Meghana Kambhampati

MXK190048

Text Classification

This notebook uses a small data set and uses it to implement text classifications. First, a graph of the target class distributions will be made. Then naive Bayes, Logistic regression, and neural networks will be performed (using sklearn) to classify the text. Each approach to text classification will be analyzed.

```
# imports for graph
import pandas as pd
import seaborn as sb
from sklearn import datasets
```

```
# imports for Naive Baynes
import pandas as pd
from nltk.corpus import stopwords
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import MultinomialNB
import math
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, confusion_matrix
from sklearn.metrics import classification_report
```

```
# imports for logistic regression
import pandas as pd
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, log_loss
```

```
# imports for neural network
from nltk.corpus import stopwords
from sklearn.pipeline import Pipeline
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, log_loss
```

Graph of Class Distributions

The class distribution graph will be made with Seaborn, a Python package used for data display. The graph will show how each piece of data is categorized in the data set. Each class represents an emotion: "happiness," "sadness," "anger," "love," "surprise," and "fear."

```
from google.colab import files
import io
uploaded = files.upload()
```

Choose Files Emotion_final.csv

- **Emotion_final.csv**(text/csv) - 2259660 bytes, last modified: 3/29/2023 - 100% done

```
Saving Emotion_final.csv to Emotion_final (2).csv
```

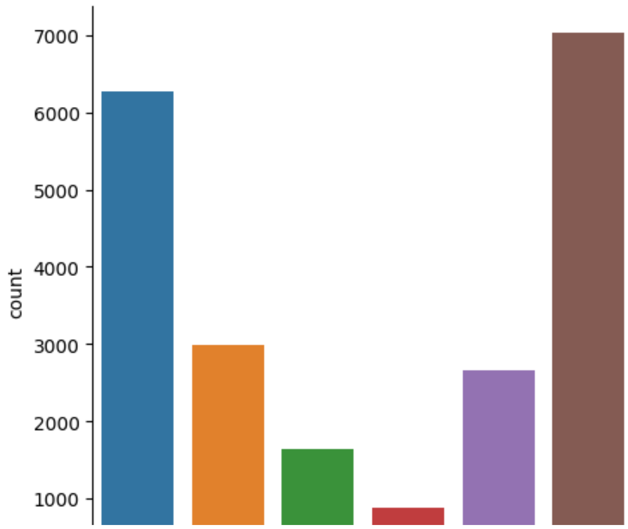
```
# load data set
df = pd.read_csv(io.StringIO(uploaded['Emotion_final.csv'].decode('utf-8')), header=0, encoding='latin-1')

X = df.Text
y = df.Emotion

df = pd.DataFrame(X, columns=df.Text)
df_y = pd.DataFrame(y, columns=['Emotion'])

# create categorical graph
sb.catplot(x="Emotion", kind='count', data=df_y)
```

```
<seaborn.axisgrid.FacetGrid at 0x7f0e8d213f10>
```



The graph shows that there is an unequal distribution of emotions based on the given pieces of text. "Happy" has the most instances, while "surprise" has the least.

Emotion

This model should be able to predict what emotion a (small) piece of text is expressing.

Naive Bayes

Naive Bayes is a linear classifier that is used for classification. It is used to predict the probability of a target value. Data is first split into training and testing sets. The training set is used by Naive Bayes to "learn" about the classifications.

```
import nltk
nltk.download('stopwords')
stopwords = set(stopwords.words('english'))
vectorizer = TfidfVectorizer(stop_words=list(stopwords))
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

```
# split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, train_size=0.8, random_state=1234)
X_train = vectorizer.fit_transform(X_train)
X_test = vectorizer.transform(X_test)


print('train size:', X_train.shape)
print(X_train.toarray()[:5])

print('\ntest size:', X_test.shape)
print(X_test.toarray()[:5])
```

```
train size: (17167, 16830)
[[0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]]

test size: (4292, 16830)
[[0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]]
```

```
# Naive Bayes
naive_bayes = MultinomialNB()
naive_bayes.fit(X_train, y_train)
```

▾ MultinomialNB

```
# check prior
print(naive_bayes.class_log_prior_[1])

# check log likelihood
print('\n')
print(naive_bayes.feature_log_prob_)
```

```
-2.108220081038027


[[-10.05719435 -10.05719435 -10.05719435 ...  -9.75727968 -10.05719435
  -10.05719435]
 [-10.02099864  -9.71954129 -10.02099864 ... -10.02099864 -10.02099864
  -10.02099864]
 [ -9.94887707 -10.37882928 -10.37882928 ... -10.37882928 -10.37882928
   -9.8220513 ]
 [ -9.92897393  -9.92897393  -9.92897393 ...  -9.92897393  -9.92897393
   -9.92897393]
 [-10.3211675  -10.3211675  -10.02051945 ... -10.3211675  -10.0241105
  -10.3211675 ]
 [ -9.84107668  -9.84107668  -9.84107668 ...  -9.84107668  -9.84107668
   -9.84107668]]
```

```
pred = naive_bayes.predict(X_test)

# confusion matrix
print(confusion_matrix(y_test, pred))
```

```
[[ 198    6  182    0  199    0]
 [   8  125  216    0  218    0]
 [   1    0 1391    0   46    0]
 [   0    0  248   20   58    0]
 [   3    1   79    0 1132    0]
 [   0    3   99    0   57    2]]
```

```
print('accuracy score: ', accuracy_score(y_test, pred))
print('\n\t\t sadness\t anger\t love\t surprise\t fear\t happy')
print('\nprecision score: ', precision_score(y_test, pred, average=None))
print('\nrecall score:', recall_score(y_test, pred, average=None))
print('\nf1 score: ', f1_score(y_test, pred, average=None))
```

```
accuracy score:  0.668219944082013

                 sadness         anger   love    surprise        fear    happy

precision score:  [0.94285714 0.92592593 0.62799097 1.         0.6619883  1.        ]

recall score: [0.33846154 0.22045855 0.96731572 0.06134969 0.93168724 0.01242236]

f1 score:  [0.49811321 0.35612536 0.76156584 0.11560694 0.77401709 0.02453988]
```

The results of the Naive Bayes technique are a little underwhelming. This method does assume that all of the predictors are independent (hence the name "naive"), which may cause some precision and accuracy issues. Below is a more neatly organized table of the results.

```
print(classification_report(y_test, pred))
```

```
              precision    recall  f1-score   support

       anger       0.94      0.34      0.50       585
        fear       0.93      0.22      0.36       567
       happy       0.63      0.97      0.76      1438
        love       1.00      0.06      0.12       326
     sadness       0.66      0.93      0.77      1215
    surprise       1.00      0.01      0.02       161

    accuracy                           0.67      4292
   macro avg       0.86      0.42      0.42      4292
weighted avg       0.76      0.67      0.60      4292
```

```
# mis-classified items
y_test[y_test != pred]
```

```
16441    surprise
14450        love
```

```
1727     surprise
3258     surprise
4003         fear
           ...
2264         love
18613        love
16028        fear
10454     sadness
3654         fear
Name: Emotion, Length: 1424, dtype: object
```

Since there are many "emotion" classifications, achieving perfection is much harder. Not surprisingly, more complex emotions are often misidentified. The Naive Bayes approach seems to be lacking a bit on this data set. The accuracy score is pretty low, which is not ideal. This means that the model did, in fact, learn something (these numbers are better than random chance), but is not as reliable as it could be. Naive Bayes has high bias, which means it tends to make assumptions about the data that may not necessarily be true. This can lead to incorrect predictions down the line.

Second Try

```
# load data set
df = pd.read_csv(io.StringIO(uploaded['Emotion_final.csv'].decode('utf-8')), header=0, encoding='latin-1')

X = df.Text
y = df.Emotion

# remove "I" and "feel" (common words)
df['Text'].replace('I|i|her|she', ' ', regex=True, inplace=True)
df['Text'].replace('feel', ' ', regex=True, inplace=True)
df['Text'].replace(',.', ' ', regex=True, inplace=True)


X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, train_size=0.8, random_state=1234)

X_train = vectorizer.fit_transform(X_train)
X_test = vectorizer.transform(X_test)

print(naive_bayes.fit(X_train, y_train))

pred = naive_bayes.predict(X_test)
print('accuracy score: ', accuracy_score(y_test, pred))
print('\n\t\t sadness\t anger\t love\t surprise\t fear\t happy')
print('\nprecision score: ', precision_score(y_test, pred, average=None))
print('\nrecall score:', recall_score(y_test, pred, average=None))
print('\nf1 score: ', f1_score(y_test, pred, average=None))

print(classification_report(y_test, pred))
```

```
MultinomialNB()
accuracy score:  0.6600652376514445

                 sadness         anger   love    surprise        fear    happy

precision score:  [0.91479821 0.90769231 0.61052166 1.         0.66988533 1.        ]

recall score: [0.34871795 0.20811287 0.96036161 0.04907975 0.91358025 0.02484472]

f1 score:  [0.5049505  0.33859397 0.74648649 0.09356725 0.7729805  0.04848485]
              precision    recall  f1-score   support

       anger       0.91      0.35      0.50       585
        fear       0.91      0.21      0.34       567
       happy       0.61      0.96      0.75      1438
        love       1.00      0.05      0.09       326
     sadness       0.67      0.91      0.77      1215
    surprise       1.00      0.02      0.05       161

    accuracy                           0.66      4292
   macro avg       0.85      0.42      0.42      4292
weighted avg       0.75      0.66      0.59      4292
```

The removal of common words and punctuation made little change, and much of the changes it made were negative. The accuracy of anger and fear went down slightly. Some of the F1 scores went up or down by 0.01, which is not very significant.

Third Try

```
# load data set
df = pd.read_csv(io.StringIO(uploaded['Emotion_final.csv'].decode('utf-8')), header=0, encoding='latin-1')

X = df.Text
y = df.Emotion

# remove "I" and "feel" (common words)
df['Text'].replace('I|i|her|she', ' noun ', regex=True, inplace=True)
df['Text'].replace('feel', ' am very ', regex=True, inplace=True)
df['Text'].replace(',.!?', ' punct ', regex=True, inplace=True)


X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, train_size=0.8, random_state=1234)

X_train = vectorizer.fit_transform(X_train)
X_test = vectorizer.transform(X_test)

print(naive_bayes.fit(X_train, y_train))

pred = naive_bayes.predict(X_test)
print('accuracy score: ', accuracy_score(y_test, pred))
print('\n\t\t sadness\t anger\t love\t surprise\t fear\t happy')
print('\nprecision score: ', precision_score(y_test, pred, average=None))
print('\nrecall score:', recall_score(y_test, pred, average=None))
print('\nf1 score: ', f1_score(y_test, pred, average=None))

print(classification_report(y_test, pred))
```

```
MultinomialNB()
accuracy score:  0.6439888164026095

                 sadness         anger   love    surprise        fear    happy

precision score:  [0.90810811 0.93333333 0.59662776 1.         0.65971396 1.        ]

recall score: [0.28717949 0.17283951 0.9596662  0.03067485 0.91111111 0.00621118]

f1 score:  [0.43636364 0.29166667 0.73580379 0.05952381 0.76529554 0.01234568]
              precision    recall  f1-score   support

       anger       0.91      0.29      0.44       585
        fear       0.93      0.17      0.29       567
       happy       0.60      0.96      0.74      1438
        love       1.00      0.03      0.06       326
     sadness       0.66      0.91      0.77      1215
    surprise       1.00      0.01      0.01       161

    accuracy                           0.64      4292
   macro avg       0.85      0.39      0.38      4292
weighted avg       0.75      0.64      0.57      4292
```

Replacing common words and punctuation with labels is worse than removing them entirely.

Forth Try

```
# load data set
df = pd.read_csv(io.StringIO(uploaded['Emotion_final.csv'].decode('utf-8')), header=0, encoding='latin-1')
vectorizer = TfidfVectorizer(stop_words=None)

X = df.Text
y = df.Emotion

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, train_size=0.8, random_state=1234)

X_train = vectorizer.fit_transform(X_train)
X_test = vectorizer.transform(X_test)

print(naive_bayes.fit(X_train, y_train))

pred = naive_bayes.predict(X_test)
print('accuracy score: ', accuracy_score(y_test, pred))
print('\n\t\t sadness\t anger\t love\t surprise\t fear\t happy')
```

```
print('\nprecision score: ', precision_score(y_test, pred, average=None))
print('\nrecall score:', recall_score(y_test, pred, average=None))
print('\nf1 score: ', f1_score(y_test, pred, average=None))

print(classification_report(y_test, pred))
```

```
MultinomialNB()
accuracy score:  0.630475302889096

                 sadness         anger    love    surprise        fear    happy

precision score:  [0.95798319 0.94366197 0.58835846 1.         0.65263158 0.        ]

recall score: [0.19487179 0.11816578 0.97705146 0.01226994 0.91851852 0.        ]

f1 score:  [0.32386364 0.21003135 0.73444851 0.02424242 0.76307692 0.        ]
              precision    recall  f1-score   support

       anger       0.96      0.19      0.32       585
        fear       0.94      0.12      0.21       567
       happy       0.59      0.98      0.73      1438
        love       1.00      0.01      0.02       326
     sadness       0.65      0.92      0.76      1215
    surprise       0.00      0.00      0.00       161

    accuracy                           0.63      4292
   macro avg       0.69      0.37      0.34      4292
weighted avg       0.71      0.63      0.54      4292

/usr/local/lib/python3.9/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Precision is ill-defined and bei
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.9/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Precision and F-score are ill-de
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.9/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Precision and F-score are ill-de
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.9/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Precision and F-score are ill-de
  _warn_prf(average, modifier, msg_start, len(result))
```

Keeping stopwords in the text actually improves the precision slightly for anger and fear, but worsens for happiness, sadness, and surprise. This is a very interesting development. In addition, the overall accuracy and F1-score seem to decrease. Improving accuracy for this data set may not be achieved by manipulating the input. The first try with the Naive Bayes algorithm was the best.

.

Logistic Regression

Logistic Regression is another linear classifier. A decision boundary, the linear combination of the parameters, separates the classes.

Pipelining is a technique that allows for easy adjustment of parameters in a dataset. It makes experimenting with data set attributes easier. Without any tweaks, it will produce the same results as the standard Logistic Regression model.

```
# set up X and y
X = df.Text
y = df.Emotion

# divide into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, train_size=0.8, random_state=1234)

# vectorizer
vectorizer = TfidfVectorizer()
X_train = vectorizer.fit_transform(X_train)
X_test = vectorizer.transform(X_test)


# we have already split the data into train and test sets under Naive Bayes,
# so we can reuse it here

classifier = LogisticRegression(solver='lbfgs', max_iter = 3000, class_weight='balanced')
classifier.fit(X_train, y_train)
```

```
LogisticRegression
```

```
pred = classifier.predict(X_test)
print('accuracy score: ', accuracy_score(y_test, pred))
print('precision score: ', precision_score(y_test, pred, average=None))
print('recall score: ', recall_score(y_test, pred, average=None))
print('f1 score: ', f1_score(y_test, pred, average=None))
probs = classifier.predict_proba(X_test)
print(classification_report(y_test, pred))
print('log loss: ', log_loss(y_test, probs))
```

```
accuracy score:  0.8676607642124884
precision score:  [0.84236453 0.85082873 0.93378995 0.69614512 0.92586207 0.62666667]
recall score:  [0.87692308 0.81481481 0.85326843 0.94171779 0.88395062 0.8757764 ]
f1 score:  [0.85929648 0.83243243 0.89171512 0.80052151 0.90442105 0.73056995]
              precision    recall  f1-score   support

       anger       0.84      0.88      0.86       585
        fear       0.85      0.81      0.83       567
       happy       0.93      0.85      0.89      1438
        love       0.70      0.94      0.80       326
     sadness       0.93      0.88      0.90      1215
    surprise       0.63      0.88      0.73       161

    accuracy                           0.87      4292
   macro avg       0.81      0.87      0.84      4292
weighted avg       0.88      0.87      0.87      4292

log loss:  0.6969705450440576
```

The results of the Logistic Regression algorithm are not as good as the Naive Bayes. This is very interesting, as Naive Bayes is a much more simplistic algorithm that tends to work better with smaller sample. This data set is relatively large (21,460 data pairs). Logistic regression tends to have higher variance than Naive Bayes does, which can lead to overfitting the training data. This can result in skewed results.

.

Neural Network

```
vectorizer = TfidfVectorizer(stop_words=list(stopwords), binary=True)
```

```
df.columns = df.columns.str.strip()
X = vectorizer.fit_transform(df.Text)
y = df.Emotion
```

```
pipe1 = Pipeline([
        ('tfidf', TfidfVectorizer()),
        ('neuralnet', MLPClassifier(solver='lbfgs', alpha=1e-5,
                   hidden_layer_sizes=(15, 7), random_state=1, max_iter=3000)),
         ])

pipe1.fit(df.Text, df.Emotion)
```

```
Pipeline
TfidfVectorizer
MLPClassifier
```

```
pred = pipe1.predict(df.Text)

from sklearn import metrics
print(metrics.classification_report(df.Emotion, pred))

print("Confusion matrix:\n", metrics.confusion_matrix(df.Emotion, pred))

import numpy as np
print("\nOverall accuracy: ", np.mean(pred==df.Emotion))
```

```
              precision    recall  f1-score   support

       anger       1.00      1.00      1.00      2993
```

```
        fear       1.00      1.00      1.00      2652
       happy       1.00      1.00      1.00      7029
        love       0.99      0.99      0.99      1641
     sadness       1.00      1.00      1.00      6265
    surprise       0.99      1.00      0.99       879

    accuracy                           1.00     21459
   macro avg       1.00      1.00      1.00     21459
weighted avg       1.00      1.00      1.00     21459

Confusion matrix:
 [[2984    5    0    0    4    0]
 [   0 2645    0    0    1    6]
 [   0    0 7010   16    0    3]
 [   0    0   13 1628    0    0]
 [   0    2    0    0 6263    0]
 [   0    2    0    0    0  877]]

Overall accuracy:  0.9975767743138078
```

The neural network did a much better job at prediction than the Naive Bayes and logistic regression. This is because there is a lot of data available for training. The processing power of neural networks is also much higher than the other two algorithms, ultimately resulting in improved performance (in this case). Since there is a lot of data and the relationship between a piece of text and its tone (or emotion being evoked) is rather complex, the neural network technique is the best choice here.

.

Conclusion

The Naive Bayes, Logistic Regression, and Neural Network algorithms all have their uses. Naive Bayes works best for smaller data sets and is computationally simple. Logistic Regression is best used when the classes in the data set can be separated linearly. Neural Networks work best on large sets of more complex data, like the one used here. The neural network algortihm provided the best results with the given data set. The accuracy, precision, and f1-scores were all consistently higher with the neural network.

✓ 0s completed at 2:49 PM