Text Classification

Notebook Content

Library and Data

In [1]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.feature_extraction.text import CountVectorizer
from sklearn import feature_selection, linear_model, model_selection, preprocessing
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.pipeline import Pipeline
from sklearn.svm import LinearSVC
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import MultinomialNB
from sklearn.naive_bayes import BernoulliNB
from sklearn.naive_bayes import MultinomialNB

from sklearn.ensemble import GradientBoostingClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import SGDClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.preprocessing import LabelEncoder
from sklearn.gaussian_process import GaussianProcessClassifier

from sklearn.metrics import accuracy_score, confusion_matrix,classification_report

import nltk
import nltk as nlp
import string
import re
true = pd.read_csv("dataset/True.csv")
fake = pd.read_csv("dataset/Fake.csv")
```

Go to top

Reading Data

In [2]:
```
fake['target'] = 'fake'
true['target'] = 'true'
news = pd.concat([fake, true]).reset_index(drop = True)
news.head()
```

Out[2]:

	title	text	subject	date	target
0	Donald Trump Sends Out Embarrassing New Year'...	Donald Trump just couldn t wish all Americans ...	News	December 31, 2017	fake
1	Drunk Bragging Trump Staffer Started Russian ...	House Intelligence Committee Chairman Devin Nu...	News	December 31, 2017	fake
2	Sheriff David Clarke Becomes An Internet Joke,...	On Friday, it was revealed that former Milwauk...	News	December 30, 2017	fake
3	Trump Is So Obsessed He Even Has Obama's Name...	On Christmas day, Donald Trump announced that ...	News	December 29, 2017	fake
4	Pope Francis Just Called Out Donald Trump Dur...	Pope Francis used his annual Christmas Day mes...	News	December 25, 2017	fake

Go to top

Logistic Regression Classifier

In this algorithm, the probabilities describing the possible outcomes of a single trial are modelled using a logistic function. It is used to model the probability of a certain class or event existing such as pass/fail, win/lose, alive/dead or healthy/sick.

In [3]:
```
x_train,x_test,y_train,y_test = train_test_split(news['text'], news.target, test_size=0.2, r
andom_state=2020)

pipe = Pipeline([('vect', CountVectorizer()),
                 ('tfidf', TfidfTransformer()),
                 ('model', LogisticRegression())])

model = pipe.fit(x_train, y_train)
prediction = model.predict(x_test)
print("accuracy: {}%".format(round(accuracy_score(y_test, prediction)*100,2)))
```

accuracy: 98.78%

In [4]:
```
print(confusion_matrix(y_test, prediction))
```

```
[[4674   66]
 [  46 4195]]
```

In [5]:
```
print(classification_report(y_test, prediction))
```

```
              precision    recall  f1-score   support

        fake       0.99      0.99      0.99      4740
        true       0.98      0.99      0.99      4241

    accuracy                           0.99      8981
   macro avg       0.99      0.99      0.99      8981
weighted avg       0.99      0.99      0.99      8981
```

Go to top

Support Vector Classifier

The support vector machine is a classifier that represents the training data as points in space separated into categories by a gap as wide as possible. New points are then added to space by predicting which category they fall into and which space they will belong to.

More often text classification use cases will have linearly separable data and LinearSVC is apt for such scenarios

In [6]:
```
x_train,x_test,y_train,y_test = train_test_split(news['text'], news.target, test_size=0.2, r
andom_state=2020)

pipe = Pipeline([('vect', CountVectorizer()),
                 ('tfidf', TfidfTransformer()),
                 ('model', LinearSVC())])

model = pipe.fit(x_train, y_train)
prediction = model.predict(x_test)
print("accuracy: {}%".format(round(accuracy_score(y_test, prediction)*100,2)))
```

accuracy: 99.55%

In [7]:
```
print(confusion_matrix(y_test, prediction))
```

```
[[4720   20]
 [  20 4221]]
```

In [8]:
```
print(classification_report(y_test, prediction))
```

```
              precision    recall  f1-score   support

        fake       1.00      1.00      1.00      4740
        true       1.00      1.00      1.00      4241

    accuracy                           1.00      8981
   macro avg       1.00      1.00      1.00      8981
weighted avg       1.00      1.00      1.00      8981
```

Go to top

Multinomial Naive Bayes Classifier

It is based on Bayes's theorem which gives an assumption of independence among predictors. A Naive Bayes classifier assumes that the presence of a particular feature in a class is unrelated to the presence of any other feature. Multinomialnb implements the naive Bayes algorithm for multinomially distributed data.it works with occurrence counts of words and to form vector

In [9]:
```
pipe = Pipeline([('vect', CountVectorizer()),
                 ('tfidf', TfidfTransformer()),
                 ('model', MultinomialNB())])

model = pipe.fit(x_train, y_train)
prediction = model.predict(x_test)
print("accuracy: {}%".format(round(accuracy_score(y_test, prediction)*100,2)))
```

accuracy: 93.56%

In [10]:
```
print(confusion_matrix(y_test, prediction))
```

```
[[4686  254]
 [ 324 3917]]
```

In [11]:
```
print(classification_report(y_test, prediction))
```

```
              precision    recall  f1-score   support

        fake       0.93      0.95      0.94      4740
        true       0.94      0.92      0.93      4241

    accuracy                           0.94      8981
   macro avg       0.94      0.93      0.94      8981
weighted avg       0.94      0.94      0.94      8981
```

Go to top

Bernoulli Naive Bayes Classifier

BernoulliNB implements the naive Bayes training and classification algorithms for data that is distributed according to multivariate Bernoulli distributions; i.e., there may be multiple features but each one is assumed to be a binary-valued (Bernoulli, boolean) variable. Therefore, this class requires samples to be represented as binary-valued feature vectors

In [12]:
```
pipe = Pipeline([('vect', CountVectorizer()),
                 ('tfidf', TfidfTransformer()),
                 ('model', BernoulliNB())])

model = pipe.fit(x_train, y_train)
prediction = model.predict(x_test)
print("accuracy: {}%".format(round(accuracy_score(y_test, prediction)*100,2)))
```

accuracy: 94.14%

In [13]:
```
print(confusion_matrix(y_test, prediction))
```

```
[[4377  363]
 [ 163 4077]]
```

In [14]:
```
print(classification_report(y_test, prediction))
```

```
              precision    recall  f1-score   support

        fake       0.96      0.92      0.94      4740
        true       0.92      0.96      0.94      4241

    accuracy                           0.94      8981
   macro avg       0.94      0.94      0.94      8981
weighted avg       0.94      0.94      0.94      8981
```

Go to top

Gradient Boost Classifier

GB builds an additive model in a forward stage-wise fashion it allows for the optimization of arbitrary differentiable loss functions. Binary classification is a special case where only a single regression tree is induced.

In [15]:
```
pipe = Pipeline([('vect', CountVectorizer()),
                 ('tfidf', TfidfTransformer()),
                 ('model', GradientBoostingClassifier(loss = 'deviance',
                                                      learning_rate = 0.01,
                                                      n_estimators = 10,
                                                      max_depth = 5,
                                                      random_state=55))])

model = pipe.fit(x_train, y_train)
prediction = model.predict(x_test)
print("accuracy: {}%".format(round(accuracy_score(y_test, prediction)*100,2)))
```

accuracy: 99.52%

In [16]:
```
print(confusion_matrix(y_test, prediction))
```

```
[[4707   33]
 [  10 4230]]
```

In [17]:
```
print(classification_report(y_test, prediction))
```

```
              precision    recall  f1-score   support

        fake       1.00      0.99      1.00      4740
        true       0.99      1.00      0.99      4241

    accuracy                           1.00      8981
   macro avg       1.00      1.00      1.00      8981
weighted avg       1.00      1.00      1.00      8981
```

Go to top

XGBoost Classifier

XGBoost is a decision-tree-based ensemble Machine Learning algorithm that uses a gradient boosting framework. Rather than training all the models in isolation of one another, boosting trains models in succession with each new model being trained to correct the errors made by the previous ones.

In a standard ensemble method where models are trained in isolation, all of the models might simply end up making the same mistakes. We should use this algorithm when we require fast and accurate predictions after the model is deployed

In [18]:
```
pipe = Pipeline([('vect', CountVectorizer()),
                 ('tfidf', TfidfTransformer()),
                 ('model', XGBClassifier(loss = 'deviance',
                                                      learning_rate = 0.01,
                                                      n_estimators = 10,
                                                      max_depth = 5,
                                                      random_state=2020))])

model = pipe.fit(x_train, y_train)
prediction = model.predict(x_test)
print("accuracy: {}%".format(round(accuracy_score(y_test, prediction)*100,2)))
```

```
[07:50:19] WARNING: C:\Users\Administrator\workspace\xgboost-win64_release_1.1.0\src\learner.
cc:480:
Parameters: { loss } might not be used.

  This may not be accurate due to some parameters are only used in language bindings but
  passed down to XGBoost core.  Or some parameters are not used but slip through this
  verification. Please open an issue if you find above cases.
```

accuracy: 99.52%

In [19]:
```
print(confusion_matrix(y_test, prediction))
```

```
[[4707   33]
 [  10 4230]]
```

In [20]:
```
print(classification_report(y_test, prediction))
```

```
              precision    recall  f1-score   support

        fake       1.00      0.99      1.00      4740
        true       0.99      1.00      0.99      4241

    accuracy                           1.00      8981
   macro avg       1.00      1.00      1.00      8981
weighted avg       1.00      1.00      1.00      8981
```

Go to top

Random Forest Classifier

Random forest are an ensemble learning method. It operates by constructing a multitude of decision trees at training time and outputs the class that is the mode of the classes of the individual trees. A random forest is a meta-estimator that fits a number of trees on various subsamples of data sets and then uses an average to improve the accuracy in the model's predictive nature. The sub-sample size is always the same as that of the original input size but the samples are often drawn with replacements.

We should use this algorithm when we need high accuracy while working with large datasets with higher dimensions. We can also use it if there are missing values in the dataset. We should not use it if we have less time for modeling or if large computational costs and memory space are a constrain.

In [21]:
```
pipe = Pipeline([('vect', CountVectorizer()),
                 ('tfidf', TfidfTransformer()),
                 ('model', RandomForestClassifier())])

model = pipe.fit(x_train, y_train)
prediction = model.predict(x_test)
print("accuracy: {}%".format(round(accuracy_score(y_test, prediction)*100,2)))
```

accuracy: 99.18%

In [22]:
```
print(confusion_matrix(y_test, prediction))
```

```
[[4705   35]
 [  39 4202]]
```

In [23]:
```
print(classification_report(y_test, prediction))
```

```
              precision    recall  f1-score   support

        fake       0.99      0.99      0.99      4740
        true       0.99      0.99      0.99      4241

    accuracy                           0.99      8981
   macro avg       0.99      0.99      0.99      8981
weighted avg       0.99      0.99      0.99      8981
```

Go to top

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]: