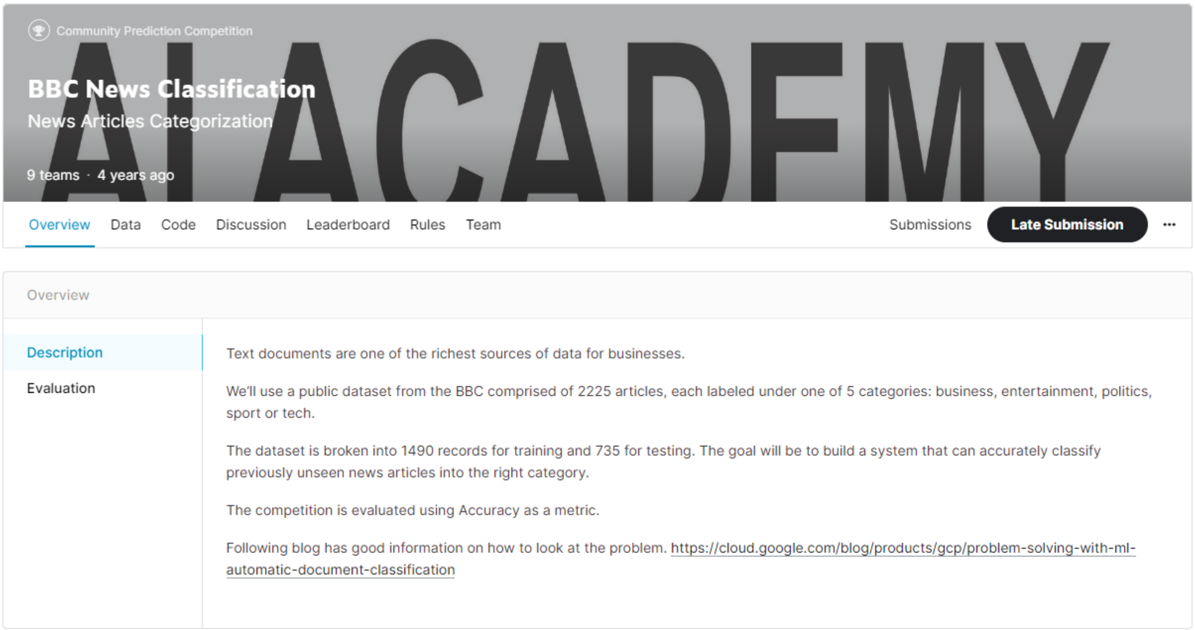
BBC News Articles Classification Project



BBC News Classification

News Articles Categorization

9 teams · 4 years ago

Overview

Text documents are one of the richest sources of data for businesses.

We'll use a public dataset from the BBC comprised of 2225 articles, each labeled under one of 5 categories: business, entertainment, politics, sport or tech.

The dataset is broken into 1490 records for training and 735 for testing. The goal will be to build a system that can accurately classify previously unseen news articles into the right category.

The competition is evaluated using Accuracy as a metric.

Following blog has good information on how to look at the problem. https://cloud.google.com/blog/products/gcp/problem-solving-with-ml-automatic-document-classification

1. Peoject Description

Dataset Background:

1. A public dataset from the BBC comprised of 2225 articles with label of 5 categories: business, entertainment, politics, sport or tech.
2. The dataset is broken into 1490 records for training and 735 for testing.

Project Goal:

1. Build a model using the matrix factorization method to predict the test data labels and measure the performances of the model.
2. Build a model using the supervised learning method to predict the test data labels and measure the performances of the model.
3. Compare the result from both two models.

2. Exploratory Data Analysis

First, we will complete the data cleaning, checking for null values and duplicate values.

Second, we will examine the distribution of the dependent variable and the statistical data of the text.

Third, we will perform a TF-IDF transformation on the text data to give it data features for subsequent predictive analysis.

In [1]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import itertools

from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.decomposition import NMF
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import cross_val_score, train_test_split, GridSearchCV
from sklearn.metrics import accuracy_score, confusion_matrix
from collections import Counter

import warnings
```

2.1 Loading Data

In [2]:

```
train = pd.read_csv('BBC News Train.csv')
test = pd.read_csv('BBC News Test.csv')
```

In [3]:

```
train.shape
```

Out[3]:

```
(1490, 3)
```

In [4]:

```
train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1490 entries, 0 to 1489
Data columns (total 3 columns):
 #   Column     Non-Null Count  Dtype 
---  ------     --------------  ----- 
 0   ArticleId  1490 non-null   int64 
 1   Text       1490 non-null   object
 2   Category   1490 non-null   object
dtypes: int64(1), object(2)
memory usage: 35.0+ KB
```

In [5]:

```
train.head(10)
```

Out[5]:

	ArticleId	Text	Category
0	1833	worldcom ex-boss launches defence lawyers defe...	business
1	154	german business confidence slides german busin...	business
2	1101	bbc poll indicates economic gloom citizens in ...	business
3	1976	lifestyle governs mobile choice faster bett...	tech
4	917	enron bosses in $168m payout eighteen former e...	business
5	1582	howard truanted to play snooker conservative...	politics
6	651	wales silent on grand slam talk rhys williams ...	sport
7	1797	french honour for director parker british film...	entertainment
8	2034	car giant hit by mercedes slump a slump in pro...	business
9	1866	fockers fuel festive film chart comedy meet th...	entertainment

We are going to use 'Text' to predict 'Category'

2.2 Checking Missing Value

In [6]:

```
train.isnull().sum()
```

Out[6]:

```
ArticleId    0
Text         0
Category     0
dtype: int64
```

There is no missing value.

2.3 Checking Duplicates

In [7]:

```
duplicates = train.duplicated(subset=["Text"]).sum()
duplicates
```

Out[7]:

```
50
```

In [8]:

```
train.drop_duplicates(subset=["Text"], inplace=True)
train.shape
```

Out[8]:

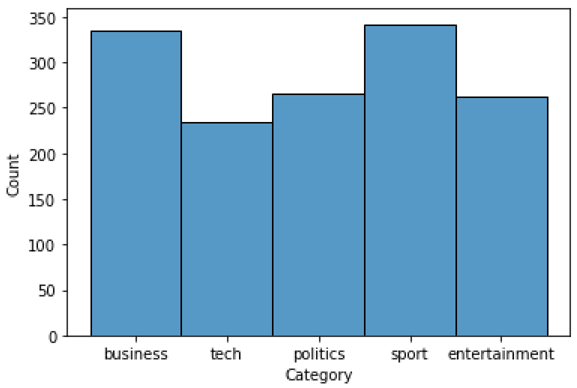
```
(1440, 3)
```

There are 50 'Text' duplicates removed from train set.

2.4 Checking Distribution

In [9]:
```
sns.histplot(train["Category"], bins=20)
```

Out[9]:
```
<AxesSubplot:xlabel='Category', ylabel='Count'>
```



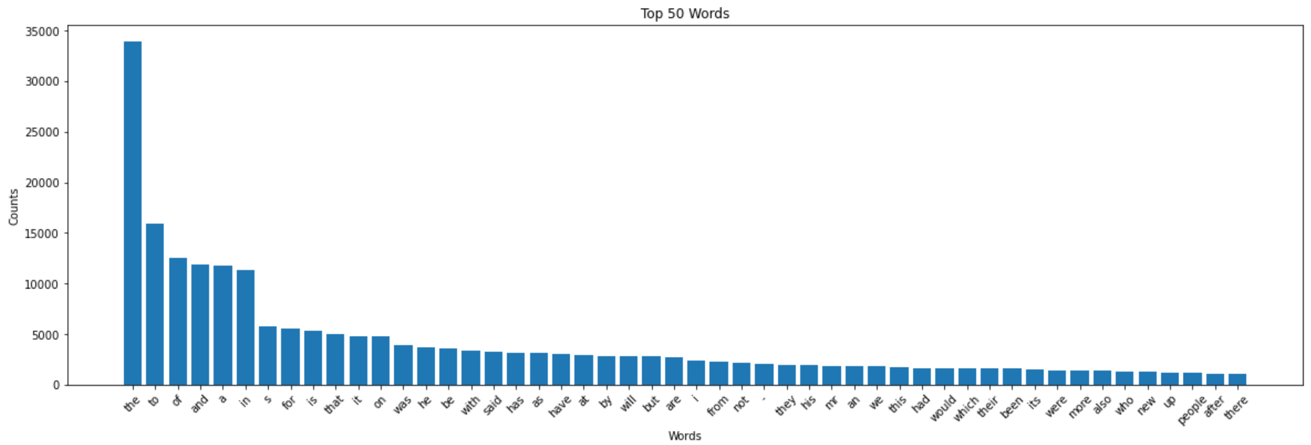
The difference in the number of categories is not significant, so we will keep all of them.

2.5 Top Word Distribution

Total Word

In [10]:
```
# Count Words
all_text = train['Text'].str.cat(sep=' ')
words = all_text.lower().split()
total_words = len(words)
print('Total World:', total_words)
word_counts = Counter(words)
top_words = word_counts.most_common(50)
words = [word[0] for word in top_words]
counts = [word[1] for word in top_words]
# Bar Chart
plt.figure(figsize=(20, 6))
plt.bar(words, counts)
plt.xticks(rotation=45)
plt.title('Top 50 Words')
plt.xlabel('Words')
plt.ylabel('Counts')
plt.show()
```

```
Total World: 554711
```



Total Unique Word

In [11]:

```
unique_words = set()
categories = train['Category'].unique()

for cat in categories:
    text = train[train['Category'] == cat]['Text'].str.cat(sep=' ')
    words = text.lower().split()
    unique_words.update(words)

unique_words_counts = len(unique_words)
print("Total unique words: {}".format(unique_words_counts))
```

```
Total unique words: 35594
```

2.6 TF-IDF Transformation

In [12]:

```
tf = TfidfVectorizer(sublinear_tf=True,
                            max_df=0.8, #try 0.5, 0.7, 0.9
                            min_df=8, # try 6, 8, 10
                            stop_words="english")
tf.fit(train["Text"])
tf_train = tf.transform(train["Text"])
tf_test = tf.transform(test["Text"])
print(tf_train.shape)
print(tf_test.shape)
```

```
(1440, 4878)
(735, 4878)
```

Using a TF-IDF vectorizer to transform text data into numerical features.

3. Building Supervised Model

Logistic Regression Model

100% Lables:

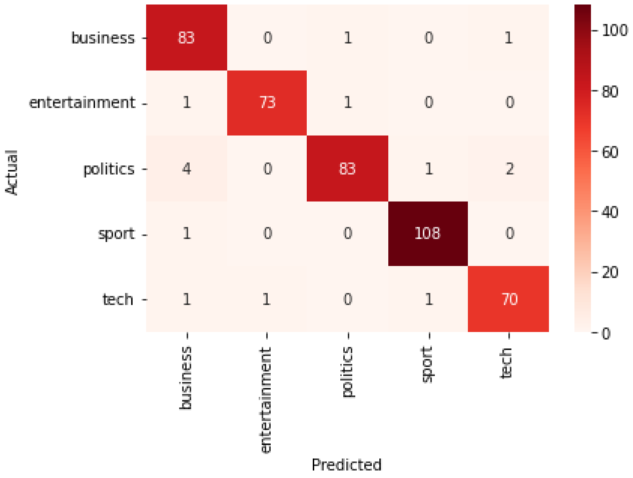
In [13]:

```
X_train, X_valid, y_train, y_valid = train_test_split(tf_train, train['Category'], test_size=0.3, random_state=26)
```

In [14]:

```
lr = LogisticRegression(random_state=26)
lr.fit(X_train, y_train)
y_pred = lr.predict(X_valid)
acc = accuracy_score(y_valid, y_pred)
print("Validation Accuracy:", acc)
cm = confusion_matrix(y_valid, y_pred)
sns.heatmap(cm, annot=True, fmt='g', cmap='Reds', xticklabels=lr.classes_, yticklabels=lr.classes_)
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
y_test_pred = lr.predict(tf_test)
# Save
output = pd.DataFrame({'ArticleId': test['ArticleId'], 'Category': y_test_pred})
output.to_csv('lr100.csv', index=False)
```

```
Validation Accuracy: 0.9652777777777778
```



75% Lables:

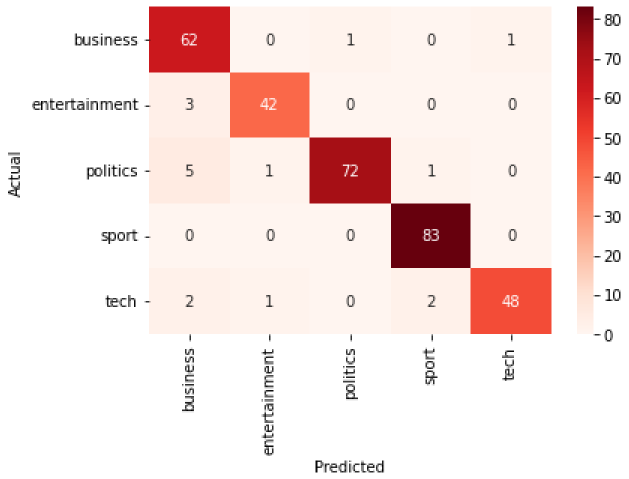
In [15]:

```
train_75 = train.sample(frac=0.75, random_state=26)
tf = TfidfVectorizer(sublinear_tf=True,
                            max_df=0.8, #try 0.5, 0.7, 0.9
                            min_df=8, # try 6, 8, 10
                            stop_words="english")
tf.fit(train_75["Text"])
tf_train = tf.transform(train_75["Text"])
tf_test = tf.transform(test["Text"])
X_train, X_valid, y_train, y_valid = train_test_split(tf_train, train_75['Category'], test_size=0.3, random_state=26)
```

In [16]:

```
lr = LogisticRegression(random_state=26)
lr.fit(X_train, y_train)
y_pred = lr.predict(X_valid)
acc = accuracy_score(y_valid, y_pred)
print("Validation Accuracy:", acc)
cm = confusion_matrix(y_valid, y_pred)
sns.heatmap(cm, annot=True, fmt='g', cmap='Reds', xticklabels=lr.classes_, yticklabels=lr.classes_)
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
y_test_pred = lr.predict(tf_test)
# Save
output = pd.DataFrame({'ArticleId': test['ArticleId'], 'Category': y_test_pred})
output.to_csv('lr75.csv', index=False)
```

```
Validation Accuracy: 0.9475308641975309
```



50% Lables:

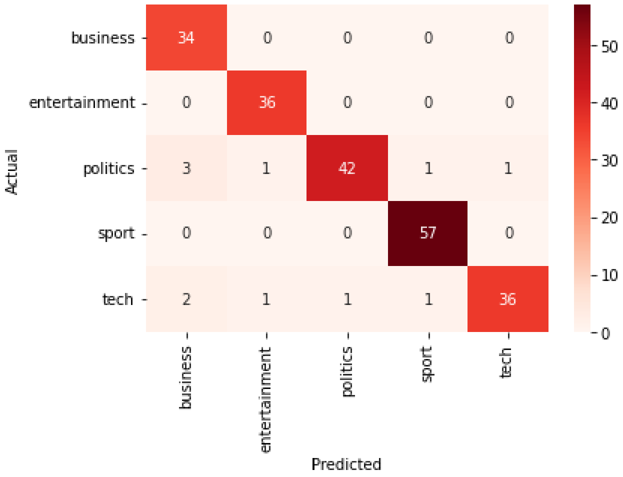
In [17]:

```
train_50 = train.sample(frac=0.50, random_state=26)
tf = TfidfVectorizer(sublinear_tf=True,
                            max_df=0.8, #try 0.5, 0.7, 0.9
                            min_df=8, # try 6, 8, 10
                            stop_words="english")
tf.fit(train_50["Text"])
tf_train = tf.transform(train_50["Text"])
tf_test = tf.transform(test["Text"])
X_train, X_valid, y_train, y_valid = train_test_split(tf_train, train_50['Category'], test_size=0.3, random_state=26)
```

In [18]:

```
lr = LogisticRegression(random_state=26)
lr.fit(X_train, y_train)
y_pred = lr.predict(X_valid)
acc = accuracy_score(y_valid, y_pred)
print("Validation Accuracy:", acc)
cm = confusion_matrix(y_valid, y_pred)
sns.heatmap(cm, annot=True, fmt='g', cmap='Reds', xticklabels=lr.classes_, yticklabels=lr.classes_)
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
y_test_pred = lr.predict(tf_test)
# Save
output = pd.DataFrame({'ArticleId': test['ArticleId'], 'Category': y_test_pred})
output.to_csv('lr50.csv', index=False)
```

```
Validation Accuracy: 0.9490740740740741
```



25% Lables:

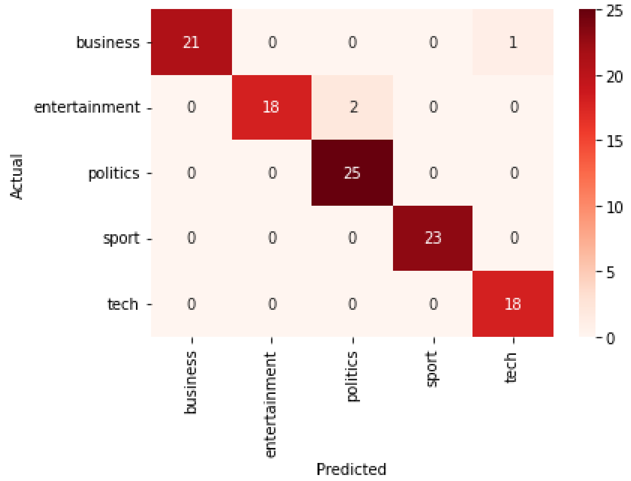
In [19]:

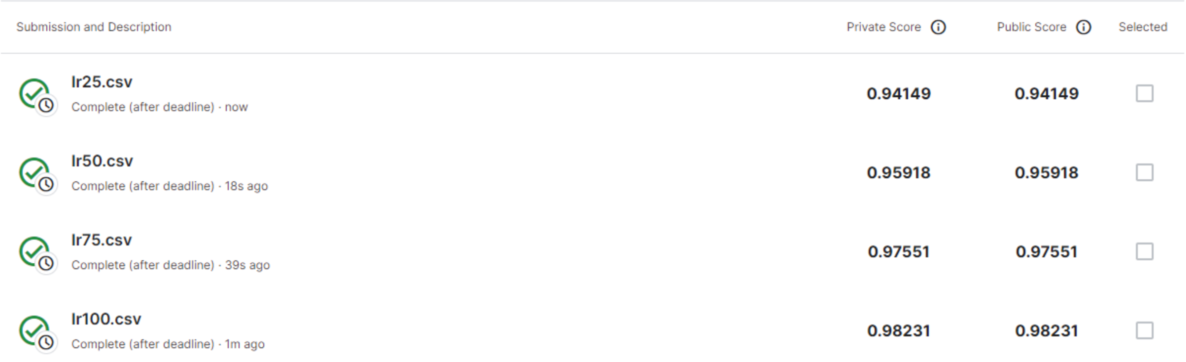
```
train_25 = train.sample(frac=0.25, random_state=26)
tf = TfidfVectorizer(sublinear_tf=True,
                            max_df=0.8, #try 0.5, 0.7, 0.9
                            min_df=8, # try 6, 8, 10
                            stop_words="english")
tf.fit(train_25["Text"])
tf_train = tf.transform(train_25["Text"])
tf_test = tf.transform(test["Text"])
X_train, X_valid, y_train, y_valid = train_test_split(tf_train, train_25['Category'], test_size=0.3, random_state=26)
```

In [20]:

```
lr = LogisticRegression(random_state=26)
lr.fit(X_train, y_train)
y_pred = lr.predict(X_valid)
acc = accuracy_score(y_valid, y_pred)
print("Validation Accuracy:", acc)
cm = confusion_matrix(y_valid, y_pred)
sns.heatmap(cm, annot=True, fmt='g', cmap='Reds', xticklabels=lr.classes_, yticklabels=lr.classes_)
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
y_test_pred = lr.predict(tf_test)
# Save
output = pd.DataFrame({'ArticleId': test['ArticleId'], 'Category': y_test_pred})
output.to_csv('lr25.csv', index=False)
```

```
Validation Accuracy: 0.9722222222222222
```





Submission and Description	Private Score ⓘ	Public Score ⓘ	Selected
lr25.csv Complete (after deadline) · now	0.94149	0.94149	□
lr50.csv Complete (after deadline) · 18s ago	0.95918	0.95918	□
lr75.csv Complete (after deadline) · 39s ago	0.97551	0.97551	□
lr100.csv Complete (after deadline) · 1m ago	0.98231	0.98231	□

```
-----Logistic Regression Model Score: 0.98231(100%), 0.97551(75%), 0.95918(50%), 0.94149(25%)-----
```

1. The Logistic Regression achieves an accuracy of over 98% on the entire dataset.
2. On a 25% sample, the score is over 94%, indicating that it is very data-efficient.
3. In addition, there is no overfitting in Logistic Regression.

4. Building Unsupervised Model

Non-negative Matrix Facorization Model

100% Lables:

In [21]:

```
tf = TfidfVectorizer(sublinear_tf=True,
                     max_df=0.8,
                     min_df=8,
                     stop_words='english')
tf.fit(train['Text'])
tf_train = tf.transform(train['Text'])
tf_test = tf.transform(test['Text'])

nmf = NMF(train['Category'].nunique(), random_state=0)
nmf.fit(tf_train)
y_pred = np.array([np.argmax(i) for i in nmf.transform(tf_train)])
y_true = np.array(train.Category).reshape(-1)
uni_labels = np.unique(y_true)
best_perm = None
best_acc = 0
for i, label in enumerate(uni_labels):
    for perm in itertools.permutations(range(len(uni_labels))):
        perm_label_map = {label: int_label for label, int_label in zip(uni_labels, perm)}
        perm_labels = np.array([perm_label_map[label] for label in y_true])
        acc = np.mean(perm_labels == y_pred)
        if acc > best_acc:
            best_perm = {num: l for num, l in zip(perm, uni_labels)}
            best_acc = acc
print('Train Accuracy:', best_acc)

y_pred = np.array([np.argmax(i) for i in nmf.transform(tf_test)])
nmf100 = pd.DataFrame(columns=['ArticleId', 'Category'])
nmf100['ArticleId'] = test.ArticleId
nmf100['Category'] = [best_perm[i] for i in y_pred]
nmf100.to_csv('nmf100.csv', index=False)
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\decomposition\_nmf.py:289: FutureWarning: The 'init' value, when 'init=No
ne' and n_components is less than n_samples and n_features, will be changed from 'nndsvd' to 'nndsvda' in 1.1 (renaming of
0.26).
  warnings.warn(

Train Accuracy: 0.9486111111111111
```

75% Lables:

In [24]:

```
train_75 = train.sample(frac=0.75, random_state=26)
tf = TfidfVectorizer(sublinear_tf=True,
                     max_df=0.8,
                     min_df=8,
                     stop_words='english')
tf.fit(train_75['Text'])
tf_train = tf.transform(train_75['Text'])
tf_test = tf.transform(test['Text'])

nmf = NMF(train_75['Category'].nunique(), random_state=0)
nmf.fit(tf_train)
y_pred = np.array([np.argmax(i) for i in nmf.transform(tf_train)])
y_true = np.array(train_75.Category).reshape(-1)
uni_labels = np.unique(y_true)
best_perm = None
best_acc = 0
for i, label in enumerate(uni_labels):
    for perm in itertools.permutations(range(len(uni_labels))):
        perm_label_map = {label: int_label for label, int_label in zip(uni_labels, perm)}
        perm_labels = np.array([perm_label_map[label] for label in y_true])
        acc = np.mean(perm_labels == y_pred)
        if acc > best_acc:
            best_perm = {num: l for num, l in zip(perm, uni_labels)}
            best_acc = acc
print('Train Accuracy:', best_acc)

y_pred = np.array([np.argmax(i) for i in nmf.transform(tf_test)])
nmf75 = pd.DataFrame(columns=['ArticleId', 'Category'])
nmf75['ArticleId'] = test.ArticleId
nmf75['Category'] = [best_perm[i] for i in y_pred]
nmf75.to_csv('nmf75.csv', index=False)
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\decomposition\_nmf.py:289: FutureWarning: The 'init' value, when 'init=No
ne' and n_components is less than n_samples and n_features, will be changed from 'nndsvd' to 'nndsvda' in 1.1 (renaming of
0.26).
  warnings.warn(

Train Accuracy: 0.9407407407407408
```

50% Lables:

In [25]:

```python
train_50 = train.sample(frac=0.5, random_state=26)
tf = TfidfVectorizer(sublinear_tf=True,
                     max_df=0.8,
                     min_df=8,
                     stop_words='english')
tf.fit(train_50['Text'])
tf_train = tf.transform(train_50['Text'])
tf_test = tf.transform(test['Text'])

nmf = NMF(train_50['Category'].nunique(), random_state=0)
nmf.fit(tf_train)
y_pred = np.array([np.argmax(i) for i in nmf.transform(tf_train)])
y_true = np.array(train_50.Category).reshape(-1)
uni_labels = np.unique(y_true)
best_perm = None
best_acc = 0
for i, label in enumerate(uni_labels):
    for perm in itertools.permutations(range(len(uni_labels))):
        perm_label_map = {label: int_label for label, int_label in zip(uni_labels, perm)}
        perm_labels = np.array([perm_label_map[label] for label in y_true])
        acc = np.mean(perm_labels == y_pred)
        if acc > best_acc:
            best_perm = {num: l for num, l in zip(perm, uni_labels)}
            best_acc = acc
print('Train Accuracy:', best_acc)

y_pred = np.array([np.argmax(i) for i in nmf.transform(tf_test)])
nmf50 = pd.DataFrame(columns=['ArticleId', 'Category'])
nmf50['ArticleId'] = test.ArticleId
nmf50['Category'] = [best_perm[i] for i in y_pred]
nmf50.to_csv('nmf50.csv', index=False)
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\decomposition\_nmf.py:289: FutureWarning: The 'init' value, when 'init=No
ne' and n_components is less than n_samples and n_features, will be changed from 'nndsvd' to 'nndsvda' in 1.1 (renaming of
0.26).
  warnings.warn(

Train Accuracy: 0.9472222222222222
```

25% Lables:

In [27]:

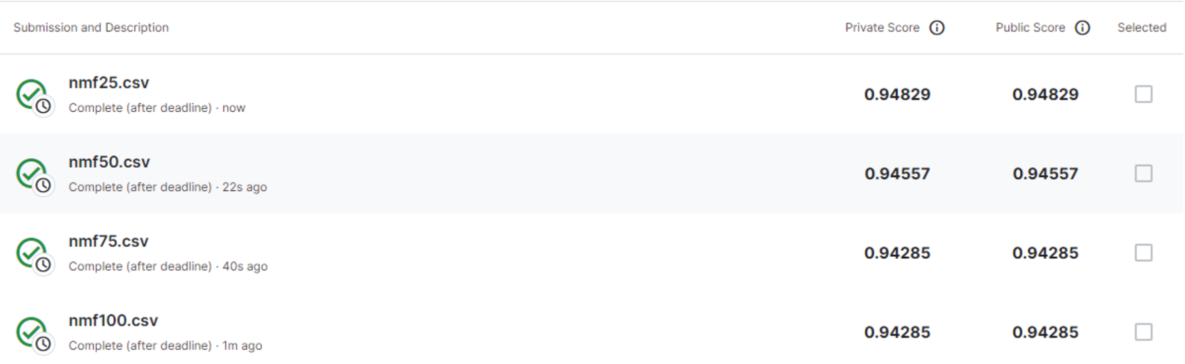
```python
train_25 = train.sample(frac=0.25, random_state=26)
tf = TfidfVectorizer(sublinear_tf=True,
                     max_df=0.8,
                     min_df=8,
                     stop_words='english')
tf.fit(train_25['Text'])
tf_train = tf.transform(train_25['Text'])
tf_test = tf.transform(test['Text'])

nmf = NMF(train_25['Category'].nunique(), random_state=0)
nmf.fit(tf_train)
y_pred = np.array([np.argmax(i) for i in nmf.transform(tf_train)])
y_true = np.array(train_25.Category).reshape(-1)
uni_labels = np.unique(y_true)
best_perm = None
best_acc = 0
for i, label in enumerate(uni_labels):
    for perm in itertools.permutations(range(len(uni_labels))):
        perm_label_map = {label: int_label for label, int_label in zip(uni_labels, perm)}
        perm_labels = np.array([perm_label_map[label] for label in y_true])
        acc = np.mean(perm_labels == y_pred)
        if acc > best_acc:
            best_perm = {num: l for num, l in zip(perm, uni_labels)}
            best_acc = acc
print('Train Accuracy:', best_acc)

y_pred = np.array([np.argmax(i) for i in nmf.transform(tf_test)])
nmf25 = pd.DataFrame(columns=['ArticleId', 'Category'])
nmf25['ArticleId'] = test.ArticleId
nmf25['Category'] = [best_perm[i] for i in y_pred]
nmf25.to_csv('nmf25.csv', index=False)
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\decomposition\_nmf.py:289: FutureWarning: The 'init' value, when 'init=No
ne' and n_components is less than n_samples and n_features, will be changed from 'nndsvd' to 'nndsvda' in 1.1 (renaming of
0.26).
  warnings.warn(

Train Accuracy: 0.9444444444444444
```



Submission and Description	Private Score	Public Score	Selected
nmf25.csv Complete (after deadline) · now	0.94829	0.94829	□
nmf50.csv Complete (after deadline) · 22s ago	0.94557	0.94557	□
nmf75.csv Complete (after deadline) · 40s ago	0.94285	0.94285	□
nmf100.csv Complete (after deadline) · 1m ago	0.94285	0.94285	□

-----Non-negative Matrix Facorization Model: 0.94285(100%), 0.94285(75%), 0.94557(50%), 0.94829(25%)-----

1. The Non-negative Matrix Facorization Model achieves an accuracy of over 94% on the entire dataset.
2. From 25% sample to 100%, the score maintains at 94%, indicating that the model is stable and very data-efficient.
3. In addition, there is no overfitting in Non-negative Matrix Facorization Model.

6. Conclusion

In this project, we used Logistic Regression model and Non-negative Matrix Facorization Model to predict the category of articles.

Both two models performed very well, with prediction accuracy over 94%, and Logistic Regression had the highest score.

Both two models show very stable performance when we narrow down the lable scope, indicating they are both data-efficient.

Besides, there is no overfitting signal.

Reference:

https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html (https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html)

https://docs.python.org/3/library/collections.html#collections.Counter (https://docs.python.org/3/library/collections.html#collections.Counter)

https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.non_negative_factorization.html (https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.non_negative_factorization.html)

In [ ]: