textclassification

April 2, 2023

1 Portfolio Assignment: Text Classification

Linus Fackler

The dataset used in this notebook can be found here: https://www.kaggle.com/datasets/lakshmi25npathi/imdb-dataset-of-50k-movie-reviews This dataset contains of 50k imdb movie reviews. We can train the model to predict whether a movie review is positive or negative.

1.0.1 Imports

```
[ ]: import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
import seaborn as sb
from nltk.corpus import stopwords
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.linear_model import LogisticRegression
from sklearn.neural_network import MLPClassifier
from sklearn.pipeline import Pipeline
from sklearn.metrics import accuracy_score, confusion_matrix,␣
 ↪classification_report, ConfusionMatrixDisplay
from wordcloud import WordCloud, STOPWORDS
```

1.0.2 Read data

```
[ ]: df = pd.read_csv('Data/IMDB Dataset.csv')
print(df.shape)
df.head()
```

```
(50000, 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
```

[ ]:
```
df.dtypes
```

[ ]:
```
review       object
sentiment    object
dtype: object
```

We are going to change the column name from sentiment to 'is_positive', meaning a movie review is positive. Value 'positive' will be changed to 1 and 'negative' to 0.

[ ]:
```
# rename column name 'sentiment' to 'is_positive'
df.rename(columns={'sentiment':'is_positive'}, inplace=True)

# positive -> 1
df.loc[df['is_positive'] == 'positive', 'is_positive'] = 1

# negative -> 0
df.loc[df['is_positive'] == 'negative', 'is_positive'] = 0

# converting type of 'is_positive' from object to int
df['is_positive'] = df['is_positive'].astype(str).astype(int)

df.head()
```

[ ]:
```
                                              review  is_positive
0  One of the other reviewers has mentioned that …            1
1  A wonderful little production. <br /><br />The…            1
2  I thought this was a wonderful way to spend ti…            1
3  Basically there's a family where a little boy …            0
4  Petter Mattei's "Love in the Time of Money" is…            1
```

There are some words, such as 'br', that we should remove from the data first.

1.0.3 Cleaning up data

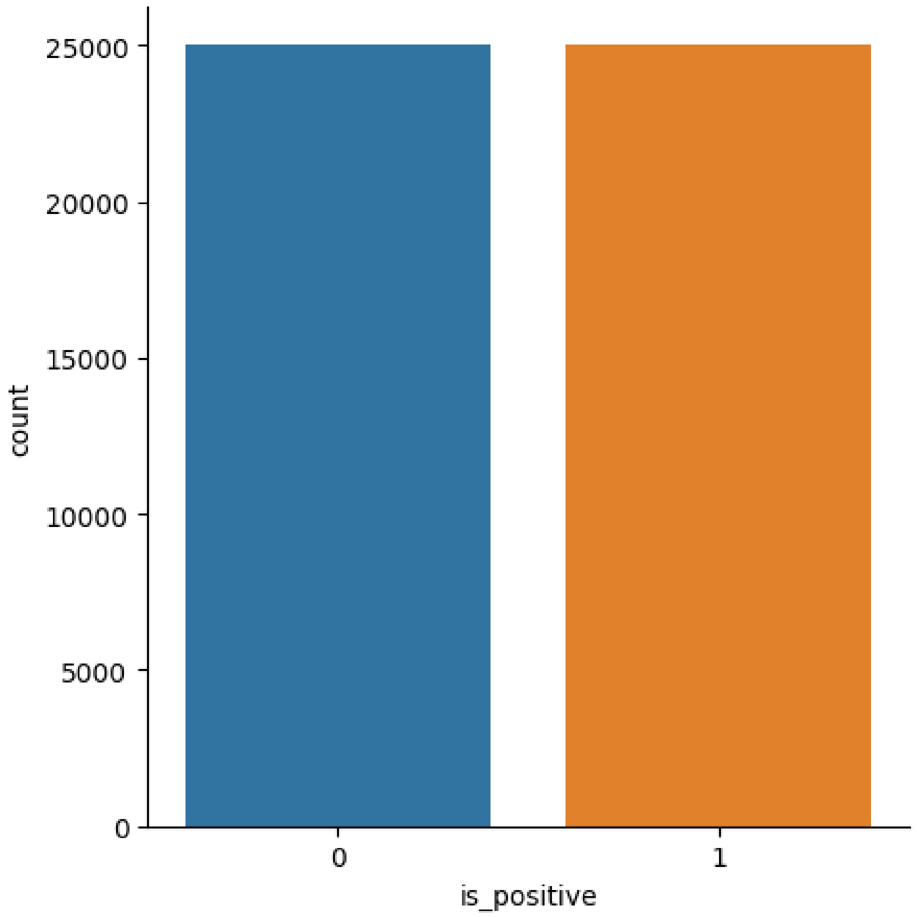
[ ]:
```
df = df.replace('<br />', ' ', regex=True)

df.head()
```

[ ]:
```
                                              review  is_positive
0  One of the other reviewers has mentioned that …            1
1  A wonderful little production.   The filming t…            1
2  I thought this was a wonderful way to spend ti…            1
3  Basically there's a family where a little boy …            0
4  Petter Mattei's "Love in the Time of Money" is…            1
```

```
[ ]: sb.catplot(x='is_positive', kind='count', data=df)
```

```
[ ]: <seaborn.axisgrid.FacetGrid at 0x2ae69f970>
```



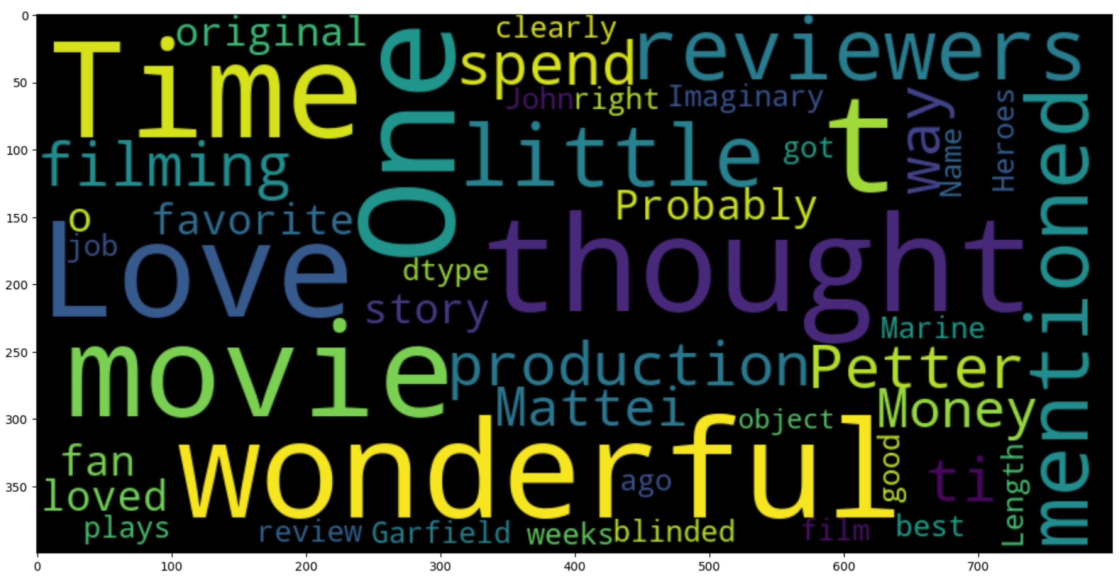
We can see that this dataset has an equal distribution of positive and negative reviews.

1.0.4 Visualize most important words

```
[ ]: wordcloud = WordCloud(background_color = 'black', stopwords = STOPWORDS,␣
      ↪max_words = 100, max_font_size = 100, random_state = 15, width = 800, height␣
      ↪= 400)

     plt.figure(figsize = (16, 12))
     wordcloud.generate(str(df.loc[df['is_positive'] == 1, 'review']))
     plt.imshow(wordcloud)
```

```
[ ]: <matplotlib.image.AxesImage at 0x2bb183be0>
```



1.0.5 tf-idf processing and train - test split

```
[ ]: x = df.review
     y = df.is_positive

     x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.25,␣
      ↪random_state = 1234)
     x_train.shape
```

```
[ ]: (37500,)
```

1.0.6 tfidf vectorizer

```
[ ]: vectorizer = TfidfVectorizer(ngram_range=(1, 2), max_features=5000, min_df=2)

     x_train = vectorizer.fit_transform(x_train)
     x_test = vectorizer.transform(x_test)
```

1.1 Naive Bayes

```
[ ]: nb = MultinomialNB()
     nb.fit(x_train, y_train)
```

```
[ ]: MultinomialNB()
```

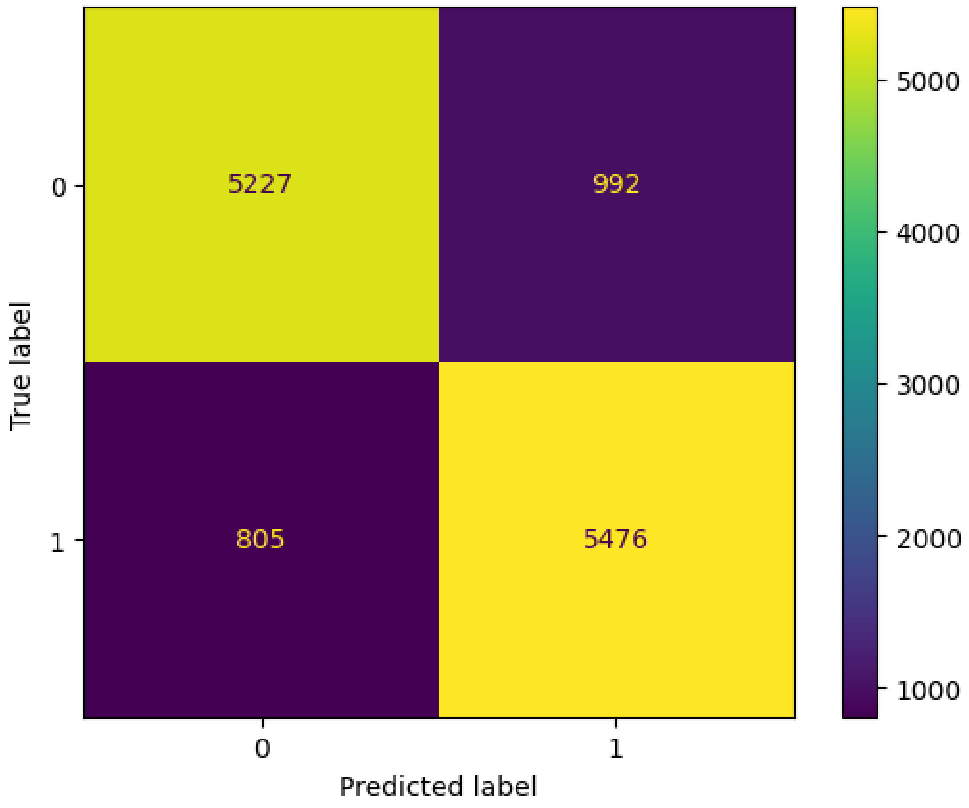
```
[ ]: pred_nb = nb.predict(x_test)
```

1.1.1 Confusion Matrix of Naive Payes prediction

```
[ ]: cm_nb = confusion_matrix(y_test, pred_nb)
     disp = ConfusionMatrixDisplay(confusion_matrix=cm_nb)

     disp.plot()
```

```
[ ]: <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x2bb207130>
```



1.1.2 Accuracy of Naive Bayes prediction

```
[ ]: acc_nb = accuracy_score(y_test, pred_nb)

     print(classification_report(y_test, pred_nb))
```

```
              precision    recall  f1-score   support

           0       0.87      0.84      0.85      6219
           1       0.85      0.87      0.86      6281

    accuracy                           0.86     12500
```

```
   macro avg       0.86      0.86      0.86     12500
weighted avg       0.86      0.86      0.86     12500
```

1.2 Logistic Regression

```
[ ]: lr = LogisticRegression(C=2.5, n_jobs=4, solver='lbfgs', random_state=17,␣
      ↪verbose=1)
     lr.fit(x_train, y_train)
```

```
[Parallel(n_jobs=4)]: Using backend LokyBackend with 4 concurrent workers.

RUNNING THE L-BFGS-B CODE

           * * *

Machine precision = 2.220D-16
 N =         5001     M =           10

At X0         0 variables are exactly at the bounds

At iterate    0    f=  2.59930D+04    |proj g|=  1.70302D+02

 This problem is unconstrained.


At iterate   50    f=  9.70471D+03    |proj g|=  2.32109D+01

At iterate  100    f=  9.68890D+03    |proj g|=  9.07050D-02

           * * *

Tit   = total number of iterations
Tnf   = total number of function evaluations
Tnint = total number of segments explored during Cauchy searches
Skip  = number of BFGS updates skipped
Nact  = number of active bounds at final generalized Cauchy point
Projg = norm of the final projected gradient
F     = final function value

           * * *

   N    Tit     Tnf  Tnint  Skip  Nact     Projg        F
 5001    100    116      1     0     0   9.070D-02   9.689D+03
  F =   9688.8960340423655

STOP: TOTAL NO. of ITERATIONS REACHED LIMIT

/opt/homebrew/lib/python3.10/site-
packages/sklearn/linear_model/_logistic.py:458: ConvergenceWarning: lbfgs failed
```

```
to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
[Parallel(n_jobs=4)]: Done   1 out of   1 | elapsed:    1.6s finished
```

```
[ ]: LogisticRegression(C=2.5, n_jobs=4, random_state=17, verbose=1)
```

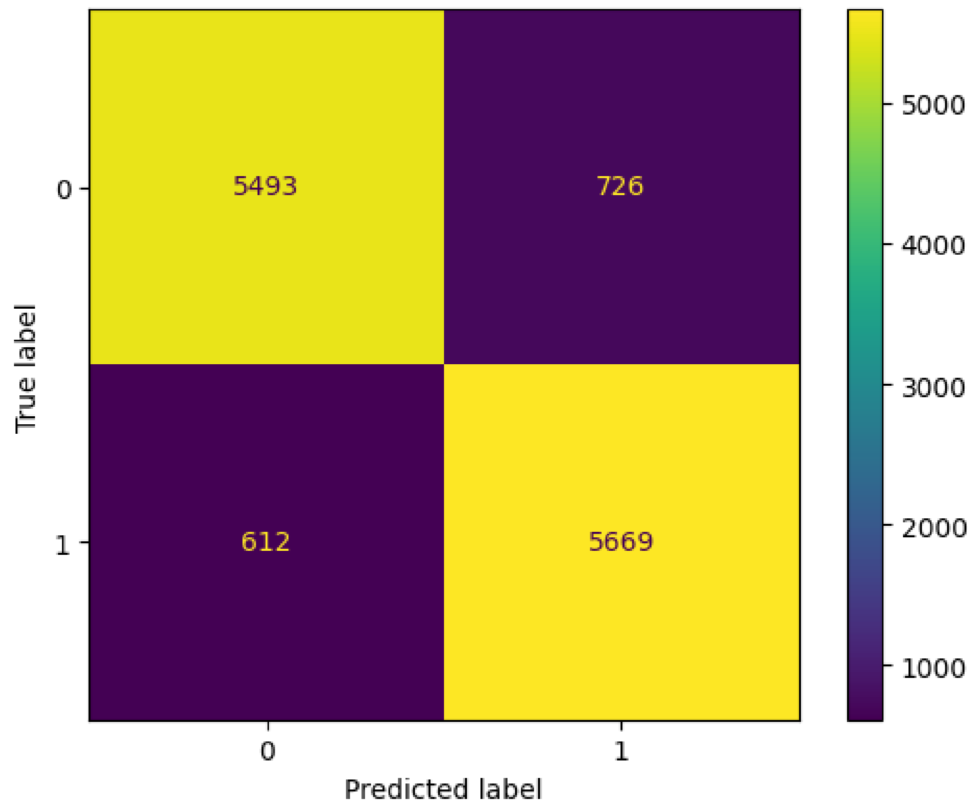
```
[ ]: pred_lr = lr.predict(x_test)
```

1.2.1 Confusion Matrix of Logistic Regression

```
[ ]: cm_lr = confusion_matrix(y_test, pred_lr)
     disp = ConfusionMatrixDisplay(confusion_matrix=cm_lr)

     disp.plot()
```

```
[ ]: <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x2c0f63310>
```



1.2.2 Accuracy of Logistic Regression

```
[ ]: acc_lr = accuracy_score(y_test, pred_lr)

     print(classification_report(y_test, pred_lr))
```

```
              precision    recall  f1-score   support

           0       0.90      0.88      0.89      6219
           1       0.89      0.90      0.89      6281

    accuracy                           0.89     12500
   macro avg       0.89      0.89      0.89     12500
weighted avg       0.89      0.89      0.89     12500
```

1.3 Neural Networks

```
[ ]: nn = MLPClassifier(hidden_layer_sizes=(8,8,8), activation='relu',␣
      ↪solver='adam', max_iter=500)
     nn.fit(x_train, y_train)
```

```
[ ]: MLPClassifier(hidden_layer_sizes=(8, 8, 8), max_iter=500)
```

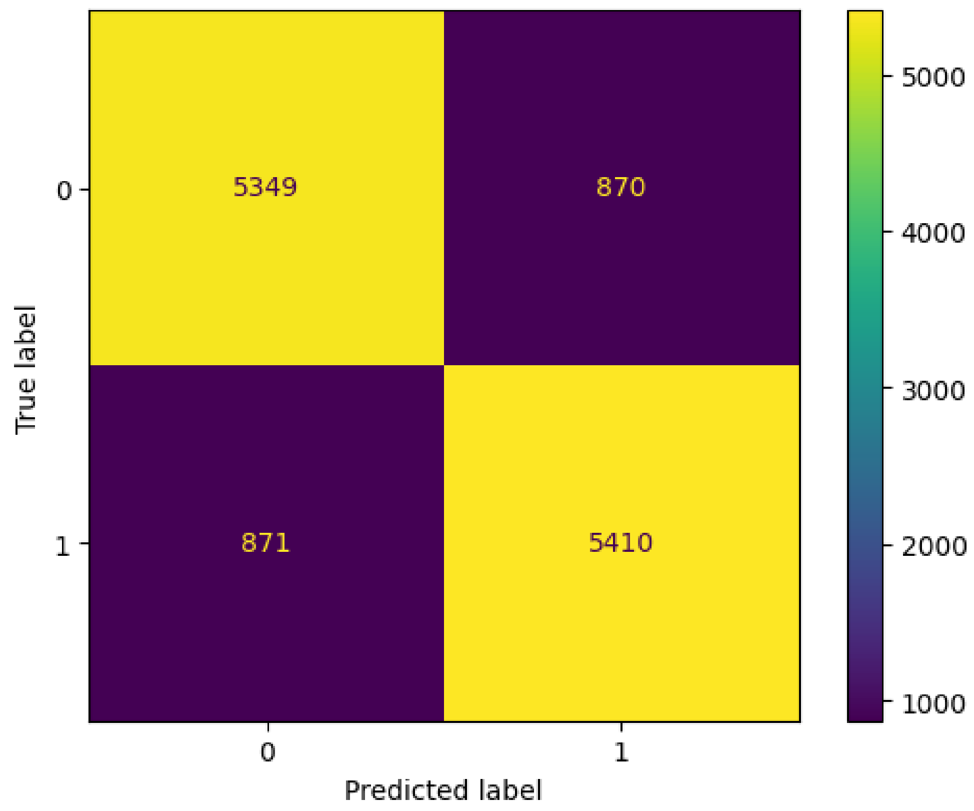
```
[ ]: pred_nn = nn.predict(x_test)
```

1.3.1 Confusion Matrix of Neural Networks

```
[ ]: cm_nn = confusion_matrix(y_test, pred_nn)
     disp = ConfusionMatrixDisplay(confusion_matrix=cm_nn)

     disp.plot()
```

```
[ ]: <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x2c599eda0>
```



1.3.2 Accuracy of Neural Networks

[ ]:
```
acc_nn = accuracy_score(y_test, pred_nn)

print(classification_report(y_test, pred_nn))
```

```
              precision    recall  f1-score   support

           0       0.86      0.86      0.86      6219
           1       0.86      0.86      0.86      6281

    accuracy                           0.86     12500
   macro avg       0.86      0.86      0.86     12500
weighted avg       0.86      0.86      0.86     12500
```

1.4 Comparison of all 3 approaches

```
[ ]: print('Accuracy of Naive Bayes approach:\t\t', acc_nb)
     print('Accuracy of Logistic Regression approach:\t', acc_lr)
     print('Accuracy of Neural Networks approach:\t\t', acc_nn)
```

```
Accuracy of Naive Bayes approach:                0.85624
Accuracy of Logistic Regression approach:        0.89296
Accuracy of Neural Networks approach:            0.86072
```

1.4.1 Naive Bayes

This is a simple and fast algorithm that works well with small datasets and is widely used for text classification. It assumes that the features, in this case words, are independent of each other, which is not always true in real-world scenarios. It still often performs well and can be used as a baseline for comparison. In this case, this method achieved a ~85.6% accuracy.

1.4.2 Logistic Regression

This is a linear model that is often used for binary classification tasks, such as sentiment analysis or spam filtering, in the case of text classification. It leans the relationship between the input features, in this case words in a movie review, and the ouput variable, whether the review was positive or negative, by fitting a logistic function. It achieved an accuracy of ~89%.

1.4.3 Neural Networks

This is a powerful ML algorithm that can learn complex non-linear relationships between input and output variables. They're known to be effective for a wide range of tasks, inclusind text classification. They require a larger input of data though, compared to other algorithms. In our case, our dataset was decently large, which is why we still achieved a good accuracy of ~86%. This could have potentially been higher with a larger dataset.

1.4.4 Why Logistic Regression achieved the highest scores

Logistic Regression can handle both numerical and categorical features. In text classification, the input features are typically words, which are categorical variables. Naive Bayes can also handle categorical features, but it assumes that the features are conditionally indepenendent given the class label. This may not be true in all cases. Neural Networks can also handle categorical features, but might require more training data to achieve optimal performance. Another reason why logistic regression outperformed the other algorithms is that it can handle high-dimensional and sparse datasets. In text classification, the input feature space can be very large and sparse, which can make it challenging to find meaningful patterns in the data. Naive Bayes can perform well on high-dimensional/sparse datasets, but might not be able to capture complex relationships between input features. Neural networks can learn complex non-linear relationships, but might require more data and time to achieve optimal performance. Logistic regression can handle high-dimensional/sparse datasets by using regularization techniques, such as L1 or L2 regularization, which can prevent overfitting and improve generalization.