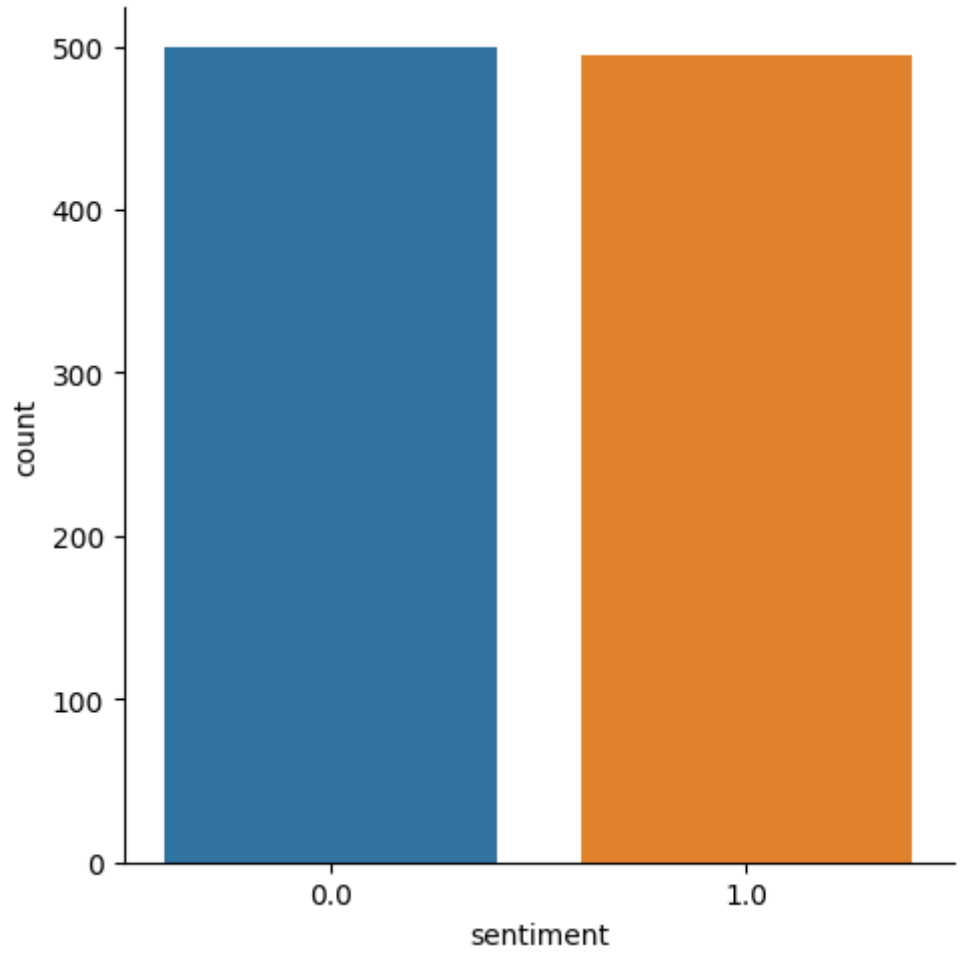
Text Classification by Priyesh Patel

IMDB Review Sentiment Analysis (where 1 is a positive review sentiment and 0 is a negative review sentiment) comes from the UCI Machine Learning Repository. Where the distrution between positive and negative is 50/50. Therefore any accruacy higher then the baseline of 50% we showcase an improved algorithm to classify a review.

```
import seaborn as sb

df = pd.read_csv('IMDB_Sentiment.csv')
df = df.dropna() #remove NAN values

sb.catplot(x='sentiment', kind='count', data=df)
```

```
<seaborn.axisgrid.FacetGrid at 0x7f4748fb6dc0>
```



Naive Bayes Analysis

comparison results between Multinomial and Binomial classifiers

```
positive review size in test data: 128
test size:  249
baseline:  0.5140562248995983
```

Multinomial Classifier Results

```
Confusion Matrix
TP = 102, FP = 26
FN = 25, TN = 96

accuracy score:  0.7951807228915663
precision score:  0.7868852459016393
recall score:  0.7933884297520661
f1 score:  0.7901234567901234
```

Binomial Classifier Results

```
Confusion Matrix
TP = 113, FP = 15
FN = 46, TN = 75

accuracy score:  0.7550200803212851
precision score:  0.8333333333333334
recall score:  0.6198347107438017
f1 score:  0.7109004739336493
```

Analysis

After using two different Naive Bayes Approaches such as Multinomial and Binomial Classifiers to train and test the dataset we can see that there is an improvement in classification. Between the two classifiers the model is not the best with have an average of 77% accuracy between the two however there is still some improvement over flipping a coin. In both the models looking the the confusion matrix there are a lot of false negative and false positive which isn't ideal. I think that this is due to a small train set. With only have around 700 training entry the model might not have enough data to build a optimal model. Overall the Multinomial classifer resulted in almost 80% accuracy.

```
import nltk
nltk.download('stopwords')
```

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

Naive Bayes Using Sklearn for Sentiment Analysis

```
import pandas as pd

df = pd.read_csv('IMDB_Sentiment.csv')
df = df.dropna() #remove NAN values

print('row and column', df.shape)
print(df.head(10))
```

```
row and column (994, 2)
                                              Review  sentiment
0  Not sure who was more lost - the flat characte...        0.0
1  Attempting artiness with black & white and cle...        0.0
2       Very little music or anything to speak of.          0.0
3  The best scene in the movie was when Gerardo i...        1.0
4  The rest of the movie lacks art, charm, meanin...        0.0
5                                Wasted two hours.          0.0
6  Saw the movie today and thought it was a good ...        1.0
7                               A bit predictable.          0.0
8  Loved the casting of Jimmy Buffet as the scien...        1.0
9              And those baby owls were adorable.           1.0
```

Text Preprocessing

```
from nltk.corpus import stopwords
from sklearn.feature_extraction.text import TfidfVectorizer

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


#get the X and y values for each entry
X = df.Review
df['sentiment'] = df['sentiment'].astype(int)
y = df.sentiment

print(X.head())
print(y.head())
```

```
0    Not sure who was more lost - the flat characte...
1    Attempting artiness with black & white and cle...
2         Very little music or anything to speak of.
3    The best scene in the movie was when Gerardo i...
4    The rest of the movie lacks art, charm, meanin...
Name: Review, dtype: object
```

```
0    0
1    0
2    0
3    1
4    0
Name: sentiment, dtype: int64
```

Training and Test Sets 75/25 split

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, train_size=(

X_train.shape
```

```
(745,)
```

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


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

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

```
train size: (745, 2500)
[[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: (249, 2500)
[[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.]]
```

Training the Naive Bayes Classifier Using MulitnomialNB (where features are discrete)

Using the MNB default setting

- alpha: additive (Laplace) smoothing (0 for no smoothing)
- fit_prior: if True, learn priors from data; if false, use a
- class_prior: lets you specify class priors

```
from sklearn.naive_bayes import MultinomialNB

naive_bayes = MultinomialNB()
naive_bayes.fit(X_train, y_train)
```

```
▾ MultinomialNB
MultinomialNB()
```

```
# priors
import math
prior_p = sum(y_train == 1)/len(y_train)
print('prior spam:', prior_p, 'log of prior:', math.log(prior_p))

# the model prior matches the prior calculated above
naive_bayes.class_log_prior_[1]
```

```
prior spam: 0.5020134228187919 log of prior: -0.6891284209650277
-0.6891284209650275
```

```
# what else did it learn from the data?
# the log likelihood of words given the class

naive_bayes.feature_log_prob_
```

```
array([[-7.19080759, -8.19911218, -7.99103515, ..., -8.03272255,
        -7.79414652, -7.78512095],
       [-6.2891433 , -7.97271051, -8.2234015 , ..., -8.2234015 ,
        -8.2234015 , -8.2234015 ]])
```

Evaluating on the test data

```
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, c

# make predictions on the test data
pred = naive_bayes.predict(X_test)

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

# confusion matrix has this form
#     tp   fp
#     fn   tn
```

```
[[102  26]
 [ 25  96]]
```

Here we can see that there are alot of cases of false positive and false negatives which isn't great for the NB Performance.

```
print('accuracy score: ', accuracy_score(y_test, pred))
print('precision score: ', precision_score(y_test, pred))
print('recall score: ', recall_score(y_test, pred))
print('f1 score: ', f1_score(y_test, pred))
```

```
accuracy score:  0.7951807228915663
precision score:  0.7868852459016393
recall score:  0.7933884297520661
f1 score:  0.7901234567901234
```

```
from sklearn.metrics import classification_report
print(classification_report(y_test, pred))
```

```
                  precision    recall  f1-score   support

               0       0.80      0.80      0.80       128
               1       0.79      0.79      0.79       121

        accuracy                           0.80       249
       macro avg       0.80      0.80      0.80       249
    weighted avg       0.80      0.80      0.80       249
```

```
print('positive review size in test data:',y_test[y_test==0].shape[0])
print('test size: ', len(y_test))
baseline = y_test[y_test==0].shape[0] / y_test.shape[0]
print('baseline: ', baseline)
```

```
positive review size in test data: 128
test size:  249
baseline:  0.5140562248995983
```

Now lets use compare the result using a Binomial Classifier instead of the Multinomial classifer

```
# binary=True gives binary data instead of counts
vectorizer_b = TfidfVectorizer()

# set up X and y
X = vectorizer_b.fit_transform(df.Review)
y = df.sentiment

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, train_size=(
```

```
from sklearn.naive_bayes import BernoulliNB

naive_bayes2 = BernoulliNB()
naive_bayes2.fit(X_train, y_train)
```

```
▾ BernoulliNB
BernoulliNB()
```

```
# make predictions on the test data
pred = naive_bayes2.predict(X_test)

# print confusion matrix
from sklearn.metrics import confusion_matrix
confusion_matrix(y_test, pred)
```

```
array([[113,  15],
       [ 46,  75]])
```

```
print('accuracy score: ', accuracy_score(y_test, pred))
print('precision score: ', precision_score(y_test, pred))
print('recall score: ', recall_score(y_test, pred))
print('f1 score: ', f1_score(y_test, pred))
```

```
accuracy score:  0.7550200803212851
precision score:  0.8333333333333334
recall score:  0.6198347107438017
f1 score:  0.7109004739336493
```

Logistic Regression Analysis - with pipelines

Logistic Regression without using pipeline

```
accuracy score:  0.8112449799196787
precision score:  0.7761194029850746
recall score:  0.859504132231405
f1 score:  0.8156862745098039
log loss:  0.5605525961846755
```

Logistic Regression with pipline and default arguments

```
accuracy score:  0.8112449799196787
precision score:  0.7761194029850746
recall score:  0.859504132231405
```

```
f1 score:  0.8156862745098039
log loss:  0.5605525961846755
```

Logistic Regression with pipline and custom arguments

```
Arguments Changes: tfidf__min_df=3, logreg__C=2.0
accuracy:  0.7791164658634538
log loss:  0.5289887343179546

Arguments Changes: tfidf__min_df=1, logreg__C=2.4, logreg__max_iter = 1000
accuracy:  0.8313253012048193
log loss:  0.510320890344046
```

Analysis

After using Logistic Regression with and without piplines we got the exact same result in all categories. However after playing with the pipeline model and implementing custom arguments we can see that some arguments caused a lower accuracy and some have a higher accuracy. Overall it seem that for this dataset this logistic Regression models seem to give higher performance with over 80% accuracy and the highest seen of 83%. In addition to the highest f1 score.

```
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, l
```

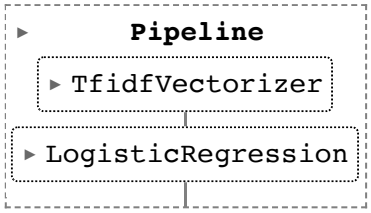
Logistic Regression without using pipelines

```
df = pd.read_csv('IMDB_Sentiment.csv')
df = df.dropna() #remove NAN values

#get the X and y values for each entry
X = df.Review
df['sentiment'] = df['sentiment'].astype(int)
y = df.sentiment

# divide into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, train_size=(

# vectorizer
vectorizer = TfidfVectorizer(binary=True)
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
classifier = LogisticRegression(solver='lbfgs', class_weight='balanced')
classifier.fit(X_train, y_train)

# evaluate
pred = classifier.predict(X_test)
print('accuracy score: ', accuracy_score(y_test, pred))
print('precision score: ', precision_score(y_test, pred))
print('recall score: ', recall_score(y_test, pred))
print('f1 score: ', f1_score(y_test, pred))
probs = classifier.predict_proba(X_test)
print('log loss: ', log_loss(y_test, probs))
```

```
accuracy score:  0.8112449799196787
precision score:  0.7761194029850746
recall score:  0.859504132231405
f1 score:  0.8156862745098039
log loss:  0.5605525961846755
```

```
from sklearn.pipeline import Pipeline

# read in data
df = pd.read_csv('IMDB_Sentiment.csv')
df = df.dropna() #remove NAN values

#get the X and y values for each entry
X = df.Review
df['sentiment'] = df['sentiment'].astype(int)
y = df.sentiment

# divide into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, train_size=(

#use pipeline to transform
pipe1 = Pipeline([
        ('tfidf', TfidfVectorizer(binary=True)),
        ('logreg', LogisticRegression(solver='lbfgs', class_weight='balanced')),
])

pipe1.fit(X_train, y_train)
```



```
pred = pipe1.predict(X_test)
print('accuracy score: ', accuracy_score(y_test, pred))
print('precision score: ', precision_score(y_test, pred))
print('recall score: ', recall_score(y_test, pred))
print('f1 score: ', f1_score(y_test, pred))
probs = pipe1.predict_proba(X_test)
print('log loss: ', log_loss(y_test, probs))
```

```
accuracy score:  0.8112449799196787
precision score:  0.7761194029850746
recall score:  0.859504132231405
f1 score:  0.8156862745098039
log loss:  0.5605525961846755
```

```
#lets see the pipline steps
pipe1.steps
```

```
[('tfidf', TfidfVectorizer(binary=True)),
 ('logreg', LogisticRegression(class_weight='balanced'))]
```

```
#lets see the coeficients of the model
pipe1.named_steps['logreg'].coef_
```

```
array([[ 0.96596455,  0.10156943, -0.11139181, ..., -0.08007848,
        -0.21239022, -0.23018771]])
```

```
#lets see the parameters of the model
pipe1.named_steps['logreg'].get_params()
```

```
{'C': 1.0,
 'class_weight': 'balanced',
 'dual': False,
 'fit_intercept': True,
 'intercept_scaling': 1,
 'l1_ratio': None,
 'max_iter': 100,
 'multi_class': 'auto',
 'n_jobs': None,
 'penalty': 'l2',
 'random_state': None,
 'solver': 'lbfgs',
 'tol': 0.0001,
 'verbose': 0,
 'warm_start': False}
```

```
# to check all the parameters of the pipeline, do this:
pipe1.get_params()
```

```
{'memory': None,
 'steps': [('tfidf', TfidfVectorizer(binary=True)),
  ('logreg', LogisticRegression(class_weight='balanced'))],
```

```
 'verbose': False,
 'tfidf': TfidfVectorizer(binary=True),
 'logreg': LogisticRegression(class_weight='balanced'),
 'tfidf__analyzer': 'word',
 'tfidf__binary': True,
 'tfidf__decode_error': 'strict',
 'tfidf__dtype': numpy.float64,
 'tfidf__encoding': 'utf-8',
 'tfidf__input': 'content',
 'tfidf__lowercase': True,
 'tfidf__max_df': 1.0,
 'tfidf__max_features': None,
 'tfidf__min_df': 1,
 'tfidf__ngram_range': (1, 1),
 'tfidf__norm': 'l2',
 'tfidf__preprocessor': None,
 'tfidf__smooth_idf': True,
 'tfidf__stop_words': None,
 'tfidf__strip_accents': None,
 'tfidf__sublinear_tf': False,
 'tfidf__token_pattern': '(?u)\\b\\w\\w+\\b',
 'tfidf__tokenizer': None,
 'tfidf__use_idf': True,
 'tfidf__vocabulary': None,
 'logreg__C': 1.0,
 'logreg__class_weight': 'balanced',
 'logreg__dual': False,
 'logreg__fit_intercept': True,
 'logreg__intercept_scaling': 1,
 'logreg__l1_ratio': None,
 'logreg__max_iter': 100,
 'logreg__multi_class': 'auto',
 'logreg__n_jobs': None,
 'logreg__penalty': 'l2',
 'logreg__random_state': None,
 'logreg__solver': 'lbfgs',
 'logreg__tol': 0.0001,
 'logreg__verbose': 0,
 'logreg__warm_start': False}
```

```
# change some parameters
pipe1.set_params(tfidf__min_df=3, logreg__C=2.0).fit(X_train, y_train)
pred = pipe1.predict(X_test)
print("accuracy: ", accuracy_score(y_test, pred))
probs = pipe1.predict_proba(X_test)
print("log loss: ", log_loss(y_test, probs))
```

```
accuracy:  0.7791164658634538
log loss:  0.5289887343179546
```

```
#change some parameters
pipe1.set_params(tfidf__min_df=1, logreg__C=2.4, logreg__max_iter = 1000).fit(X_train,
pred = pipe1.predict(X_test)
```

```
print("accuracy: ", accuracy_score(y_test, pred))
probs = pipe1.predict_proba(X_test)
print("log loss: ", log_loss(y_test, probs))
```

```
accuracy:  0.8313253012048193
log loss:  0.510320890344046
```

Neural Network Analysis

Using Sklearn NN with customized arguments

```
Classifers: solver='lbfgs', alpha=1e-5, hidden_layer_sizes=(15, 2), random_state=1
accuracy score:  0.7791164658634538
precision score:  0.75
recall score:  0.8181818181818182
f1 score:  0.7826086956521738

Classifers: solver='lbfgs', alpha=1e-5, hidden_layer_sizes=(20, 3), random_state=1
accuracy score:  0.7871485943775101
precision score:  0.7463768115942029
recall score:  0.8512396694214877
f1 score:  0.7953667953667954


Classifers: solver='lbfgs', alpha=1e-5, hidden_layer_sizes=(5, 2), random_state=1
accuracy score:  0.7951807228915663
precision score:  0.8070175438596491
recall score:  0.7603305785123967
f1 score:  0.7829787234042552
```

Analysis

After using the Sklearn Neural Network Model we got various results just by tuning the classifiers slightly in each model. Overall the sklearn Neural Network requires little code. However, I think to get a good model there need to be lot of trial and error. Where tweaking the arugements can caused varying performances. With the three model I tried they all had different results but the NN couldn't compete with the logistic regression results. I think that with more data and fine tuning NN models could have a much higher performace.

```
import pandas as pd

# read in data
```

```
df = pd.read_csv('IMDB_Sentiment.csv')
df = df.dropna() #remove NAN values


# text preprocessing
from nltk.corpus import stopwords
from sklearn.feature_extraction.text import TfidfVectorizer

vectorizer = TfidfVectorizer(binary=True)

#get the X and y values for each entry
X = vectorizer.fit_transform(df.Review)
df['sentiment'] = df['sentiment'].astype(int)
y = df.sentiment

# divide into train and test
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, train_size=(
```

Train and Test data

```
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score, recall_score, f1_score

classifier = MLPClassifier(solver='lbfgs', alpha=1e-5, hidden_layer_sizes=(15, 2), ra

classifier.fit(X_train, y_train)
pred = classifier.predict(X_test)

print('accuracy score: ', accuracy_score(y_test, pred))
print('precision score: ', precision_score(y_test, pred))
print('recall score: ', recall_score(y_test, pred))
print('f1 score: ', f1_score(y_test, pred))
```

```
accuracy score:  0.7791164658634538
precision score:  0.75
recall score:  0.8181818181818182
f1 score:  0.7826086956521738
```

```
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score, recall_score, f1_score

classifier = MLPClassifier(solver='lbfgs', alpha=1e-5, hidden_layer_sizes=(20, 3), ra

classifier.fit(X_train, y_train)
pred = classifier.predict(X_test)
```

```
print('accuracy score: ', accuracy_score(y_test, pred))
print('precision score: ', precision_score(y_test, pred))
print('recall score: ', recall_score(y_test, pred))
print('f1 score: ', f1_score(y_test, pred))
```

```
accuracy score:  0.7871485943775101
precision score:  0.7463768115942029
recall score:  0.8512396694214877
f1 score:  0.7953667953667954
```

```
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score, recall_score, f1_score

classifier = MLPClassifier(solver='lbfgs', alpha=1e-5, hidden_layer_sizes=(5, 2), ranc

classifier.fit(X_train, y_train)
pred = classifier.predict(X_test)

print('accuracy score: ', accuracy_score(y_test, pred))
print('precision score: ', precision_score(y_test, pred))
print('recall score: ', recall_score(y_test, pred))
print('f1 score: ', f1_score(y_test, pred))
```

```
accuracy score:  0.7951807228915663
precision score:  0.8070175438596491
recall score:  0.7603305785123967
f1 score:  0.7829787234042552
```