IMDB Logistic Regression

Louis Bailey

Import Libraries

In [1]:
```
import pandas as pd

from sklearn.model_selection import cross_val_score, GridSearchCV, train_test_split
from sklearn.metrics import accuracy_score, classification_report

from sklearn.linear_model import LogisticRegression
from imblearn.pipeline import make_pipeline

from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk.stem.porter import PorterStemmer
from sklearn.feature_extraction.text import TfidfVectorizer
import string
```

Import the movie review data

In [2]:
```
data = pd.read_csv('labeledTrainData.tsv', sep='\t', on_bad_lines='skip')
```

In [3]:
```
data.head()
```

Out[3]:

	id	sentiment	review
0	5814_8	1	With all this stuff going down at the moment w...
1	2381_9	1	\The Classic War of the Worlds\" by Timothy Hi...
2	7759_3	0	The film starts with a manager (Nicholas Bell)...
3	3630_4	0	It must be assumed that those who praised this...
4	9495_8	1	Superbly trashy and wondrously unpretentious 8...

Shape, Missing, Duplicates

In [4]:
```
print('Shape:     ', data.shape)
print('Missing:   ', data.isna().sum().sum())
print('Duplicates:', data.duplicated().sum())
```

```
Shape:      (25000, 3)
Missing:    0
Duplicates: 0
```

How many of each positive and negative reviews are there?

In [5]:
```
data.sentiment.value_counts()
```

Out[5]:
```
sentiment
1    12500
0    12500
Name: count, dtype: int64
```

Prepare Text for Model

In [6]:
```
my_text = data.review
```

In [7]:
```
class Prep_Text:
    def __init__(self,text):
        self.text = text

    def to_lower(self):
        self.text =  [x.lower() for x in self.text]

    def remove_punc(self):
        self.text = [''.join(char for char in x if char not in string.punctuation) for x in self.text]

    def remove_stop_words(self):
        stop_words = set(stopwords.words('english'))
        self.text  =  [' '.join(x for x in word_tokenize(words) if x not in stop_words) for words in self.text]

    def get_stems(self):
        porter = PorterStemmer()
        self.text =  [' '.join(porter.stem(word) for word in word_tokenize(char)) for char in self.text]
        return self.text
```

In [8]:
```
#run methods
prep = Prep_Text(my_text)
prep.to_lower()
prep.remove_punc()
prep.remove_stop_words()
```

Train/Test Split

In [9]:
```
X = prep.get_stems()
y = data['sentiment']
```

In [11]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=1)
```

Pipeline & Gridsearch

In [12]:
```
logreg_pipeline = make_pipeline(TfidfVectorizer(),
                                LogisticRegression(random_state=1, max_iter=200))
```

In [13]:
```
search_spaces = {
        'logisticregression': [
        {'logisticregression__C': [0.001, 0.01, 0.1, 1, 10, 100],
         'logisticregression__penalty': ['l1', 'l2'],
         'logisticregression__solver': ['liblinear'],
         'logisticregression__class_weight': [None, 'balanced']},
        {'logisticregression__C': [0.001, 0.01, 0.1, 1, 10, 100],
         'logisticregression__penalty': ['l2'],
         'logisticregression__solver': ['newton-cg', 'lbfgs', 'saga'],
         'logisticregression__class_weight': [None, 'balanced']},
        {'logisticregression__C': [0.001, 0.01, 0.1, 1, 10, 100],
         'logisticregression__penalty': ['elasticnet'],
         'logisticregression__solver': ['saga'],
         'logisticregression__l1_ratio': [0, 0.5, 1],
         'logisticregression__class_weight': [None, 'balanced']}
    ]
}
```

In [14]:
```
logreg_search = GridSearchCV(logreg_pipeline, search_spaces['logisticregression'], cv=3,
                             scoring='accuracy', n_jobs=-1).fit(X_train, y_train)

print(f'Logistic Regression tuned params: {logreg_search.best_params_} \n')
```

```
Logistic Regression tuned params: {'logisticregression__C': 10, 'logisticregression__class_weight': None, 'logisti
cregression__penalty': 'l2', 'logisticregression__solver': 'liblinear'}
```

Logistic Regression

In [15]:
```
tuned_model  = logreg_search.best_estimator_

y_pred       = tuned_model.predict(X_test)
y_pred_train = tuned_model.predict(X_train)
```

In [16]:
```
print("Train Accuracy:", accuracy_score(y_train, y_pred_train))
print("Test Accuracy :", accuracy_score(y_test, y_pred))
```

```
Train Accuracy: 0.9891
Test Accuracy : 0.8858
```

In [17]:
```
print("Classification Report:\n", classification_report(y_test, y_pred))
```

```
Classification Report:
              precision    recall  f1-score   support

           0       0.89      0.88      0.89      2503
           1       0.88      0.89      0.89      2497

    accuracy                           0.89      5000
   macro avg       0.89      0.89      0.89      5000
weighted avg       0.89      0.89      0.89      5000
```

In [18]:
```
print(cross_val_score(tuned_model, X, y, cv=5))
```

```
[0.8874 0.8912 0.8782 0.8922 0.886 ]
```

The train and test accuracies appear close enough to indicate the model is not overfitting.

The classification report indicates the model is very balanced when classifying negative vs positive.

The cross validated scores are very close together indicating the model is consistent across different subsets of the data.

Learning Curve

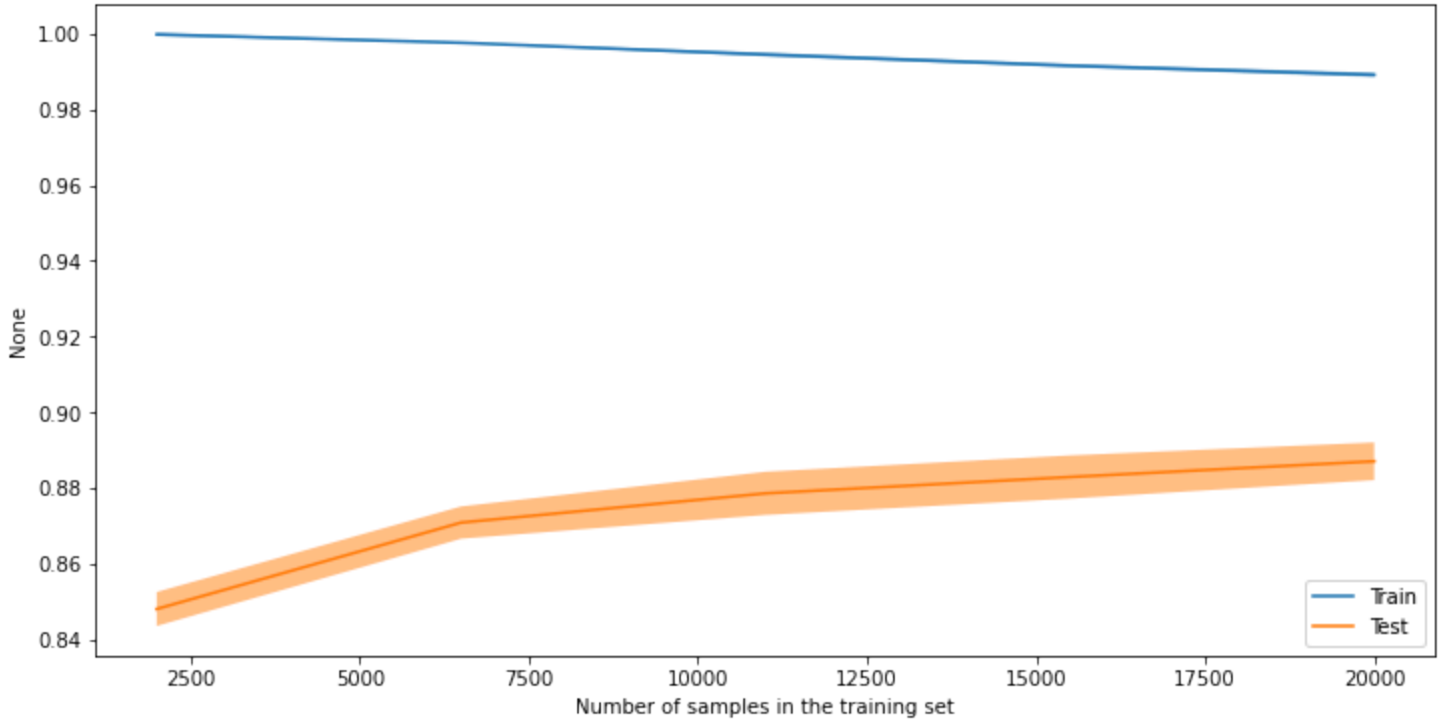
In [20]:

```
import matplotlib.pyplot as plt
from sklearn.model_selection import learning_curve
from sklearn.model_selection import LearningCurveDisplay

train_sizes, train_scores, test_scores = learning_curve(
    tuned_model, X, y, cv=5
)

plt.rcParams['figure.figsize'] = [12, 6]
LearningCurveDisplay(
    train_sizes=train_sizes,
    train_scores=train_scores,
    test_scores=test_scores
).plot()

plt.show()
```



The stabilization of the scores suggests that adding more data is unlikely to drastically improve the model's performance.

Save Tuned Model For Use in Last Airbender Notebook

```
import joblib

joblib.dump(tuned_model, 'tuned_logreg_model_imdb.pkl')
```