In-Class Activity Topic 5 - Supervised Document Classification

DATA 5420/6420

Name: Dallin Moore

In this in-class exercise we will train several supervised classifiers to predict the genre label on the Brown corpus. We will also utilize grid searching to help determine optimal parameters for a final model. Once we've decided on a final model, we will apply it to a toy corpus of short texts and predict their genre.

We begin as always, by loading in our dependencies:

```
# Import necessary libraries
import nltk
import re
from nltk.corpus import brown
from nltk.stem import WordNetLemmatizer
from nltk.corpus import stopwords
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
import pandas as pd

from sklearn.naive_bayes import MultinomialNB
from sklearn.svm import LinearSVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score

from sklearn.metrics import confusion_matrix, classification_report
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import cross_val_score

import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix

# Download the nltk features
nltk.download('wordnet')
nltk.download('stopwords')
nltk.download('punkt')
```

```
[nltk_data] Downloading package wordnet to /root/nltk_data...
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.
True
```

```
df = pd.read_csv('/content/Cleaned_Brown.csv')
df.head()
```

	Unnamed: 0	doc_para_id	label	combined_tokenized_text	clean_text
0	0	ca01_0	news	The Fulton County Grand Jury said Friday an in...	fulton county grand jury said friday investiga...
1	1	ca01_1	news	The jury further said in term-end presentments...	jury said termend presentment city executive c...

urged next legislature

Peform any necessary cleaning and text normalization

```
# Text normalization function
def normalize_text(text):
    text = text.lower()
    text = re.sub(r'[^a-zA-Z\s]', '', text)
    words = nltk.word_tokenize(text)
    words = [word for word in words if word not in stopwords.words('english')]
    lemmatizer = WordNetLemmatizer()
    words = [lemmatizer.lemmatize(word) for word in words]
    return ' '.join(words)

# Apply text normalization to the 'text' column
df['clean_text'] =


#df.to_csv('Cleaned_Brown.csv')
# df = pd.read_csv('')
```

Let's examine the breakdown of the genre labels...

What are the most common genres in this dataset, which are the least? What impact might this have on the predictive accuracy of the models we train?

```
# check for missing data, and drop out observations as necessary
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 15667 entries, 0 to 15666
Data columns (total 5 columns):
 #   Column                   Non-Null Count  Dtype
---  ------                   --------------  -----
 0   Unnamed: 0               15667 non-null  int64
 1   doc_para_id              15667 non-null  object
 2   label                    15667 non-null  object
 3   combined_tokenized_text  15667 non-null  object
 4   clean_text               15622 non-null  object
dtypes: int64(1), object(4)
memory usage: 612.1+ KB
```

```
df.dropna(inplace=True)
```

```
# examine labels
df['label'].value_counts()
```

```
news               2234
learned            1418
belles_lettres     1405
adventure          1387
romance            1253
lore               1203
mystery            1164
hobbies            1119
fiction            1043
editorial          1003
government          851
reviews             629
religion            369
science_fiction     335
humor               254
Name: label, dtype: int64
```

There is not even distribution between the different genres. The model will be trained on many more articles in the 'news' genre compared to the 'humor' genre. It may be better at classifying news because it has more data to examine.

Split the data into a training and test set

```
# Split data into text and labels
texts = df['clean_text'].tolist()
labels = df['label'].tolist()

# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(texts, labels, test_size=.2, random_state=42)
```

Peform feature engineering by vectorizing the text with TF-IDF

Let's set `min_df` and `max_df` to limit the number of features, then we'll check to see how many features we have in our dataset:

```
# Create a TF-IDF vectorizer
vectorizer = TfidfVectorizer(stop_words = 'english', min_df=1, max_df=0.8)
X_train_vec = vectorizer.fit_transform(X_train)
X_test_vec = vectorizer.transform(X_test) # we need to have the same number of features in the test set as the training set (even if it's ze

num_features_train = X_train_vec.shape[1]
num_features_test = X_test_vec.shape[1]
print(num_features_test,num_features_train)
```

```
 36120 36120
```

Now let's train a few supervised classifiers using our training set

```
models = {
    'Naive Bayes': MultinomialNB(),
    'Linear SVM': LinearSVC(random_state=42),
    'Random Forests': RandomForestClassifier(random_state=42)
}

# Perform cross-validation for each model
for name, model in models.items():
    scores = cross_val_score(model,X_train_vec,y_train,cv=5)
    print(f"{name} Cross-Validation Mean Accuracy: {scores.mean():.4f}")
```

```
 Naive Bayes Cross-Validation Mean Accuracy: 0.4688
 Linear SVM Cross-Validation Mean Accuracy: 0.6868
 Random Forests Cross-Validation Mean Accuracy: 0.4780
```

Which model appears to be performing best?

Linear SVM has the highest score and preformed the best, while naive bayes and random forest were about the same in accuracy and not far above the baseline.

Finally, let's select the best performing model and perform a CV grid search to select optimal parameters

```
param_grid_svc = {
    'C': [0.001, 0.01, 0.1, 1, 10],
    'loss': ['hinge', 'squared_hinge'],
    'max_iter': [1000, 5000, 10000]
}

# Grid search for LinearSVC
grid_search_svc = GridSearchCV(LinearSVC(random_state=42), param_grid_svc, cv=5, verbose=10)
grid_search_svc.fit(X_train_vec, y_train)

# After fitting, you would typically print the best parameters as follows:
print("Best parameters for LinearSVC:", grid_search_svc.best_params_)
```

```
[CV 2/5; 26/30] START C=10, loss=hinge, max_iter=5000.....................
[CV 2/5; 26/30] END C=10, loss=hinge, max_iter=5000;, score=0.646 total time=   3.3s
[CV 3/5; 26/30] START C=10, loss=hinge, max_iter=5000.....................
[CV 3/5; 26/30] END C=10, loss=hinge, max_iter=5000;, score=0.643 total time=   4.4s
[CV 4/5; 26/30] START C=10, loss=hinge, max_iter=5000.....................
[CV 4/5; 26/30] END C=10, loss=hinge, max_iter=5000;, score=0.656 total time=   3.5s
[CV 5/5; 26/30] START C=10, loss=hinge, max_iter=5000.....................
[CV 5/5; 26/30] END C=10, loss=hinge, max_iter=5000;, score=0.646 total time=   4.2s
[CV 1/5; 27/30] START C=10, loss=hinge, max_iter=10000....................
[CV 1/5; 27/30] END C=10, loss=hinge, max_iter=10000;, score=0.645 total time=   4.4s
[CV 2/5; 27/30] START C=10, loss=hinge, max_iter=10000....................
[CV 2/5; 27/30] END C=10, loss=hinge, max_iter=10000;, score=0.646 total time=   3.2s
[CV 3/5; 27/30] START C=10, loss=hinge, max_iter=10000....................
[CV 3/5; 27/30] END C=10, loss=hinge, max_iter=10000;, score=0.643 total time=   3.5s
[CV 4/5; 27/30] START C=10, loss=hinge, max_iter=10000....................
[CV 4/5; 27/30] END C=10, loss=hinge, max_iter=10000;, score=0.656 total time=   4.0s
[CV 5/5; 27/30] START C=10, loss=hinge, max_iter=10000....................
[CV 5/5; 27/30] END C=10, loss=hinge, max_iter=10000;, score=0.646 total time=   4.4s
[CV 1/5; 28/30] START C=10, loss=squared_hinge, max_iter=1000..............
[CV 1/5; 28/30] END C=10, loss=squared_hinge, max_iter=1000;, score=0.649 total time=   2.9s
[CV 2/5; 28/30] START C=10, loss=squared_hinge, max_iter=1000..............
[CV 2/5; 28/30] END C=10, loss=squared_hinge, max_iter=1000;, score=0.653 total time=   2.6s
[CV 3/5; 28/30] START C=10, loss=squared_hinge, max_iter=1000..............
[CV 3/5; 28/30] END C=10, loss=squared_hinge, max_iter=1000;, score=0.650 total time=   3.4s
[CV 4/5; 28/30] START C=10, loss=squared_hinge, max_iter=1000..............
[CV 4/5; 28/30] END C=10, loss=squared_hinge, max_iter=1000;, score=0.667 total time=   2.7s
[CV 5/5; 28/30] START C=10, loss=squared_hinge, max_iter=1000..............
[CV 5/5; 28/30] END C=10, loss=squared_hinge, max_iter=1000;, score=0.653 total time=   2.7s
[CV 1/5; 29/30] START C=10, loss=squared_hinge, max_iter=5000..............
[CV 1/5; 29/30] END C=10, loss=squared_hinge, max_iter=5000;, score=0.649 total time=   2.8s
[CV 2/5; 29/30] START C=10, loss=squared_hinge, max_iter=5000..............
[CV 2/5; 29/30] END C=10, loss=squared_hinge, max_iter=5000;, score=0.653 total time=   3.0s
[CV 3/5; 29/30] START C=10, loss=squared_hinge, max_iter=5000..............
[CV 3/5; 29/30] END C=10, loss=squared_hinge, max_iter=5000;, score=0.650 total time=   3.1s
[CV 4/5; 29/30] START C=10, loss=squared_hinge, max_iter=5000..............
[CV 4/5; 29/30] END C=10, loss=squared_hinge, max_iter=5000;, score=0.667 total time=   2.7s
[CV 5/5; 29/30] START C=10, loss=squared_hinge, max_iter=5000..............
[CV 5/5; 29/30] END C=10, loss=squared_hinge, max_iter=5000;, score=0.653 total time=   2.7s
[CV 1/5; 30/30] START C=10, loss=squared_hinge, max_iter=10000.............
[CV 1/5; 30/30] END C=10, loss=squared_hinge, max_iter=10000;, score=0.649 total time=   2.8s
[CV 2/5; 30/30] START C=10, loss=squared_hinge, max_iter=10000.............
[CV 2/5; 30/30] END C=10, loss=squared_hinge, max_iter=10000;, score=0.653 total time=   3.5s
[CV 3/5; 30/30] START C=10, loss=squared_hinge, max_iter=10000.............
[CV 3/5; 30/30] END C=10, loss=squared_hinge, max_iter=10000;, score=0.650 total time=   2.6s
[CV 4/5; 30/30] START C=10, loss=squared_hinge, max_iter=10000.............
[CV 4/5; 30/30] END C=10, loss=squared_hinge, max_iter=10000;, score=0.667 total time=   2.7s
[CV 5/5; 30/30] START C=10, loss=squared_hinge, max_iter=10000.............
[CV 5/5; 30/30] END C=10, loss=squared_hinge, max_iter=10000;, score=0.653 total time=   2.7s
Best parameters for LinearSVC: {'C': 1, 'loss': 'squared_hinge', 'max_iter': 1000}
```

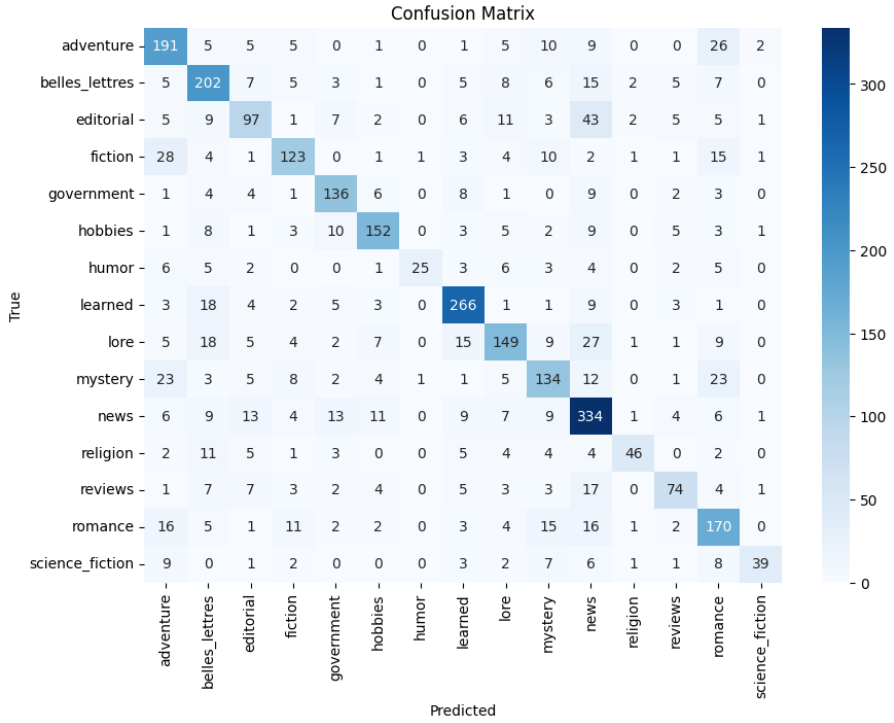
Now apply the final model to the test set, and produce a confusion matrix and classification report:

```
y_pred = grid_search_svc.predict(X_test_vec)


# Compute confusion matrix
cm = confusion_matrix(y_test, y_pred)
labels = sorted(list(set(y_test)))

# Plot the heatmap
plt.figure(figsize=(10, 7))
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", xticklabels=labels, yticklabels=labels)
plt.xlabel('Predicted')
plt.ylabel('True')
plt.title('Confusion Matrix')
plt.show()

print("\nClassification Report:")
print(classification_report(y_test, y_pred))
```




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

     adventure       0.63      0.73      0.68       260
 belles_lettres      0.66      0.75      0.70       271
     editorial       0.61      0.49      0.55       197
       fiction       0.71      0.63      0.67       195
    government       0.74      0.78      0.76       175
       hobbies       0.78      0.75      0.76       203
         humor       0.93      0.40      0.56        62
       learned       0.79      0.84      0.82       316
          lore       0.69      0.59      0.64       252
       mystery       0.62      0.60      0.61       222
          news       0.65      0.78      0.71       427
      religion       0.84      0.53      0.65        87
       reviews       0.70      0.56      0.62       131
       romance       0.59      0.69      0.64       248
science_fiction       0.85      0.49      0.62        79

      accuracy                           0.68      3125
     macro avg       0.72      0.64      0.67      3125
  weighted avg       0.69      0.68      0.68      3125
```

Which categories were commonly confused, and is that surprising?

The most common was editorial was predicted to be news. This is not surprising as the categories are very similar.

The next was fiction being predicted as adventure. Once again these are similar as adventure is often also fiction.

Next is adventure predicted as romance. This one is more surprising, but if we looked at what the model was using to predict the genre as romance (vs adventure) it may provide insights.

Which metric(s) might we choose to evaluate our model's performance based on for this dataset and why?

It doesn't seem like false negatives or false positives are more important than the other, so F-measure can be used to evaluate performance. (Also, the distribution is not even so accuracy should not be used)

As a final step, I've provide three different new texts I pulled from various sources, let's see how our model works in predicting labels on these unseen texts...

```
# Toy corpus
toy_corpus = [
    "On Tuesday, the Organisation for Economic Cooperation and Development (OECD) published its latest outlook for the global economy. Descri
    "His mother’s roses were in full bloom, red and pink and riotous, just the way they all liked them, and as Nicholas drew close, he felt th
    "Ah sinful nation, a people laden with iniquity, a seed of evildoers, children that are corrupters: they have forsaken the LORD, they have
]

# Vectorize and predict using the best model
toy_corpus_vec = vectorizer.transform((toy_corpus))
num_features = toy_corpus_vec.shape[1]
num_features
```

```
36120
```

Why is it important that we use the same number of features here as we did for the training corpus?

so that there are columns that have value of zero that will provide information on what the text is about

```
predicted_labels = grid_search_svc.predict(toy_corpus_vec)

for text, label in zip(toy_corpus, predicted_labels):
    print(f"'{text}' is predicted as '{label}' category.")
```

```
'On Tuesday, the Organisation for Economic Cooperation and Development (OECD) published its latest outlook for the global economy. Descr
'His mother’s roses were in full bloom, red and pink and riotous, just the way they all liked them, and as Nicholas drew close, he felt
'Ah sinful nation, a people laden with iniquity, a seed of evildoers, children that are corrupters: they have forsaken the LORD, they ha
```

Did the model seem to apply the correct label for the toy corpus?

For the first 2 definitely, but the third one doesn't quite seem right. We likely wanted religion to be the genre identified, but adventure may be that far off if the religion articles were more than just scripture.