# Project-7

April 2, 2023

[1]:
```
#https://www.kaggle.com/datasets/lakshmi25npathi/
 ↪imdb-dataset-of-50k-movie-reviews?resource=download

'''
This project classifies a data set of 50,000 imdb movie reviews as positive or␣
 ↪negative based on their text review.
It uses Naive Bayes, Logistic Regression, and Neural Networks.
'''
```

[1]:
```
'\nThis project classifies a data set of 50,000 imdb movie reviews as positive
or negative based on their text review. \nIt uses Naive Bayes, Logistic
Regression, and Neural Networks.\n'
```

[2]:
```
#Imports
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
from sklearn.metrics import accuracy_score, precision_score, recall_score,␣
 ↪f1_score, confusion_matrix
from sklearn.metrics import classification_report
from sklearn.linear_model import LogisticRegression
from sklearn.neural_network import MLPClassifier

#Read Dataset
df = pd.read_csv('IMDB Dataset.csv', header=0)
df.head(10)
```

[2]:

| | review | sentiment |
|---|---|---|
| 0 | One of the other reviewers has mentioned that … | positive |
| 1 | A wonderful little production. <br /><br />The… | positive |
| 2 | I thought this was a wonderful way to spend ti… | positive |
| 3 | Basically there's a family where a little boy … | negative |
| 4 | Petter Mattei's "Love in the Time of Money" is… | positive |
| 5 | Probably my all-time favorite movie, a story o… | positive |
| 6 | I sure would like to see a resurrection of a u… | positive |
| 7 | This show was an amazing, fresh & innovative i… | negative |

```
8  Encouraged by the positive comments about this…  negative
9  If you like original gut wrenching laughter yo…  positive
```

[3]:
```
#Naive Bayes

#Preprocessing
stopwords = set(stopwords.words('english'))
vectorizer = TfidfVectorizer(stop_words=stopwords)

#Isolate X and y
X = df.review
y = df.sentiment

#Test-Train Split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪train_size=0.8, random_state=1337)

#Apply tfidf vectorizer
X_train = vectorizer.fit_transform(X_train)  # fit and transform the train data
X_test = vectorizer.transform(X_test)        # transform only the test data

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

#Predict
predNB = naive_bayes.predict(X_test)

#Print results
print(classification_report(y_test, predNB))
```

```
              precision    recall  f1-score   support

    negative       0.86      0.88      0.87      5016
    positive       0.87      0.85      0.86      4984

    accuracy                           0.87     10000
   macro avg       0.87      0.87      0.87     10000
weighted avg       0.87      0.87      0.87     10000
```

[4]:
```
#Logistic Regression

#Train Logistic Regression
clf = LogisticRegression(C=2.5, n_jobs=4, solver='lbfgs', random_state=17,␣
 ↪verbose=1)
clf.fit(X_train, y_train)
```

```
#Predict
predLR = clf.predict(X_test)

#Print Results
print(classification_report(y_test, predLR))
```

```
[Parallel(n_jobs=4)]: Using backend LokyBackend with 4 concurrent workers.
[Parallel(n_jobs=4)]: Done   1 out of   1 | elapsed:    3.0s finished
```

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| negative | 0.91 | 0.88 | 0.90 | 5016 |
| positive | 0.89 | 0.91 | 0.90 | 4984 |
| accuracy | | | 0.90 | 10000 |
| macro avg | 0.90 | 0.90 | 0.90 | 10000 |
| weighted avg | 0.90 | 0.90 | 0.90 | 10000 |

[5]:
```
#Neural Networks

#Train on data
classifier = MLPClassifier(solver='lbfgs', alpha=1e-5, hidden_layer_sizes=(20,␣
 ↪10), random_state=1)
classifier.fit(X_train, y_train)

predNN = classifier.predict(X_test)

print(classification_report(y_test, predNN))
```

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| negative | 0.88 | 0.87 | 0.87 | 5016 |
| positive | 0.87 | 0.88 | 0.88 | 4984 |
| accuracy | | | 0.87 | 10000 |
| macro avg | 0.87 | 0.87 | 0.87 | 10000 |
| weighted avg | 0.87 | 0.87 | 0.87 | 10000 |

[6]:
```
#Write up - Analysis of various approaches
'''
In general the most accurate approach was Logistic Regression. It had the␣
 ↪overall best precision, recall, and f1 score.
The neural network and Naive Bayes approach are both really similar, however␣
 ↪originally when running the neural network with (15, 2)
```

```
layers it resulted in a horrible ~.50 precision score, it only started matching␣
 ↪Naive bayes at a (20, 10) layer of nodes.
The Neural Network is also magnitudes slower to run since it needs more layers␣
 ↪to get a decent accuracy.

Overall I'd say for the current data Logistic Regression did the best, followed␣
 ↪by Naive Bayes, then Neural Networks.
However, by changing the layers on the neural network it's possible to get even␣
 ↪higher levels of accuracy.
'''
```

[6]: "\nIn general the most accurate approach was Logistic Regression. It had the overall best precision, recall, and f1 score.\nThe neural network and Naive Bayes approach are both really similar, however originally when running the neural network with (15, 2)\nlayers it resulted in a horrible ~.50 precision score, it only started matching Naive bayes at a (20, 10) layer of nodes.\nThe Neural Network is also magnitudes slower to run since it needs more layers to get a decent accuracy.\n\nOverall I'd say for the current data Logistic Regression did the best, followed by Naive Bayes, then Neural Networks.\nHowever, by changing the layers on the neural network it's possible to get even higher levels of accuracy.\n"