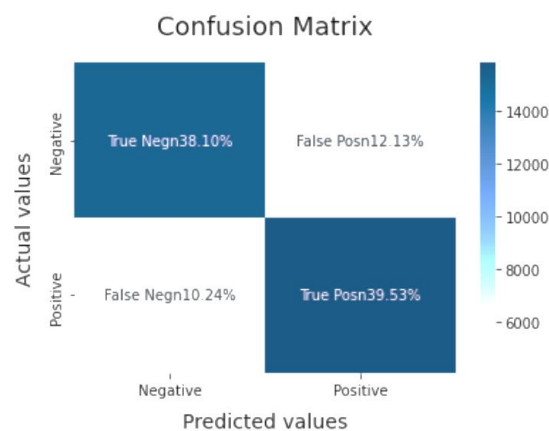
Finally, we tried the logistic regression model and showed the matrix and ROC.

```

LRmodel = LogisticRegression(C = 2, max_iter = 1000, n_jobs=-1)
LRmodel.fit(X_train_tf, y_train)
model_Evaluate(LRmodel)
y_pred3 = LRmodel.predict(X_test_tf)

```

	precision	recall	f1-score	support
0	0.79	0.76	0.77	20092
1	0.77	0.79	0.78	19908
accuracy			0.78	40000
macro avg	0.78	0.78	0.78	40000
weighted avg	0.78	0.78	0.78	40000



Picture 4- 7

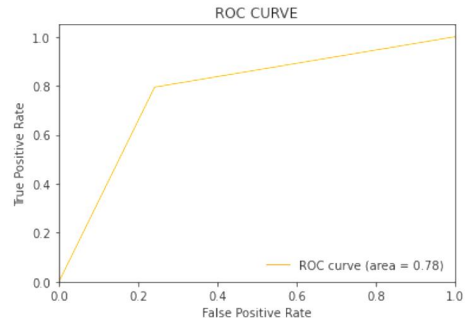
```

from sklearn.metrics import roc_curve, auc

y_pred3_int = [int(num) for num in y_pred3]
fpr, tpr, thresholds = roc_curve(y_test_int, y_pred3_int)
roc_auc = auc(fpr, tpr)

plt.figure()
plt.plot(fpr, tpr, color='darkorange', lw=1, label='ROC curve (area = %0.2f)' % roc_auc)
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC CURVE')
plt.legend(loc="lower right")
plt.show()

```



Picture 4- 8

Comparison of Supervising Models

Chart 1

	Precision	Recall	F1-score	Support
Bernoulli NB	0.76	0.76	0.76	40000
SVC	0.78	0.78	0.78	40000
Logistic	0.78	0.78	0.78	40000

In this chart, we can see that logistic regression and SVC share the value of matrices (0.78), which are higher than BNB—only 0.76 for all. However, 0.78 is not high enough in this assignment, so we decided to build deep learning models.

Neural Network Model

We built two deep learning models: Flair and a two-layer shallow neural network.

Flair

Structure of model

We imported the integrated functions of Flair like TextClassifier and ModelTrainer to train our dataset. Firstly, Loaded the folder where the train, test, and validation datasets are in, built a map to indicate which columns hold the text and labels, loaded the corpus containing training, test, and validation data by CSVClassificationCorpus. Then created the label dictionary by make_label_dictionary, then initialized the transformer document embeddings by using the function of TransformerDocumentEmbeddings. Besides, created the text classifier by TextClassifier, initialized the text classifier trainer with Adam optimizer, which is super practical in the classification assignment. Finally, trained the model with some parameters like a very small learning rate like 0.0001, batch size (128), epochs (5), and optimizer (Adam).

```
data_folder = './data_folder'

column_name_map = {0: "text", 1: "label"}

corpus: Corpus = CSVClassificationCorpus(data_folder,
                                         column_name_map,
                                         skip_header=True,
                                         label_type='sentiment')

label_dict = corpus.make_label_dictionary(label_type='sentiment')

document_embeddings = TransformerDocumentEmbeddings('distilbert-base-uncased', fine_tune=True)

classifier = TextClassifier(document_embeddings, label_dictionary=label_dict, label_type='sentiment')

trainer = ModelTrainer(classifier, corpus)

trainer.train('./model_result',
              learning_rate=0.0001,
              mini_batch_size=128,
              mini_batch_chunk_size=4,
              optimizer=Adam
              )
```

Picture 5- 1

```
Parameters:
- learning_rate: "0.0001"
- mini_batch_size: "128"
- patience: "3"
- anneal_factor: "0.5"
- max_epochs: "5"
- shuffle: "True"
- train_with_dev: "False"
- batch_growth_annealing: "False"
```

Picture 5- 2

Training and Result

```
EPOCH 1 done: loss 0.1399 - lr 0.0001000
DEV : loss 0.10235065221786499 - f1-score (micro avg) 0.8185
```

```
EPOCH 2 done: loss 0.0776 - lr 0.0001000
DEV : loss 0.11744333058595657 - f1-score (micro avg) 0.811
```

```
EPOCH 3 done: loss 0.0336 - lr 0.0001000
DEV : loss 0.1545609086751938 - f1-score (micro avg) 0.81
```

```
EPOCH 4 done: loss 0.0153 - lr 0.0001000
DEV : loss 0.19493556022644043 - f1-score (micro avg) 0.787
```

```
EPOCH 5 done: loss 0.0111 - lr 0.0001000
DEV : loss 0.22807437181472778 - f1-score (micro avg) 0.8085
```

Picture 5- 3

Results:

- F-score (micro) 0.8135
- F-score (macro) 0.8133
- Accuracy 0.8135

By class:

	precision	recall	f1-score	support
0	0.8126	0.8277	0.8201	1027
1	0.8145	0.7986	0.8064	973
micro avg	0.8135	0.8135	0.8135	2000
macro avg	0.8135	0.8131	0.8133	2000
weighted avg	0.8135	0.8135	0.8134	2000
samples avg	0.8135	0.8135	0.8135	2000

Picture 5- 4

```
{'test_score': 0.8135,
 'dev_score_history': [0.8184999999999999, 0.811, 0.81, 0.787, 0.8085],
 'train_loss_history': [0.13986889523174614,
 0.0775751546297688,
 0.033619829821686535,
 0.015287930650208484,
 0.0110993000204694],
 'dev_loss_history': [tensor(0.1024),
 tensor(0.1174),
 tensor(0.1546),
 tensor(0.1949),
 tensor(0.2281)]}
```

Picture 5- 5

We can see that the precision and F1-score are higher than any other models in this report, so we supposed that the Flair model is the most accurate.

Two-layer shallow neural network

Structure of network

We built a two-layer neural network to train the dataset. Firstly, preprocessing dataset by defining the function `pre_process_function()`. This function is used for preprocessing data, counting all words appearing in reviews, generating word2index, counting all the words that appear in the dataset, counting all the values that appear in the target, building word2idx, and assigning a "house number" to each word. Then, initializing parameters of the network by function `init_network()`, like learning rate, number of input nodes, number of hidden nodes, number of output nodes.

```
class SentimentNetwork(object):
    def __init__(self, reviews, labels, hidden_nodes=10, learning_rate = 0.1):

        np.random.seed(1)

        self.pre_process_data(reviews, labels)

        self.init_network(len(self.review_vocab), hidden_nodes, 1, learning_rate)

    def pre_process_data(self, reviews, labels):
        review_vocab = set()
        for review in reviews.values:
            word = review.split(' ')
            review_vocab.update(word)

        self.review_vocab = list(review_vocab)

        label_vocab = set()
        for label in labels.values:
            label_vocab.add(label)
        self.label_vocab = list(label_vocab)

        self.word2index = dict()
        for idx, word in enumerate(self.review_vocab):
            self.word2index[word] = idx

    def init_network(self, input_nodes, hidden_nodes, output_nodes, learning_rate):

        self.learning_rate = learning_rate
        self.input_nodes = input_nodes
        self.hidden_nodes = hidden_nodes
        self.output_nodes = output_nodes

        self.weights_0_1 = np.random.normal( 0.0, self.input_nodes**-0.5, (self.input_nodes, self.hidden_nodes) )
        self.weights_1_2 = np.random.normal( 0.0, self.hidden_nodes**-0.5, (self.hidden_nodes, self.output_nodes) )

        self.layer_1 = np.zeros((1, self.hidden_nodes))

    def sigmoid(self, x):
        return 1 / (1 + np.exp(-x))

    def sigmoid_output_2_derivative(self, output):
        return output * (1 - output)

    def get_target_for_label(self, label):
        if label == 1:
            return 1
        else:
            return 0
```

Picture 5- 6

```

def train(self, training_reviews_raw, training_labels):
    assert(len(training_reviews_raw) == len(training_labels))

    training_reviews = list()
    for review in training_reviews_raw.values:
        words = review.split(' ')
        indicates = set()
        for word in words:
            word = word.lower()
            if word in self.word2index.keys():
                indicates.add(self.word2index[word])
        training_reviews.append(list(indicates))

    assert(len(training_reviews) == len(training_labels))

    correct_so_far = 0

    start = time.time()

    for i in range(len(training_reviews)):
        review = training_reviews[i]
        label = training_labels.iloc[i]

        self.layer_1 *= 0
        for index in review:
            self.layer_1 += self.weights_0_1[index]

        layer_1_o = self.layer_1

        layer_2_i = np.dot( layer_1_o, self.weights_1_2 )
        layer_2_o = self.sigmoid( layer_2_i )

        layer_2_error = layer_2_o - self.get_target_for_label(label)
        layer_2_delta = layer_2_error * self.sigmoid_output_2_derivative(layer_2_o)

        layer_1_error = np.dot( layer_2_delta, self.weights_1_2.T )
        layer_1_delta = layer_1_error

        self.weights_1_2 -= np.dot(layer_1_o.T, layer_2_delta) * self.learning_rate

        for index in review:
            self.weights_0_1[index] -= layer_1_delta[0] * self.learning_rate

        if(layer_2_o >= 0.5 and label== 1): #'positive'
            correct_so_far += 1
        elif(layer_2_o < 0.5 and label== 0): #'negative'
            correct_so_far += 1

        elapsed_time = float(time.time() - start)
        reviews_per_second = i / elapsed_time if elapsed_time > 0 else 0

        sys.stdout.write("\rProgress:" + str(100 * i/float(len(training_reviews)))[4] \
            + "% Speed(reviews/sec):" + str(reviews_per_second)[0:5] \
            + " #Correct:" + str(correct_so_far) + " #Trained:" + str(i+1) \
            + " Training Accuracy:" + str(correct_so_far * 100 / float(i+1))[4] + "%")

        if(i % 2500 == 0):
            print("")

```

Picture 5- 7

```

def test(self, testing_reviews, testing_labels):
    assert(len(testing_reviews) == len(testing_labels))

    correct = 0

    start = time.time()

    for i in range(len(testing_reviews)):
        review = testing_reviews.iloc[i]
        label = testing_labels.iloc[i]

        pred = self.run(review)
        if pred == label:
            correct += 1

        elapsed_time = float(time.time() - start)
        reviews_per_second = i / elapsed_time if elapsed_time > 0 else 0

        sys.stdout.write("\rProgress:" + str(100 * i/float(len(testing_reviews)))[4] \
            + "% Speed(reviews/sec):" + str(reviews_per_second)[0:5] \
            + " #Correct:" + str(correct) + " #Tested:" + str(i+1) \
            + " Testing Accuracy:" + str(correct * 100 / float(i+1))[4] + "%")

```

Picture 5- 8

```

def predict(self, predict_review):
    start = time.time()
    predict = []
    for i in range(len(predict_review)):
        review = predict_review.iloc[i]
        predict.append(self.run(review))
        elapsed_time = float(time.time() - start)
        reviews_per_second = i / elapsed_time if elapsed_time > 0 else 0

        sys.stdout.write("\rProgress:" + str(100 * i/float(len(predict_review)))[0:4] \
                        + "% Speed(reviews/sec):" + str(reviews_per_second)[0:5])

    return predict

def run(self, review):
    indicates = set()
    for word in review.split(' '):
        if word in self.word2index.keys():
            indicates.add( self.word2index[word] )

    self.layer_1 *= 0
    for idx in indicates:
        self.layer_1 += self.weights_0_1[idx]

    layer_1_o = self.layer_1

    layer_2_i = np.dot( layer_1_o, self.weights_1_2 )
    layer_2_o = self.sigmoid( layer_2_i )
    if layer_2_o >= 0.5:
        return 1
    else:
        return 0

```

Picture 5- 9

Training and Result

Then we split the train and test dataset with the proportions of 80% and 20%, trained the model with 20 hidden nodes and a 0.01 learning rate.

```

reviews, X_test, labels, y_test = train_test_split(dataset_processed['tweet'] ,
                                                    data_targets['target'], test_size = 0.2)
mlp = SentimentNetwork(X_train, y_train, hidden_nodes=20, learning_rate=0.01)
mlp.train(X_train, y_train)

```

```

Progress:71.8% Speed(reviews/sec):8762. #Correct:85455 #Trained:115001 Training Accuracy:74.3%
Progress:73.4% Speed(reviews/sec):8758. #Correct:87358 #Trained:117501 Training Accuracy:74.3%
Progress:75.0% Speed(reviews/sec):8756. #Correct:89288 #Trained:120001 Training Accuracy:74.4%
Progress:76.5% Speed(reviews/sec):8751. #Correct:91178 #Trained:122501 Training Accuracy:74.4%
Progress:78.1% Speed(reviews/sec):8749. #Correct:93055 #Trained:125001 Training Accuracy:74.4%
Progress:79.6% Speed(reviews/sec):8743. #Correct:94940 #Trained:127501 Training Accuracy:74.4%
Progress:81.2% Speed(reviews/sec):8743. #Correct:96873 #Trained:130001 Training Accuracy:74.5%
Progress:82.8% Speed(reviews/sec):8752. #Correct:98802 #Trained:132501 Training Accuracy:74.5%
Progress:84.3% Speed(reviews/sec):8755. #Correct:100763 #Trained:135001 Training Accuracy:74.6%
Progress:85.9% Speed(reviews/sec):8731. #Correct:102643 #Trained:137501 Training Accuracy:74.6%
Progress:87.5% Speed(reviews/sec):8723. #Correct:104528 #Trained:140001 Training Accuracy:74.6%
Progress:89.0% Speed(reviews/sec):8705. #Correct:106433 #Trained:142501 Training Accuracy:74.6%
Progress:90.6% Speed(reviews/sec):8676. #Correct:108312 #Trained:145001 Training Accuracy:74.6%
Progress:92.1% Speed(reviews/sec):8679. #Correct:110210 #Trained:147501 Training Accuracy:74.7%
Progress:93.7% Speed(reviews/sec):8685. #Correct:112153 #Trained:150001 Training Accuracy:74.7%
Progress:95.3% Speed(reviews/sec):8675. #Correct:114061 #Trained:152501 Training Accuracy:74.7%
Progress:96.8% Speed(reviews/sec):8675. #Correct:115946 #Trained:155001 Training Accuracy:74.8%
Progress:98.4% Speed(reviews/sec):8674. #Correct:117855 #Trained:157501 Training Accuracy:74.8%
Progress:99.9% Speed(reviews/sec):8674. #Correct:119759 #Trained:160000 Training Accuracy:74.8%

```

Picture 5- 10

```
mlp.test(X_test,y_test)
Progress:4.83% Speed(reviews/sec):10343 #Correct:1493 #Tested:1934 Testing Accuracy:77.1%
Progress:10.2% Speed(reviews/sec):10946 #Correct:3161 #Tested:4102 Testing Accuracy:77.0%
Progress:15.9% Speed(reviews/sec):11370 #Correct:4918 #Tested:6395 Testing Accuracy:76.9%
Progress:21.1% Speed(reviews/sec):11284 #Correct:6484 #Tested:8456 Testing Accuracy:76.6%
Progress:26.3% Speed(reviews/sec):11246 #Correct:8098 #Tested:10540 Testing Accuracy:76.8%
Progress:31.5% Speed(reviews/sec):11231 #Correct:9740 #Tested:12617 Testing Accuracy:77.1%
Progress:36.8% Speed(reviews/sec):11223 #Correct:11339 #Tested:14726 Testing Accuracy:76.9%
Progress:42.0% Speed(reviews/sec):11224 #Correct:12950 #Tested:16824 Testing Accuracy:76.9%
Progress:47.4% Speed(reviews/sec):11240 #Correct:14563 #Tested:18973 Testing Accuracy:76.7%
Progress:52.4% Speed(reviews/sec):11193 #Correct:16076 #Tested:20978 Testing Accuracy:76.6%
Progress:57.8% Speed(reviews/sec):11219 #Correct:17748 #Tested:23143 Testing Accuracy:76.6%
Progress:63.3% Speed(reviews/sec):11271 #Correct:19445 #Tested:25350 Testing Accuracy:76.7%
Progress:68.5% Speed(reviews/sec):11244 #Correct:20989 #Tested:27409 Testing Accuracy:76.5%
Progress:73.9% Speed(reviews/sec):11272 #Correct:22650 #Tested:29586 Testing Accuracy:76.5%
Progress:78.6% Speed(reviews/sec):11199 #Correct:24093 #Tested:31480 Testing Accuracy:76.5%
```

Picture 5- 11

Finally, we got 74.8% accuracy of training and 76.5% of testing.


















Comparison of Models and Kaggle Result

Chart 2

	Precision	Recall	F1-score	Support
Bernoulli NB	0.76	0.76	0.76	40000
SVC	0.78	0.78	0.78	40000
Logistic	0.78	0.78	0.78	40000
Flair	0.8135	0.8135	0.8135	40000
Two-layer	0.748	0.748	0.748	40000

The Flair model has the highest accuracy and F1score—both are 0.8135, followed by logistic regression and support vector machines, then the Bernoulli model, and finally the self-built two-layer neural network.

As for Kaggle, we got 7th place on the leaderboard for F1-score 0.83998.

#	Team	Members	Score	Entries	Last	Code
1	allysa & prathibha	 	0.87409	6	1d	
2	Zihe Song		0.87312	3	5h	
3	Scammers!!!	 	0.86965	15	20h	
4	Sha Alex		0.85959	1	3h	
5	Great Team	 	0.85954	6	2d	
6	Yungdi	 	0.84996	1	4d	
7	Yesmola	 	0.83998	24	1d	
 Your Best Entry! Your most recent submission scored 0.83998, which is the same as your previous score. Keep trying!						
8	David Alberts		0.83727	2	1d	
9	Blake & Ryan	 	0.83294	15	2d	
10	Daniel H		0.82340	10	1d	

Picture 5- 12

Conclusion and contribution of each team member

We discussed how to process the dataset, specifically how to decrease the dimension of the attributes. We then created several functions to pre-process the data. Afterward, we built three supervising learning models including BNB, SVM, and LR models, and compared them to figure out which one is the best. Furthermore, we built two deep learning models like Flair and a two-layer shallow neural network and compared them. Finally, we compared all of the models and find that Flair had the highest accuracy.

In the mini-project, our team has two people: Xiangyu Gao and Rui Li. Xiangyu is mainly responsible for the model establishment, writing main codes, and parameter debugging, while Rui is mainly responsible for finding reference materials, correcting codes, and selecting model methods. Throughout the project, we found each other online and decided to team up together. During this time, we were coding face-to-face three times a week, because face-to-face communication allowed us to program more efficiently. Although we have our division of labor, the overall procedure is the result of both of us. We all feel very happy to work with each other.