```
# Import Libraries
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
import nltk
nltk.download('stopwords')
from nltk.corpus import stopwords
# NB
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.naive_bayes import BernoulliNB
# LR
from sklearn.pipeline import Pipeline
from sklearn.linear_model import LogisticRegression
# NN
from sklearn.neural_network import MLPClassifier
```

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

The Data Set

The dataset is meant for sentiment analysis, which may be helpful for gathering general opinions about business success.
There are 1000 instances in this dataset, the feature is a text review.
This specific dataset is scraped from Yelp, so the reviews are pertain to restaurants.
The possible classes that the following models try to predict are categorical, given by a number representing the overall sentiment of the review:

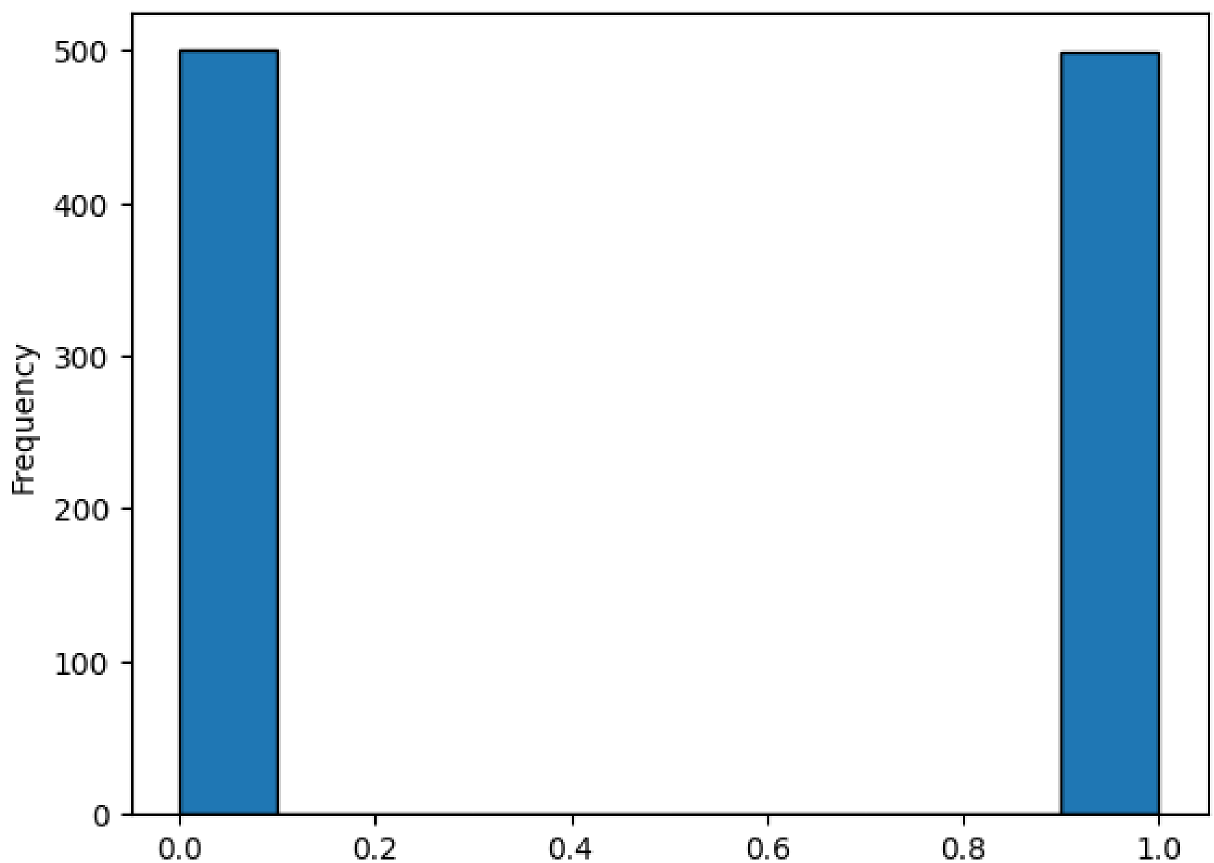
0. Negative
1. Positive

```
# load dataset from google drive
url = 'https://drive.google.com/file/d/1CQe3CpBdRQFdmhTwq3GaflQ-PpGQrBT9/view?usp=sharing'
path = 'https://drive.google.com/uc?export=download&id='+url.split('/')[-2]

df = pd.read_csv(path, sep="\\")
df.columns=['review', 'sentiment']

# display distribution of target class values
df['sentiment'].plot(kind='hist', edgecolor='black')
```

```
<Axes: ylabel='Frequency'>
```



```
# Naive Bayes Model

# split the data into training and testing sets 80/20
X = df.review
y = df.sentiment
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,  train_size=0.8, random_state=1234)

print('--------Multinomial NB--------')
# fit X
my_stopwords = set(stopwords.words('english'))
my_stopwords= list(my_stopwords)  # vectorizer accepts lists only
vectorizer = TfidfVectorizer(stop_words=my_stopwords)
X_train = vectorizer.fit_transform(X_train)
X_test = vectorizer.transform(X_test)

# train NB classifier
naive_bayes = MultinomialNB()
naive_bayes.fit(X_train, y_train)

# test NB classifier
pred = naive_bayes.predict(X_test)

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

# print model evaluation
print(classification_report(y_test, pred))

# print wrong classifications
y_test[y_test != pred]
for i in [99, 738, 29, 914, 361, 241]:
  print(df.loc[i]['review'])
  print(df.loc[i]['sentiment'])

# try making a BinomialNB instead
```

```
print('--------Binomial NB--------')
naive_bayes2 = BernoulliNB()
naive_bayes2.fit(X_train, y_train)
pred = naive_bayes2.predict(X_test)
print(confusion_matrix(y_test, pred))
print(classification_report(y_test, pred))
```

```
--------Multinomial NB--------
[[73 27]
 [18 82]]
              precision    recall  f1-score   support

           0       0.80      0.73      0.76       100
           1       0.75      0.82      0.78       100

    accuracy                           0.78       200
   macro avg       0.78      0.77      0.77       200
weighted avg       0.78      0.78      0.77       200

Our server was fantastic and when he found out the wife loves roasted garlic and bone marrow, he added extra
1
Never had anything to complain about here.
1
Also there are combos like a burger, fries, and beer for 23 which is a decent deal.
1
The only thing I wasn't too crazy about was their guacamole as I don't like it puréed.
0
Pretty cool I would say.
1
By this time our side of the restaurant was almost empty so there was no excuse.
0
--------Binomial NB--------
[[71 29]
 [21 79]]
              precision    recall  f1-score   support

           0       0.77      0.71      0.74       100
           1       0.73      0.79      0.76       100
```

```
    accuracy                           0.75       200
   macro avg       0.75      0.75      0.75       200
weighted avg       0.75      0.75      0.75       200
```

Naive Bayes Evaluation

In both multinominal and binomial approaches, the model seems to be better than random guessing. The distribution of the target class was 0.5 and 0.5. The accuracy of both these NB models are about 0.76. Binomial is different from multinomial because it notes the presence of a word rather than counting words. This was interesting to me because I thought the presence or absence of words like 'delicious', 'bad' could be strong indicators of positive or negative sentiment. I attribute this unexpected performance difference to the diverse vocabulary used by review-writers.

```
# Logistic Regression
pipe1 = Pipeline([
        ('tfidf', TfidfVectorizer()),
        ('logreg', LogisticRegression(multi_class='ovr', solver='newton-cholesky',class_weight='balanced')),
])
pipe2 = Pipeline([
        ('tfidf', TfidfVectorizer()),
        ('logreg', LogisticRegression(multi_class='ovr', solver='liblinear',class_weight='balanced')),
])

# Split data
training_data = df.sample(frac=0.8, random_state=24)
testing_data = df.drop(training_data.index)

# Create model
pipe1.fit(training_data.review, training_data.sentiment)
pipe2.fit(training_data.review, training_data.sentiment)
```

```
# Evaluate model
pred = pipe1.predict(testing_data.review)
print('--------NEWTON-CHOLESKY--------')
print(confusion_matrix(testing_data.sentiment, pred))
print(classification_report(testing_data.sentiment, pred))

pred = pipe2.predict(testing_data.review)
print('--------LIBLINEAR--------')
print(confusion_matrix(testing_data.sentiment, pred))
print(classification_report(testing_data.sentiment, pred))
```

```
--------NEWTON-CHOLESKY--------
[[87 12]
 [26 75]]
              precision    recall  f1-score   support

           0       0.77      0.88      0.82        99
           1       0.86      0.74      0.80       101

    accuracy                           0.81       200
   macro avg       0.82      0.81      0.81       200
weighted avg       0.82      0.81      0.81       200

--------LIBLINEAR--------
[[87 12]
 [26 75]]
              precision    recall  f1-score   support

           0       0.77      0.88      0.82        99
           1       0.86      0.74      0.80       101

    accuracy                           0.81       200
   macro avg       0.82      0.81      0.81       200
```

```
weighted avg       0.82      0.81      0.81       200
```

Logistic Regression Evaluation

Right away, I can see that logistic regression outperformed Naive Bayes for this data set. This makes sense because logistic regression is designed specifically for binary classification while NB considers each feature independently. However, language often requires context.

I had two main other considerations when I made my model. First is that my classification is binary. A sample is either 'positive' (1) or 'negative' (0). So, I specified the multi_class mode as 'ovr'. Second,I chose the solver mode to be newton-cholesky because there were many more samples (1000) vs features (20 average words per review).

Just to be sure I also tried a liblinear AND lbfgs approach and got same exact results.

Double-click (or enter) to edit

```
# Neural Network
vectorizer = TfidfVectorizer(stop_words=my_stopwords, binary=True)

# Identify X and y
X = vectorizer.fit_transform(df.review)
y = df.sentiment

# Split train/test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, train_size=0.8, random_state=1234)

# Create model
classifier = MLPClassifier(solver='lbfgs', alpha=1e-5, hidden_layer_sizes=(10,2), random_state=1)
classifier.fit(X_train, y_train)

# Test model
pred = classifier.predict(X_test)
```

```
# Evaluate model
print(confusion_matrix(y_test, pred))
print(classification_report(y_test, pred))
```

```
[[75 25]
 [17 83]]
              precision    recall  f1-score   support

           0       0.82      0.75      0.78       100
           1       0.77      0.83      0.80       100

    accuracy                           0.79       200
   macro avg       0.79      0.79      0.79       200
weighted avg       0.79      0.79      0.79       200
```

Neural Network Evaluation

I am surprised to see that the neural network performs at about the same level as the Naive-Bayes. I even tried different variations of hyperparameters (alpha, hidden layer sizes, solver.) This would lead me to believe that the logistic regression model was the best for this dataset.
However, when I looked it up. Neural networks (theoretically) outperform logistic regression. In reality, neural networks are difficult to train and tend to overfit. Below I have repeated the procedure using the training data and as suspected the accuracy was very very high. Overfitting means that it is difficult to generalize the model to data outside of training.

```
# Neural Network 2.0
vectorizer = TfidfVectorizer(stop_words=my_stopwords, binary=True)

# Identify X and y
X = vectorizer.fit_transform(df.review)
y = df.sentiment

# Split train/test
```

```
# Split train/test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, train_size=0.8, random_state=1234)

# Create model
classifier = MLPClassifier(solver='lbfgs', alpha=1e-5, hidden_layer_sizes=(10,2), random_state=1)
classifier.fit(X_train, y_train)

# Test model
pred = classifier.predict(X_train)

# Evaluate model
print(confusion_matrix(y_train, pred))
print(classification_report(y_train, pred))
```

```
[[396   4]
 [  2 397]]
              precision    recall  f1-score   support

           0       0.99      0.99      0.99       400
           1       0.99      0.99      0.99       399

    accuracy                           0.99       799
   macro avg       0.99      0.99      0.99       799
weighted avg       0.99      0.99      0.99       799
```