Jonathan Ho

CS 4395

Text Classification

Import Packages

In [128…

```
import pandas as pd
import plotly.express as px
import seaborn as sb

# Text Preprocessing
import string
string.punctuation
import re
from nltk.corpus import stopwords
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split
stopwords = set(stopwords.words('english'))

# Evaluating Data
from sklearn.naive_bayes import BernoulliNB
from sklearn.linear_model import LogisticRegression
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, c
vectorizer = TfidfVectorizer(binary=True)
```

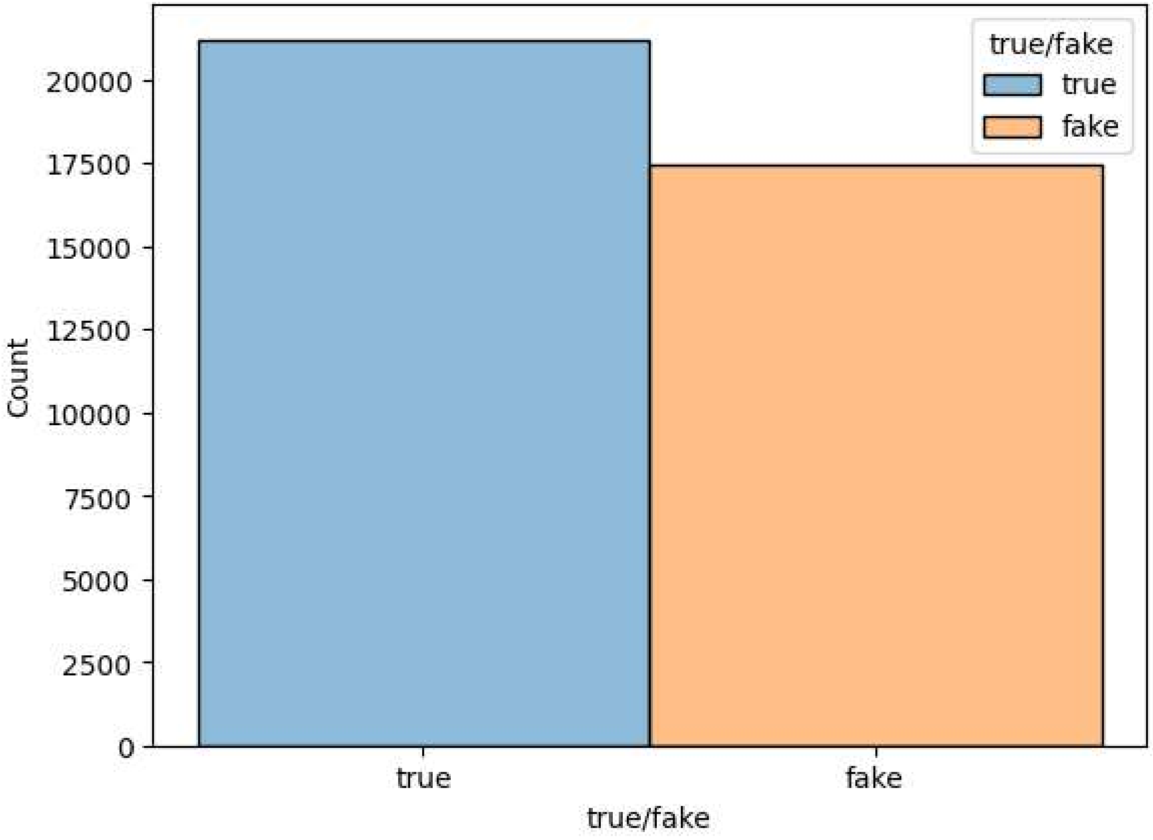
Create dataset and graph

In [129…

```
# Read in fake and true data
fake_df = pd.read_csv('fake_news_data/Fake.csv', header=0, usecols=[1], encoding='utf-
fake_df['true/fake'] = 'fake'

true_df = pd.read_csv('fake_news_data/True.csv', header=0, usecols=[1], encoding='utf-
true_df['true/fake'] = 'true'

# Merge datasets
data_set = pd.concat([true_df, fake_df], ignore_index = True)

# Get rid of duplicate entries
data_set.duplicated().sum()
data_set.drop_duplicates(inplace=True)

# Plot the data
sb.histplot(x="true/fake", hue="true/fake", data=data_set)
```

Out[129]:

```
<Axes: xlabel='true/fake', ylabel='Count'>
```



Text Pre-processing

In [130…

```
# Remove punctuation
def no_punct(t):
  no_p = "".join([i for i in t if i not in string.punctuation])
  return no_p

# Tokenize
def tokenize(t):
  tokens = re.split('\s+', ''.join(t))
  return tokens

# Remove stop words
def no_stopwords(t):
  out = [i for i in t if i not in stopwords]
  return out

# Remove punctuation, lowercase, tokenize, and remove stop words
data_set[['text']] = data_set[['text']].applymap(lambda x:no_punct(x))
data_set[['text']] = data_set[['text']].applymap(lambda y:y.lower())
data_set[['text']] = data_set[['text']].applymap(lambda w:tokenize(w))
data_set[['text']] = data_set[['text']].applymap(lambda z:no_stopwords(z))

# Convert from tokenized list to string of words
data_set['text'] = data_set['text'].astype(str)
data = data_set.iloc[:, [1, 0]]
data_set['text'] = data_set['text'].map(lambda s:' '.join(re.findall('\w+', s)))
print(data_set.head())
```

```
                                                text true/fake
0  washington reuters head conservative republica...      true
1  washington reuters transgender people allowed ...      true
2  washington reuters special counsel investigati...      true
3  washington reuters trump campaign adviser geor...      true
4  seattlewashington reuters president donald tru...      true
```

Creating Train and Test datasets

In [131…

```
X = data_set['text']
y = data_set['true/fake']

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, train_size=0.

# Apply tf-idf vectorizer
X_train = vectorizer.fit_transform(X_train)
X_test = vectorizer.transform(X_test)

print('train size:', X_train.shape)
print('\ntest size:', X_test.shape)
```

```
train size: (30917, 196046)

test size: (7730, 196046)
```

Naive Bayes

In [132…

```
# Use the Bernoulil function for NB and make a prediction
nb = BernoulliNB()
nb.fit(X_train, y_train)
pred = nb.predict(X_test)

# Evaluate the prediction
print('accuracy score: ', accuracy_score(y_test, pred))
print('precision score: ', precision_score(y_test, pred, pos_label="true"))
print('recall score: ', recall_score(y_test, pred, pos_label="true"))
print('f1 score: ', f1_score(y_test, pred, pos_label="true"))
print(confusion_matrix(y_test, pred))
```

```
accuracy score:  0.9677878395860284
precision score:  0.9474880654694249
recall score:  0.9956999522216914
f1 score:  0.970995923121724
[[3313  231]
 [  18 4168]]
```

Logistic Regression

In [133…

```
# Train the log-reg model
classifier = LogisticRegression(solver='lbfgs', class_weight='balanced')
classifier.fit(X_train, y_train)
pred = classifier.predict(X_test)

# Evaluate the model
print('accuracy score: ', accuracy_score(y_test, pred))
print('precision score: ', precision_score(y_test, pred, pos_label="true"))
print('recall score: ', recall_score(y_test, pred, pos_label="true"))
print('f1 score: ', f1_score(y_test, pred, pos_label="true"))
probs = classifier.predict_proba(X_test)
print('log loss: ', log_loss(y_test, probs))
```

```
accuracy score:  0.9910737386804657
precision score:  0.9876806443970623
recall score:  0.9959388437649307
f1 score:  0.9917925538241941
log loss:  0.07616759222479581
```

Neural Networks

In [ ]:

```
# Train the neural network model
nn_class = MLPClassifier(solver='lbfgs', alpha=1e-5,
                   hidden_layer_sizes=(8,), random_state=1)
nn_class.fit(X_train, y_train)
pred = nn_class.predict(X_test)

# Evaluate the model
print('accuracy score: ', accuracy_score(y_test, pred))
print('precision score: ', precision_score(y_test, pred, pos_label="true"))
print('recall score: ', recall_score(y_test, pred, pos_label="true"))
print('f1 score: ', f1_score(y_test, pred, pos_label="true"))
```

Analysis

Comparing the results of the three different ways to create a model, it seems that neural networks would be the best. Overall, all models output a high scores.

Naive Bayes may have had the relativelly lowest score since it had a good amount of false positives, but it seems it had not affected the results drastically. Using BernoulliNB over MultinomialNB did improve results since there are only two classes, but it still did not do better than Logistic Regression nor Neural Networks.

Logistic Regression worked very well, even resulting in a very small log loss, most likely due to the prediction determining whether news was "true" or "fake". Assuming it is not overfitting, the scores seem to signify that the given data can have a decision boundary that clearly separates the data.

There is a chance that the neural networks is overfitting as I only used 1 hidden layer with 8 hidden nodes within it, but I am not sure. I mainly chose to do one layer since there are only two inputs, so it led me to believe that more hidden layers would overfit. This seems to be supported by two hidden layers (such as (8, 2)) significantly decreases the accuracy score (~0.50-0.60).