Document/Text Classification using various models

Name: Colin Antony
College: M S Ramaiah University of Applied Sciences

Synopsis:

Problem Statement

Large amounts of text data are just raw data with no means of providing information unless processed and classified. In this assignment, a set of documents (20 newsgroup) will be classified into predefined categories using various models. We may also use unsupervised machine learning and cluster similar documents together. I will try to compare as many models as I can and try to do some unsupervised learning clustering too.

Objectives

1. To pre-process the raw data by removing objects like stop words, punctuations, and convert them into a standardized format.
2. To extract features from the models using various methods such as TF-IDF, Word embeddings, etc.
3. To use various models on the extracted features to classify the documents and compare all the models with each other using various metrics such as Accuracy, Precision, Recall, F1-score, etc.

General Steps:

1. Data Collection: Collect a set of documents from different domains and annotate them with one or more labels or categories.
2. Data Pre-processing: Clean and pre-process the documents by removing stop words, punctuations, and other irrelevant characters, and convert them into a standardized format.
3. Feature Extraction: Extract features from the pre-processed documents using different methods, such as TF-IDF, Word Embeddings, or N-grams.
4. Model Selection: Select various machine learning models, such as Naive Bayes, SVM, Decision Trees, Random Forest, and Neural Networks, and evaluate their performance using cross-validation or train-test split.
5. Hyperparameter Tuning: Tune the hyperparameters of the selected models using grid search or randomized search to optimize their performance.
6. Model Evaluation: Evaluate the performance of each model using metrics such as Accuracy, Precision, Recall, F1-score, and Confusion Matrix, and compare them to select the best performing model.
7. Model Deployment: Deploy the selected model to classify new documents into predefined categories and evaluate its performance on unseen data.

Expected Outcome

The expected outcome is to identify the best supervised/ unsupervised model for document classification using various evaluation metrics.

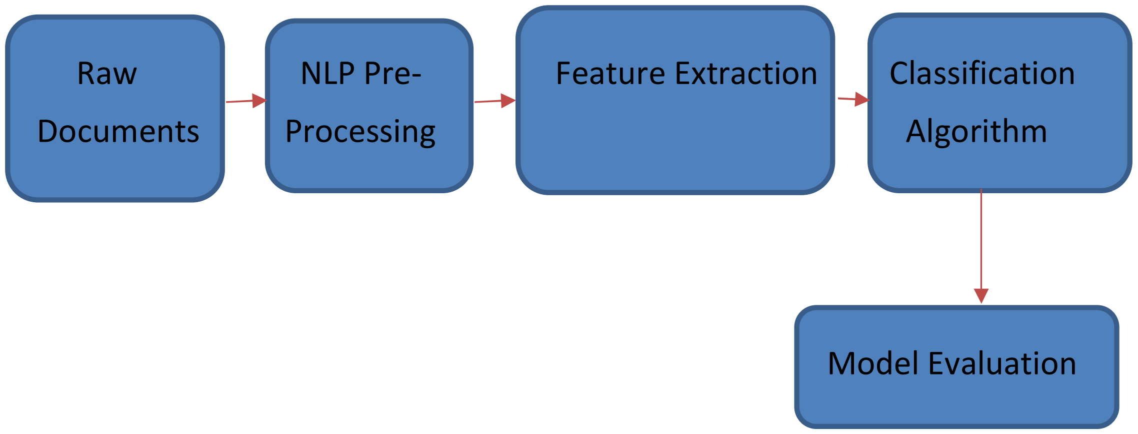
Method and Methodology:

Data:

The dataset used in this assignment is the “20 Newsgroup” dataset which is a part of the scikit-learn library. It is a dataset that contains 20,000 newsgroup documents that cover a variety of topics such as politics, sports, religion among many others. A newsgroup document refers to a newsgroup post which are similar to emails in their format and content. However, instead of being sent directly to other people, they are posted to public newsgroups.

Block Diagram:

Below is the general block diagram of a text or document classifier:



Problem Domain:

Text or Document classification falls under Natural Language Processing (NLP) and Machine Leaning (ML). In this, we categorize the text or data into different predefined categories based on their content. Some applications of text classification are:

- Information retrieval
- Spam filtering
- Sentiment analysis
- Topic modelling.

Algorithm

Even though various models are used in this assignment, all of them follow a general algorithm. Here, the general algorithm will be displayed. More details on specific algorithm will be under the implementation section next to each model.

1. *Data Pre-processing:*
    - Involves converting text to lowercase, removing punctuations, numbers, and special characters.
    - Tokenization of text, removing stop words, stemming or lemmatizing
2. *Feature Extraction:*
    - Create a bag of words using a method like CountVectorizer()
    - Perform TFI-DF transformation on it.
3. *Split Data*: Split the dataset into training and testing
4. *Train the model:* Choose the classification algorithm to be used.
5. *Evaluate the model:* Use the classification report metric
6. *Fine tune the model:*
    - Perform Hyperparameter tuning
    - Perform Regularization
    - Use above 2 methods only if necessary
7. *Apply the model:* Use the trained model to classify new documents

Technologies Used

There are many technologies I have used in this assignment to classify the "20 Newsgroups" dataset. Some of them are:

1. Natural Language Processing(NLP): Many NLP techniques have been used to prepare the data to be fed to the classifier.
2. Bag-of-Words(BoW) model: It is a simple way to represent text as a numerical vector. Each element represents the frequency of occurrence of a word in a document. It is a feature extraction technique used in document classification.
3. Term Frequency-Inverse Document Frequency (TF-IDF): It assigns weights to words based on the its frequency and importance in the document. It is also part of feature extraction.
4. Various Models used: The models picked and used include Naïve Bayes Model, Support Vector Machine(SVM), Random Forest Classifier and Logistic Regression.

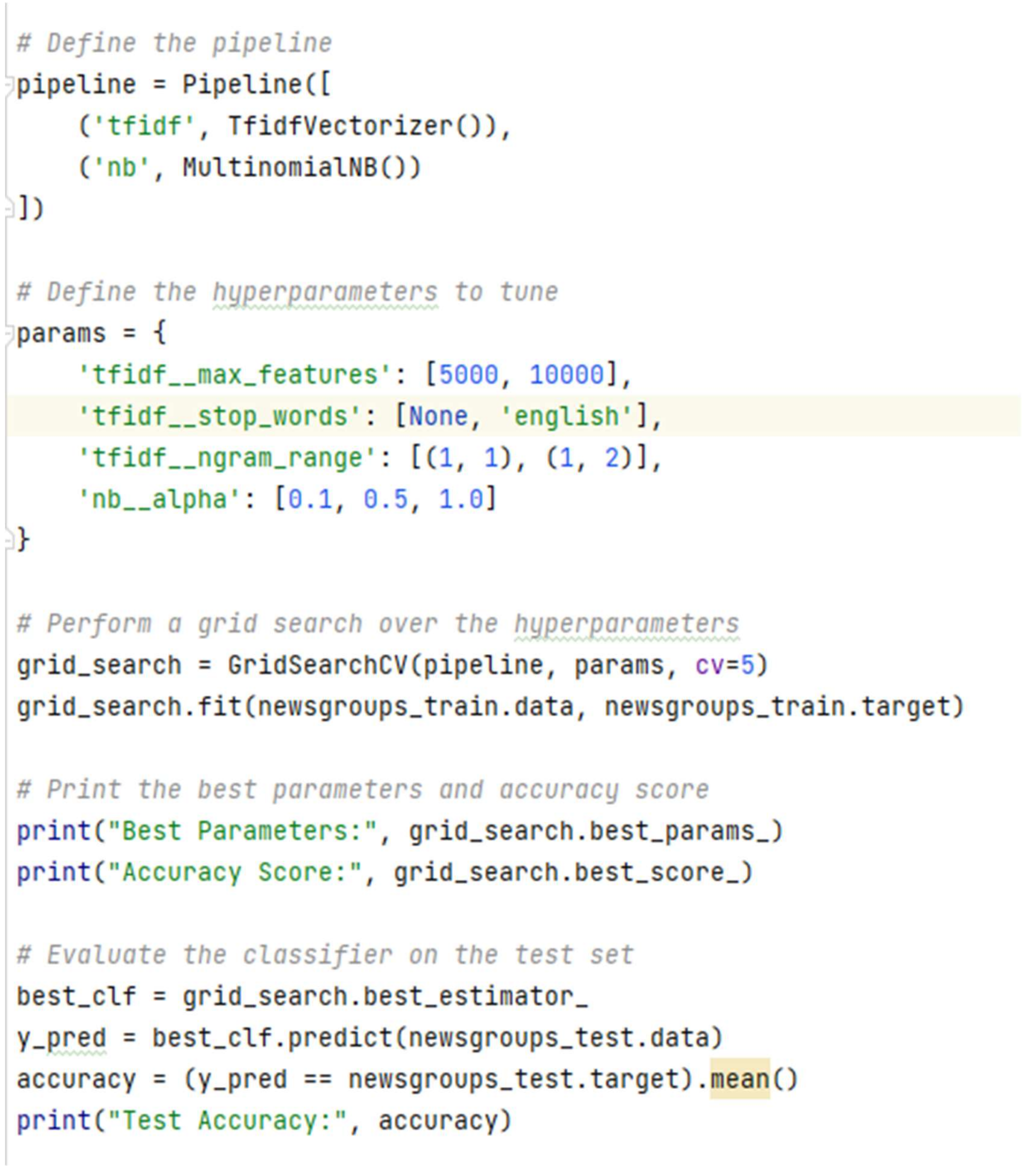
Implementation

For each implementation, the hyperparameter tuning has been done via trial-and-error, and a GridSearchCV() method. This method trains the model over a range of hyperparameters. We can select the best parameters for us by seeing which combination of parameters provides us with the best results.

In this assignment with a combined use of GridSearchCV() and various trial-and-error manually done by me, I have selected the best(to the best of my ability) hyper-parameters, regularization values for each model.

I have not included the GridSearchCV() method in the actual program codes because it is computationally very expensive, hence I have run it only once for each model and will be showing a sample of how GridSearchCV() works.

Here is an example of how Grid Search code looks:



```python
# Define the pipeline
pipeline = Pipeline([
    ('tfidf', TfidfVectorizer()),
    ('nb', MultinomialNB())
])

# Define the hyperparameters to tune
params = {
    'tfidf__max_features': [5000, 10000],
    'tfidf__stop_words': [None, 'english'],
    'tfidf__ngram_range': [(1, 1), (1, 2)],
    'nb__alpha': [0.1, 0.5, 1.0]
}

# Perform a grid search over the hyperparameters
grid_search = GridSearchCV(pipeline, params, cv=5)
grid_search.fit(newsgroups_train.data, newsgroups_train.target)

# Print the best parameters and accuracy score
print("Best Parameters:", grid_search.best_params_)
print("Accuracy Score:", grid_search.best_score_)

# Evaluate the classifier on the test set
best_clf = grid_search.best_estimator_
y_pred = best_clf.predict(newsgroups_test.data)
accuracy = (y_pred == newsgroups_test.target).mean()
print("Test Accuracy:", accuracy)
```

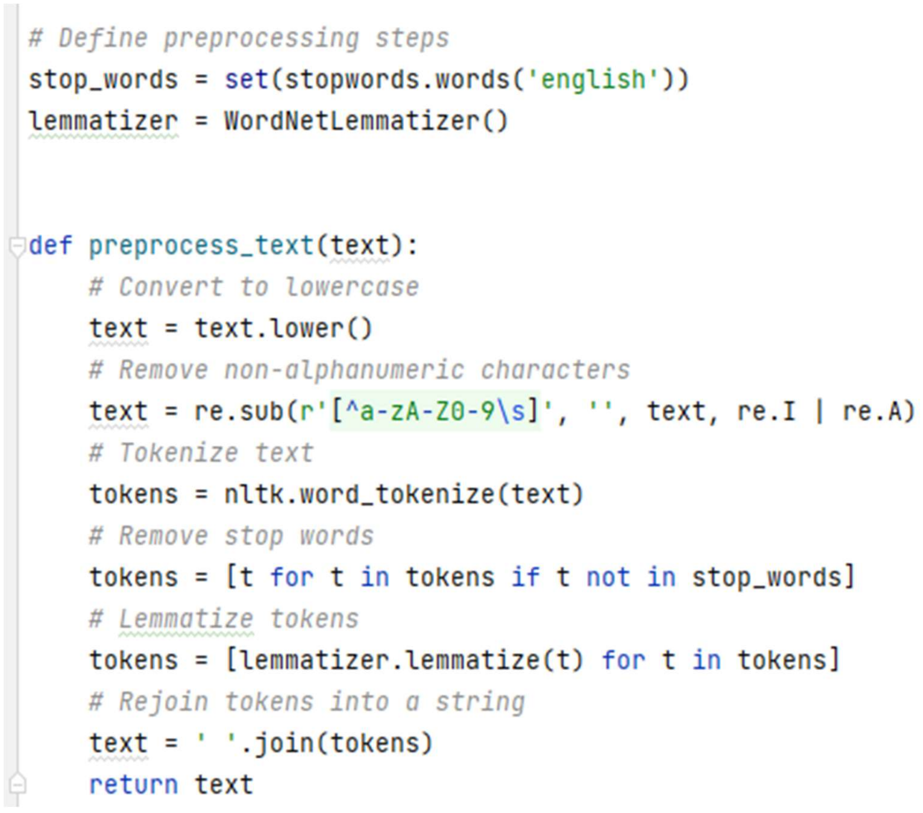
The main thing to note is the 'params' dictionary. This contains a list of all parameter values we would like to test out to get the best output.

Another method that I have used in all the models is the preprocess_text method. This is a method that I defined that performs text preprocessing on the model. The preprocessing includes:

I. Lowering the case
II. Removing all non-alphanumeric characters and replacing them with blank space.
III. Performing word tokenization

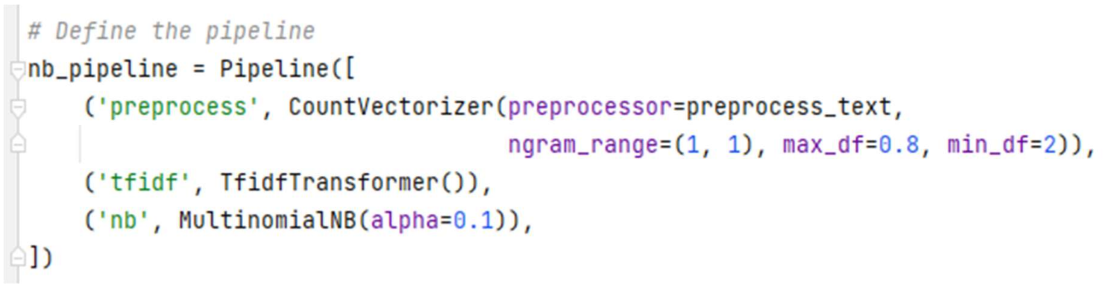
Here is an example of what the code looks like. I have called this method inside the pipeline building. It has a parameter called preprocessor which can be set to a user defined pre processing method. Here is the code:

Method code:



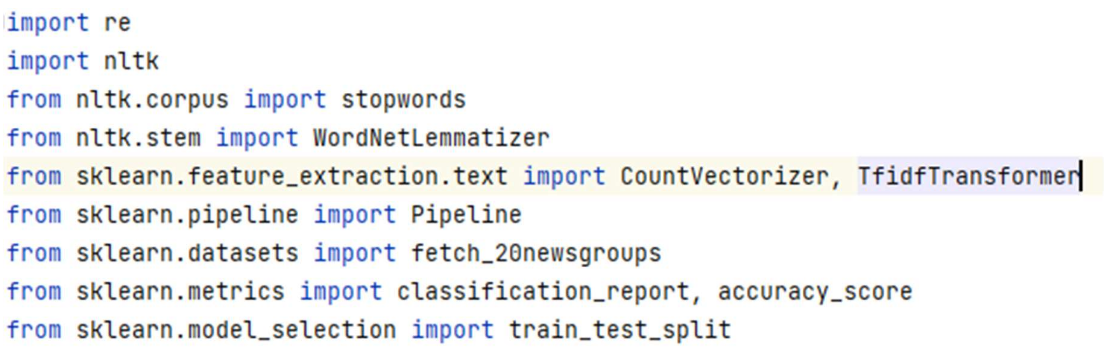
```python
# Define preprocessing steps
stop_words = set(stopwords.words('english'))
lemmatizer = WordNetLemmatizer()


def preprocess_text(text):
    # Convert to lowercase
    text = text.lower()
    # Remove non-alphanumeric characters
    text = re.sub(r'[^a-zA-Z0-9\s]', '', text, re.I | re.A)
    # Tokenize text
    tokens = nltk.word_tokenize(text)
    # Remove stop words
    tokens = [t for t in tokens if t not in stop_words]
    # Lemmatize tokens
    tokens = [lemmatizer.lemmatize(t) for t in tokens]
    # Rejoin tokens into a string
    text = ' '.join(tokens)
    return text
```

Example of using it in the pipeline:



```python
# Define the pipeline
nb_pipeline = Pipeline([
    ('preprocess', CountVectorizer(preprocessor=preprocess_text,
                                   ngram_range=(1, 1), max_df=0.8, min_df=2)),
    ('tfidf', TfidfTransformer()),
    ('nb', MultinomialNB(alpha=0.1)),
])
```

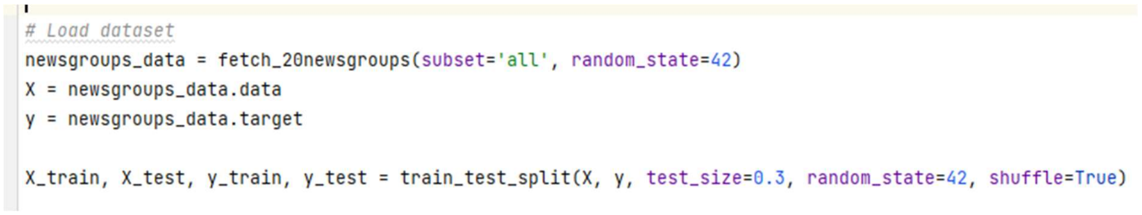
Here are the imports that are common to all the codes:



```python
import re
import nltk
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
from sklearn.feature_extraction.text import CountVectorizer, TfidfTransformer
from sklearn.pipeline import Pipeline
from sklearn.datasets import fetch_20newsgroups
from sklearn.metrics import classification_report, accuracy_score
from sklearn.model_selection import train_test_split
```

These imports are used in all the models that I have made. Will show the specific imports to each model as I display them.

Here is the training and testing data set splitting which is also common to all the codes:



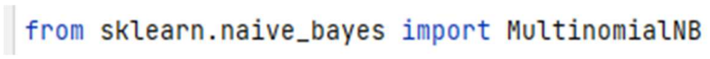
```python
# Load dataset
newsgroups_data = fetch_20newsgroups(subset='all', random_state=42)
X = newsgroups_data.data
y = newsgroups_data.target

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42, shuffle=True)
```

Models

Naïve-Bayes model

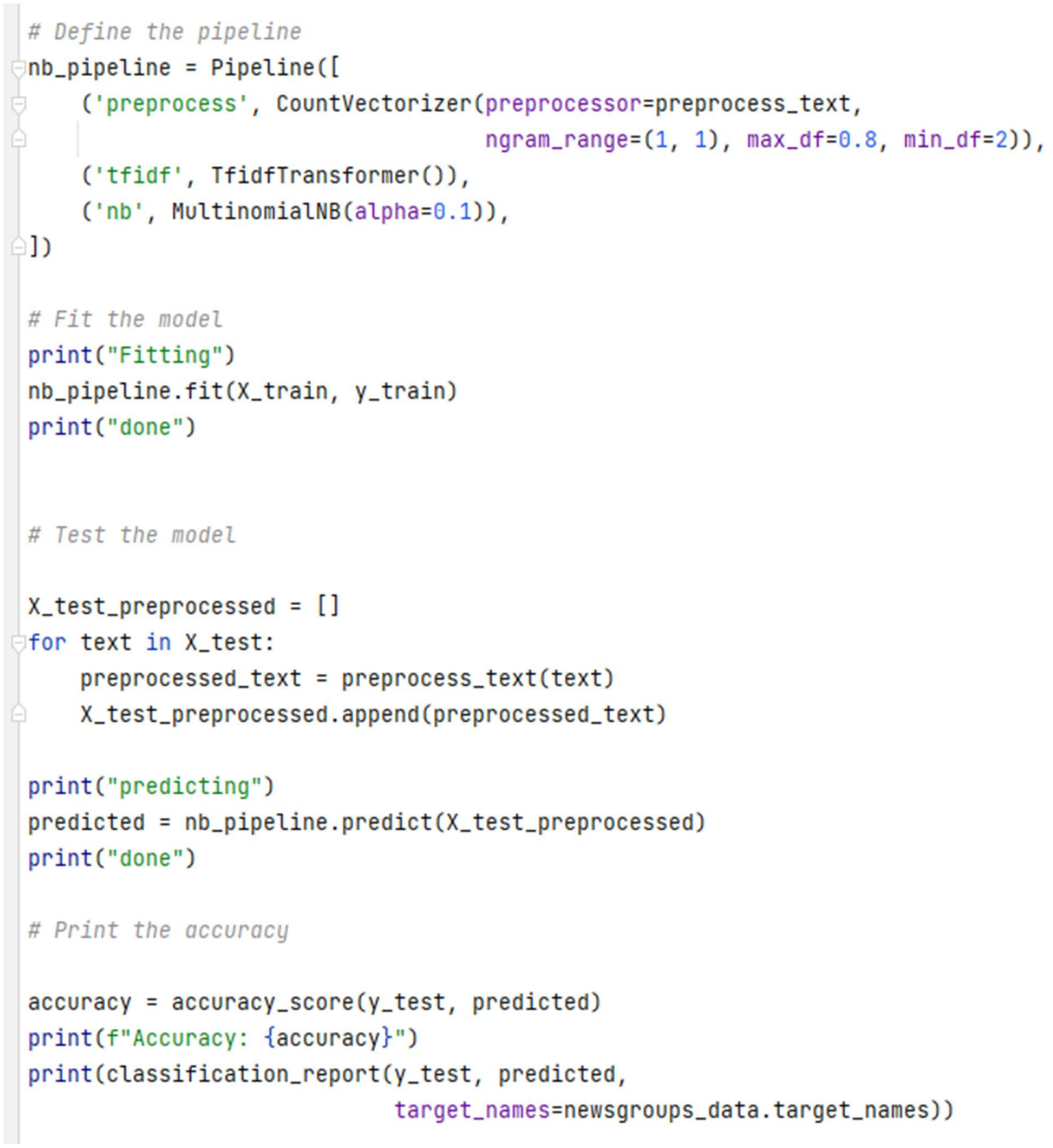
Uses multinomial Naïve Bayes

Package import:



```python
from sklearn.naive_bayes import MultinomialNB
```

Code:

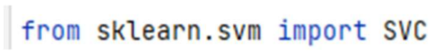


```python
# Define the pipeline
nb_pipeline = Pipeline([
    ('preprocess', CountVectorizer(preprocessor=preprocess_text,
                                   ngram_range=(1, 1), max_df=0.8, min_df=2)),
    ('tfidf', TfidfTransformer()),
    ('nb', MultinomialNB(alpha=0.1)),
])

# Fit the model
print("Fitting")
nb_pipeline.fit(X_train, y_train)
print("done")


# Test the model

X_test_preprocessed = []
for text in X_test:
    preprocessed_text = preprocess_text(text)
    X_test_preprocessed.append(preprocessed_text)

print("predicting")
predicted = nb_pipeline.predict(X_test_preprocessed)
print("done")

# Print the accuracy

accuracy = accuracy_score(y_test, predicted)
print(f"Accuracy: {accuracy}")
print(classification_report(y_test, predicted,
                            target_names=newsgroups_data.target_names))
```

Support Vector Machine (SVM)

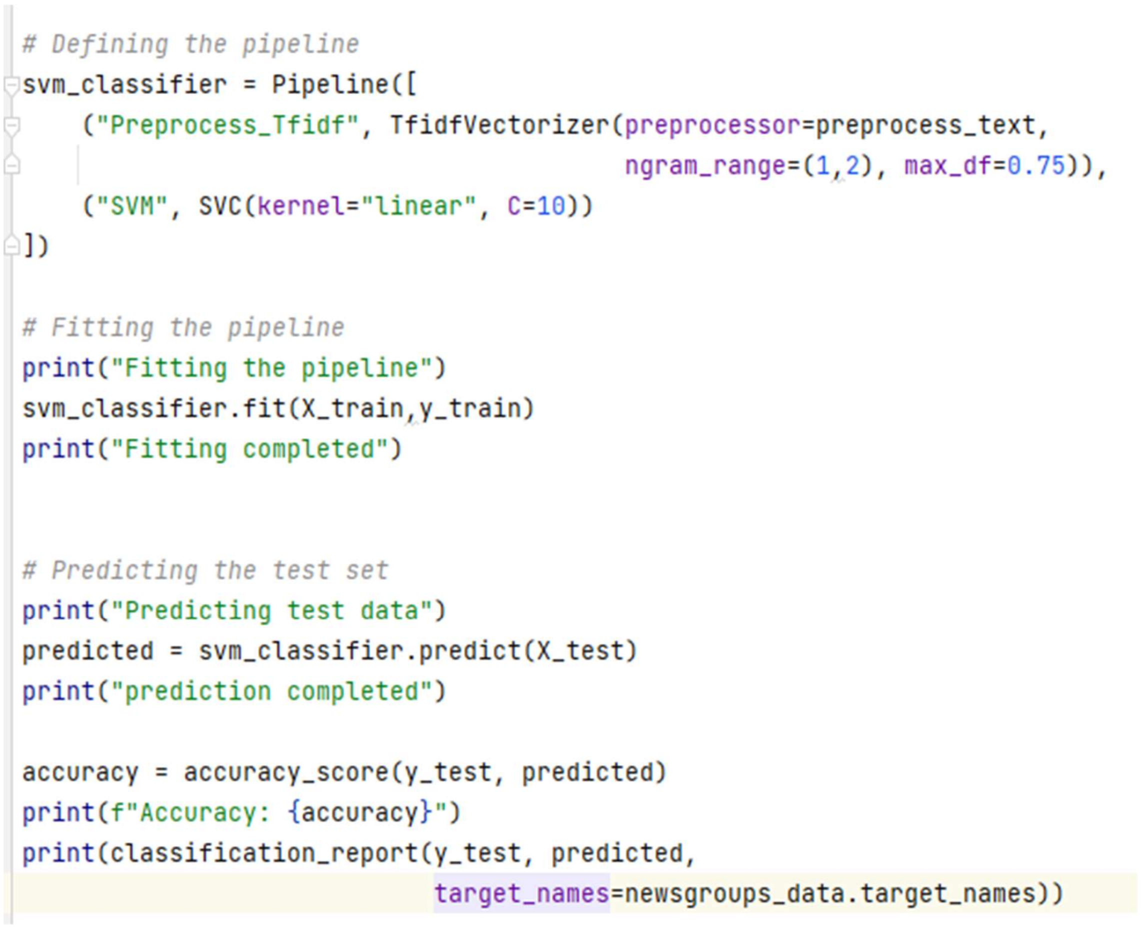
Uses SVC (Support Vector classification)

Package import:



```python
from sklearn.svm import SVC
```

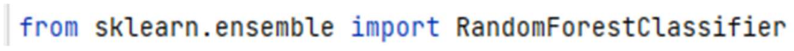
Code:



```python
# Defining the pipeline
svm_classifier = Pipeline([
    ("Preprocess_Tfidf", TfidfVectorizer(preprocessor=preprocess_text,
                                         ngram_range=(1,2), max_df=0.75)),
    ("SVM", SVC(kernel="linear", C=10))
])

# Fitting the pipeline
print("Fitting the pipeline")
svm_classifier.fit(X_train,y_train)
print("Fitting completed")


# Predicting the test set
print("Predicting test data")
predicted = svm_classifier.predict(X_test)
print("prediction completed")

accuracy = accuracy_score(y_test, predicted)
print(f"Accuracy: {accuracy}")
print(classification_report(y_test, predicted,
                            target_names=newsgroups_data.target_names))
```

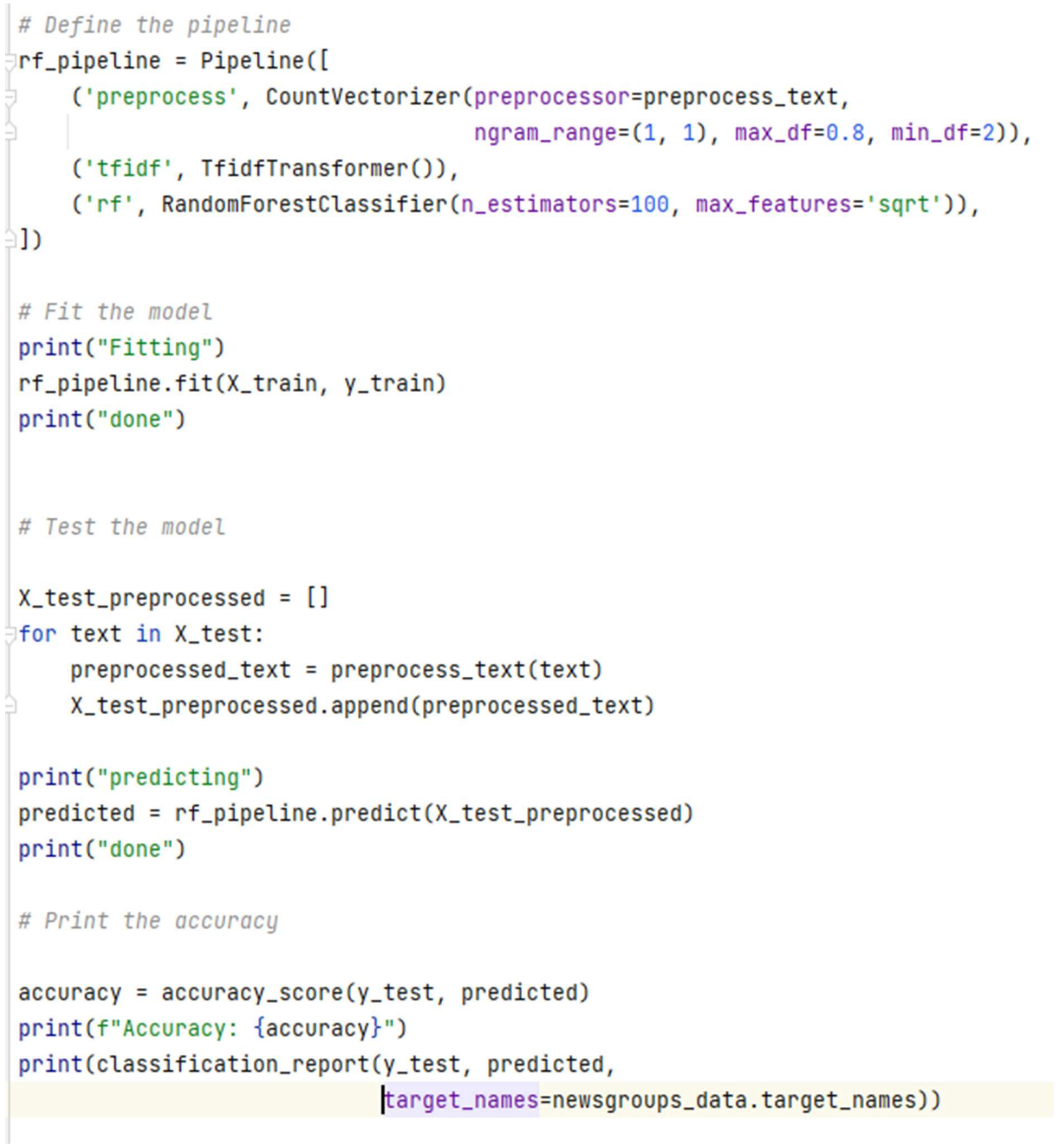
Random Forest Classifier

Package import:



```python
from sklearn.ensemble import RandomForestClassifier
```

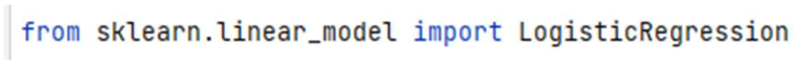
Code:



```python
# Define the pipeline
rf_pipeline = Pipeline([
    ('preprocess', CountVectorizer(preprocessor=preprocess_text,
                                   ngram_range=(1, 1), max_df=0.8, min_df=2)),
    ('tfidf', TfidfTransformer()),
    ('rf', RandomForestClassifier(n_estimators=100, max_features='sqrt')),
])

# Fit the model
print("Fitting")
rf_pipeline.fit(X_train, y_train)
print("done")


# Test the model

X_test_preprocessed = []
for text in X_test:
    preprocessed_text = preprocess_text(text)
    X_test_preprocessed.append(preprocessed_text)

print("predicting")
predicted = rf_pipeline.predict(X_test_preprocessed)
print("done")

# Print the accuracy

accuracy = accuracy_score(y_test, predicted)
print(f"Accuracy: {accuracy}")
print(classification_report(y_test, predicted,
                            target_names=newsgroups_data.target_names))
```

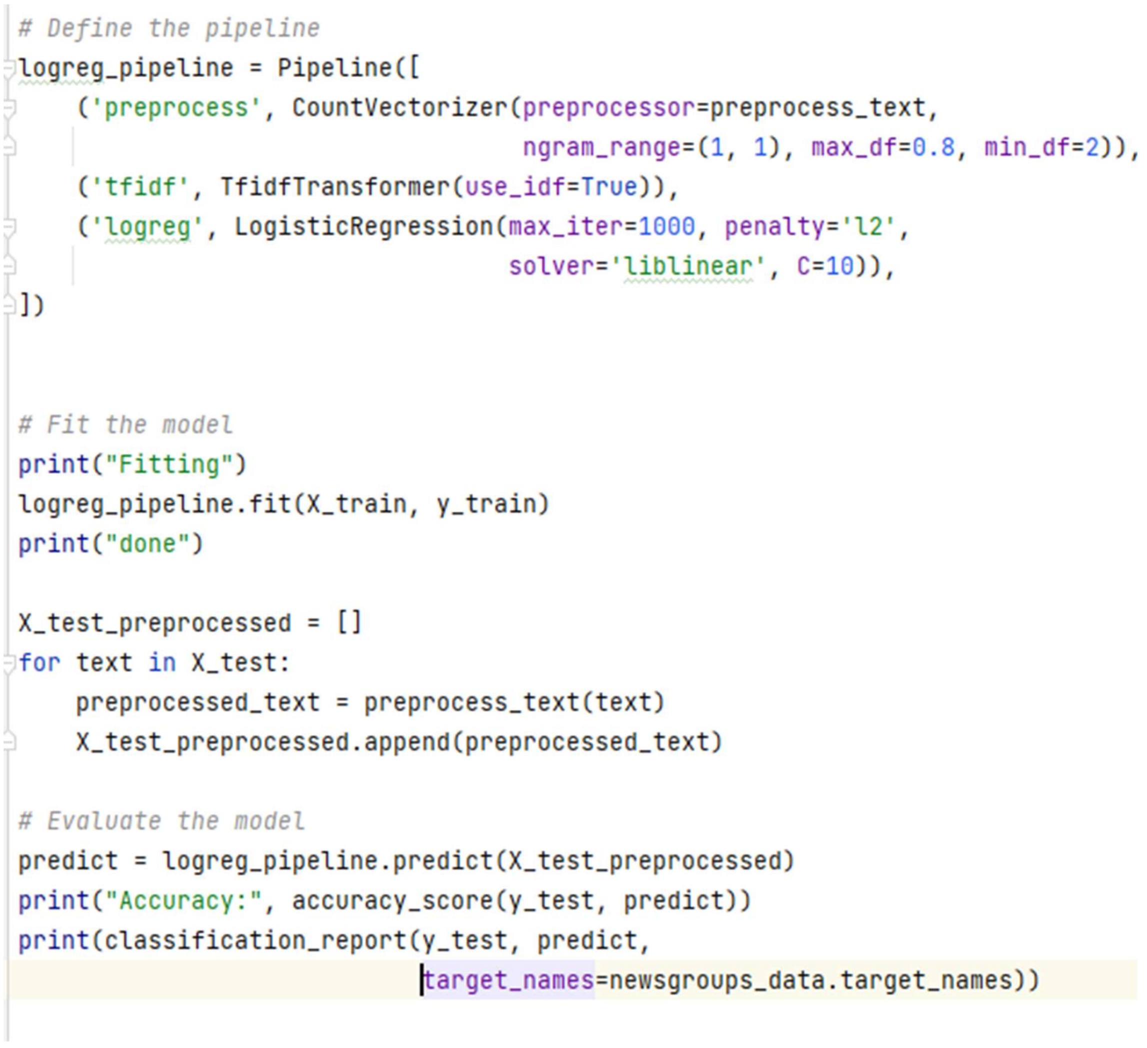
Logistic Regression

Package import:



```python
from sklearn.linear_model import LogisticRegression
```

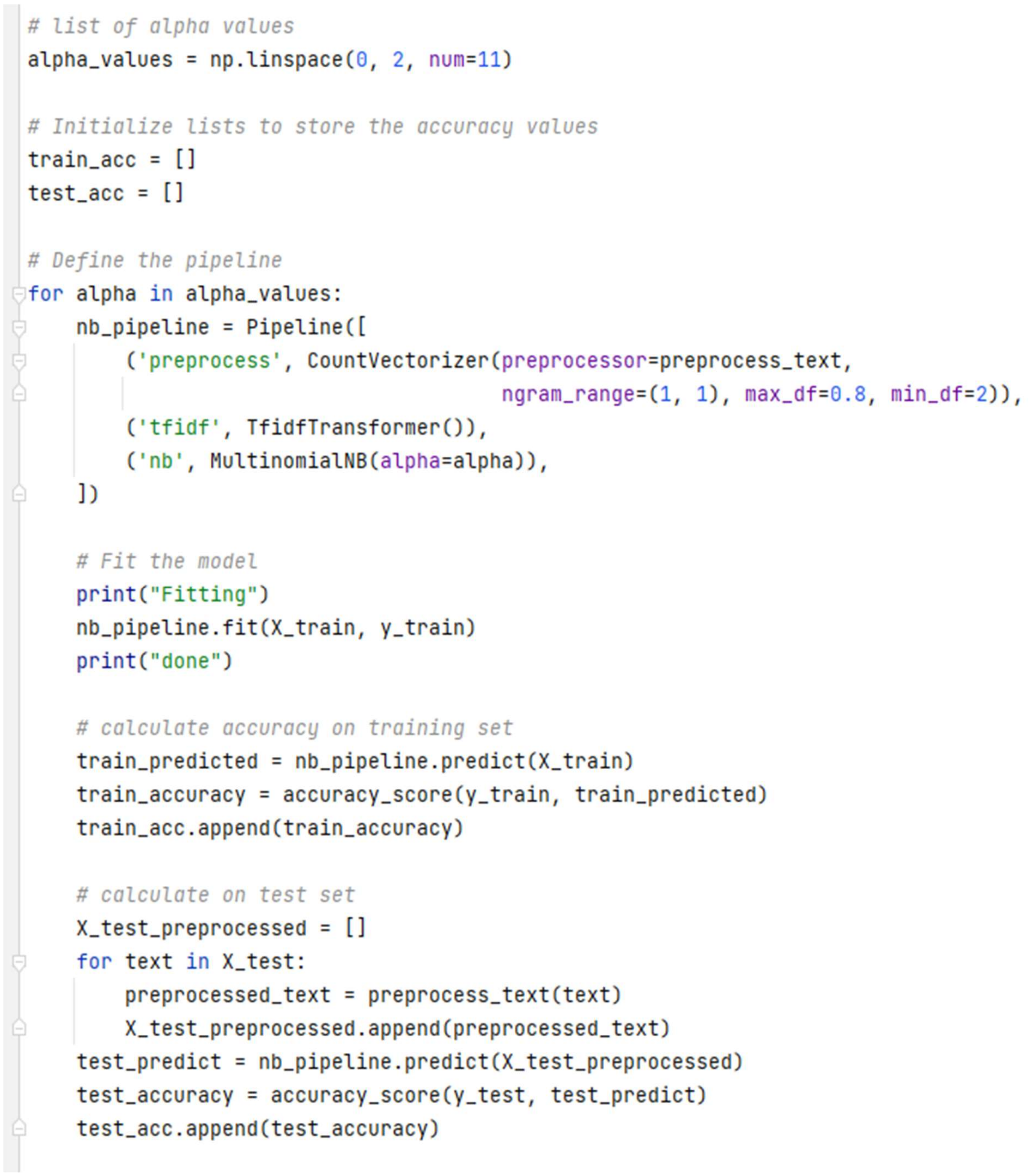
Code:



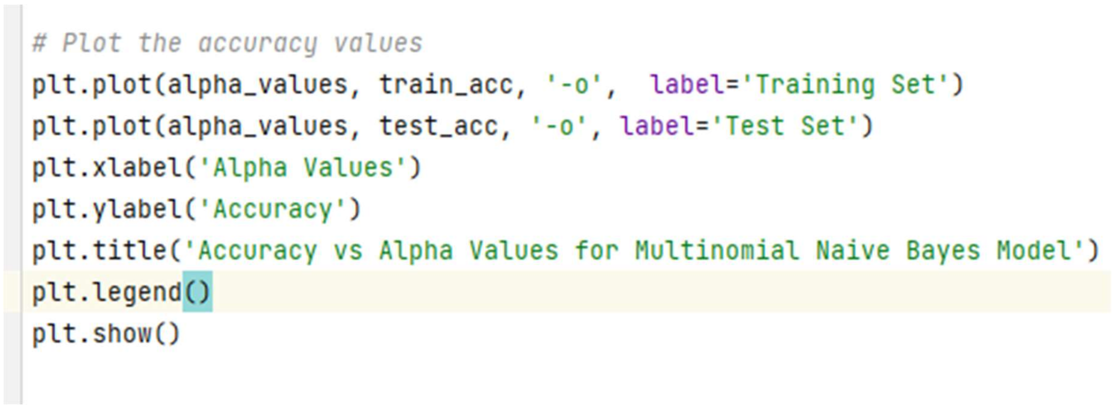
```python
# Define the pipeline
logreg_pipeline = Pipeline([
    ('preprocess', CountVectorizer(preprocessor=preprocess_text,
                                   ngram_range=(1, 1), max_df=0.8, min_df=2)),
    ('tfidf', TfidfTransformer(use_idf=True)),
    ('logreg', LogisticRegression(max_iter=1000, penalty='l2',
                                  solver='liblinear', C=10)),
])


# Fit the model
print("Fitting")
logreg_pipeline.fit(X_train, y_train)
print("done")

X_test_preprocessed = []
for text in X_test:
    preprocessed_text = preprocess_text(text)
    X_test_preprocessed.append(preprocessed_text)

# Evaluate the model
predict = logreg_pipeline.predict(X_test_preprocessed)
print("Accuracy:", accuracy_score(y_test, predict))
print(classification_report(y_test, predict,
                            target_names=newsgroups_data.target_names))
```

Graph with Naïve Bayes

This graph code shows the plot of accuracy vs alpha value:



```python
# list of alpha values
alpha_values = np.linspace(0, 2, num=11)

# Initialize lists to store the accuracy values
train_acc = []
test_acc = []

# Define the pipeline
for alpha in alpha_values:
    nb_pipeline = Pipeline([
        ('preprocess', CountVectorizer(preprocessor=preprocess_text,
                                       ngram_range=(1, 1), max_df=0.8, min_df=2)),
        ('tfidf', TfidfTransformer()),
        ('nb', MultinomialNB(alpha=alpha)),
    ])

    # Fit the model
    print("Fitting")
    nb_pipeline.fit(X_train, y_train)
    print("done")

    # calculate accuracy on training set
    train_predicted = nb_pipeline.predict(X_train)
    train_accuracy = accuracy_score(y_train, train_predicted)
    train_acc.append(train_accuracy)

    # calculate on test set
    X_test_preprocessed = []
    for text in X_test:
        preprocessed_text = preprocess_text(text)
        X_test_preprocessed.append(preprocessed_text)
    test_predict = nb_pipeline.predict(X_test_preprocessed)
    test_accuracy = accuracy_score(y_test, test_predict)
    test_acc.append(test_accuracy)
```



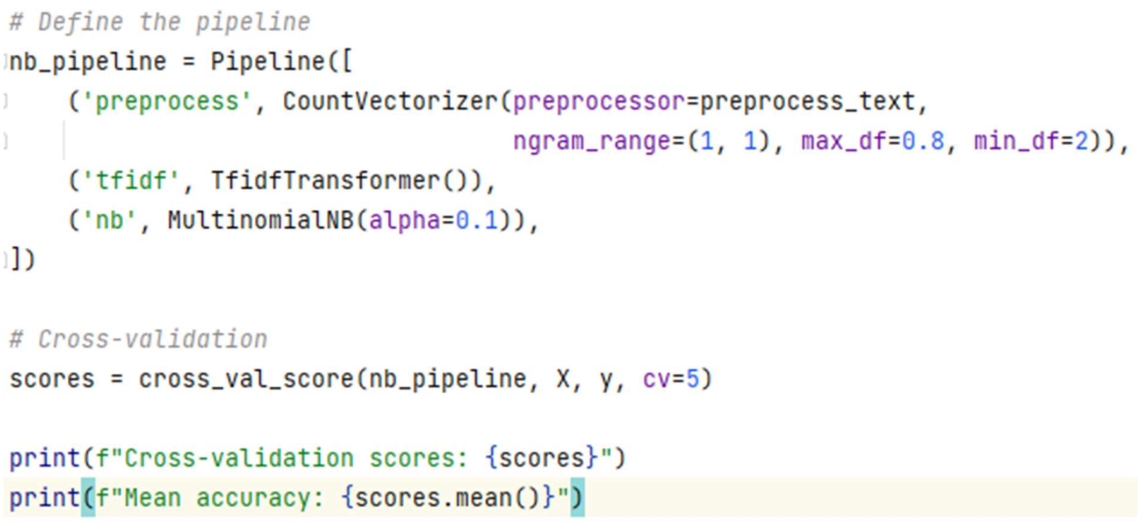
```python
# Plot the accuracy values
plt.plot(alpha_values, train_acc, '-o',  label='Training Set')
plt.plot(alpha_values, test_acc, '-o', label='Test Set')
plt.xlabel('Alpha Values')
plt.ylabel('Accuracy')
plt.title('Accuracy vs Alpha Values for Multinomial Naive Bayes Model')
plt.legend()
plt.show()
```

Naïve Bayes cross validation:

Package import:

```python
from sklearn.model_selection import  cross_val_score
```

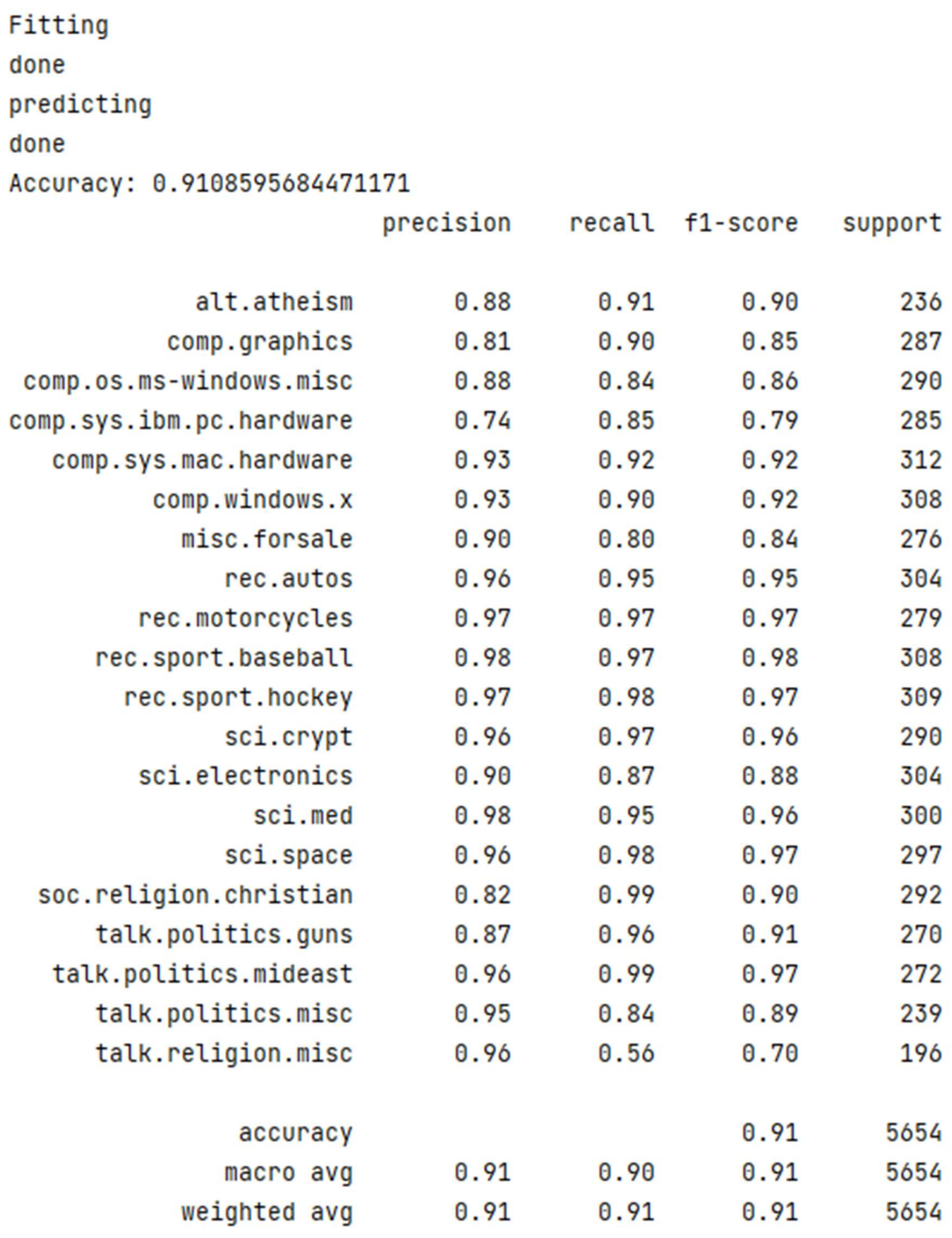
Code:



```python
# Define the pipeline
nb_pipeline = Pipeline([
    ('preprocess', CountVectorizer(preprocessor=preprocess_text,
                                   ngram_range=(1, 1), max_df=0.8, min_df=2)),
    ('tfidf', TfidfTransformer()),
    ('nb', MultinomialNB(alpha=0.1)),
])

# Cross-validation
scores = cross_val_score(nb_pipeline, X, y, cv=5)

print(f"Cross-validation scores: {scores}")
print(f"Mean accuracy: {scores.mean()}")
```

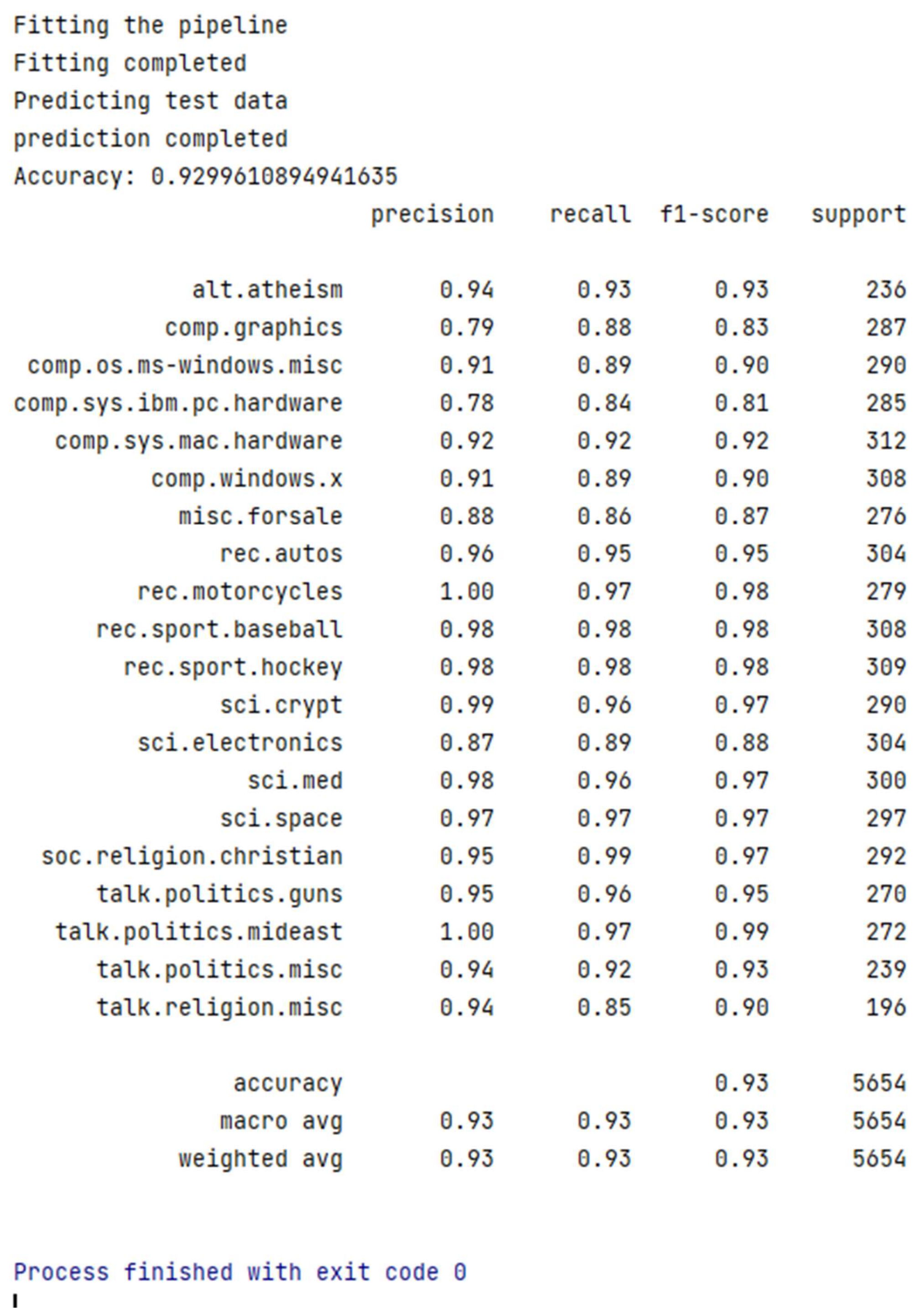
Results

Naïve Bayes Result:



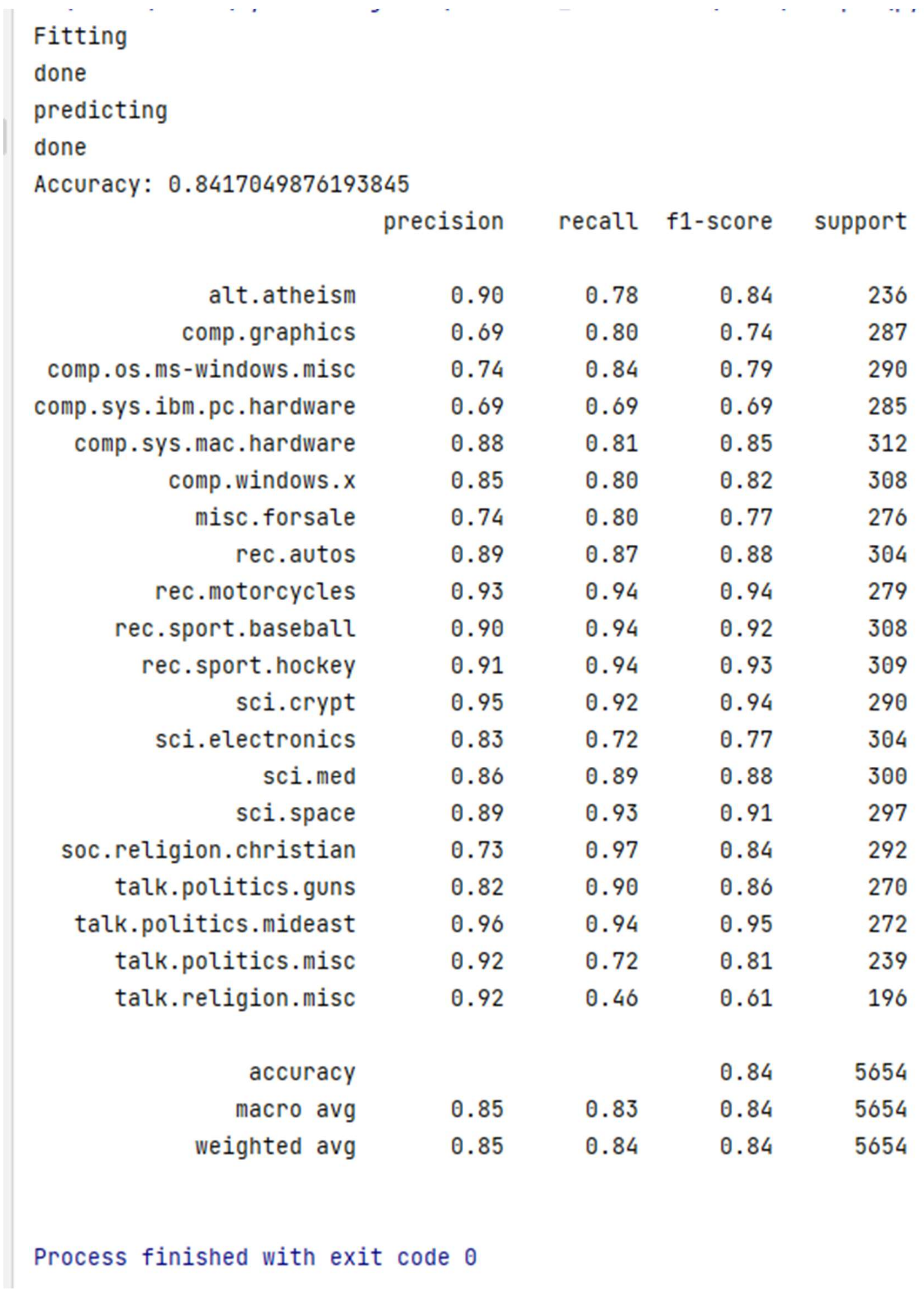
```
Fitting
done
predicting
done
Accuracy: 0.9108595684471171
                          precision    recall  f1-score   support

             alt.atheism       0.88      0.91      0.90       236
           comp.graphics       0.81      0.90      0.85       287
 comp.os.ms-windows.misc       0.88      0.84      0.86       290
comp.sys.ibm.pc.hardware       0.74      0.85      0.79       285
   comp.sys.mac.hardware       0.93      0.92      0.92       312
          comp.windows.x       0.93      0.90      0.92       308
            misc.forsale       0.90      0.80      0.84       276
               rec.autos       0.96      0.95      0.95       304
         rec.motorcycles       0.97      0.97      0.97       279
      rec.sport.baseball       0.98      0.97      0.98       308
        rec.sport.hockey       0.97      0.98      0.97       309
               sci.crypt       0.96      0.97      0.96       290
         sci.electronics       0.90      0.87      0.88       304
                 sci.med       0.98      0.95      0.96       300
               sci.space       0.96      0.98      0.97       297
  soc.religion.christian       0.82      0.99      0.90       292
      talk.politics.guns       0.87      0.96      0.91       270
   talk.politics.mideast       0.96      0.99      0.97       272
      talk.politics.misc       0.95      0.84      0.89       239
      talk.religion.misc       0.96      0.56      0.70       196

                accuracy                           0.91      5654
               macro avg       0.91      0.90      0.91      5654
            weighted avg       0.91      0.91      0.91      5654
```

SVM Result:



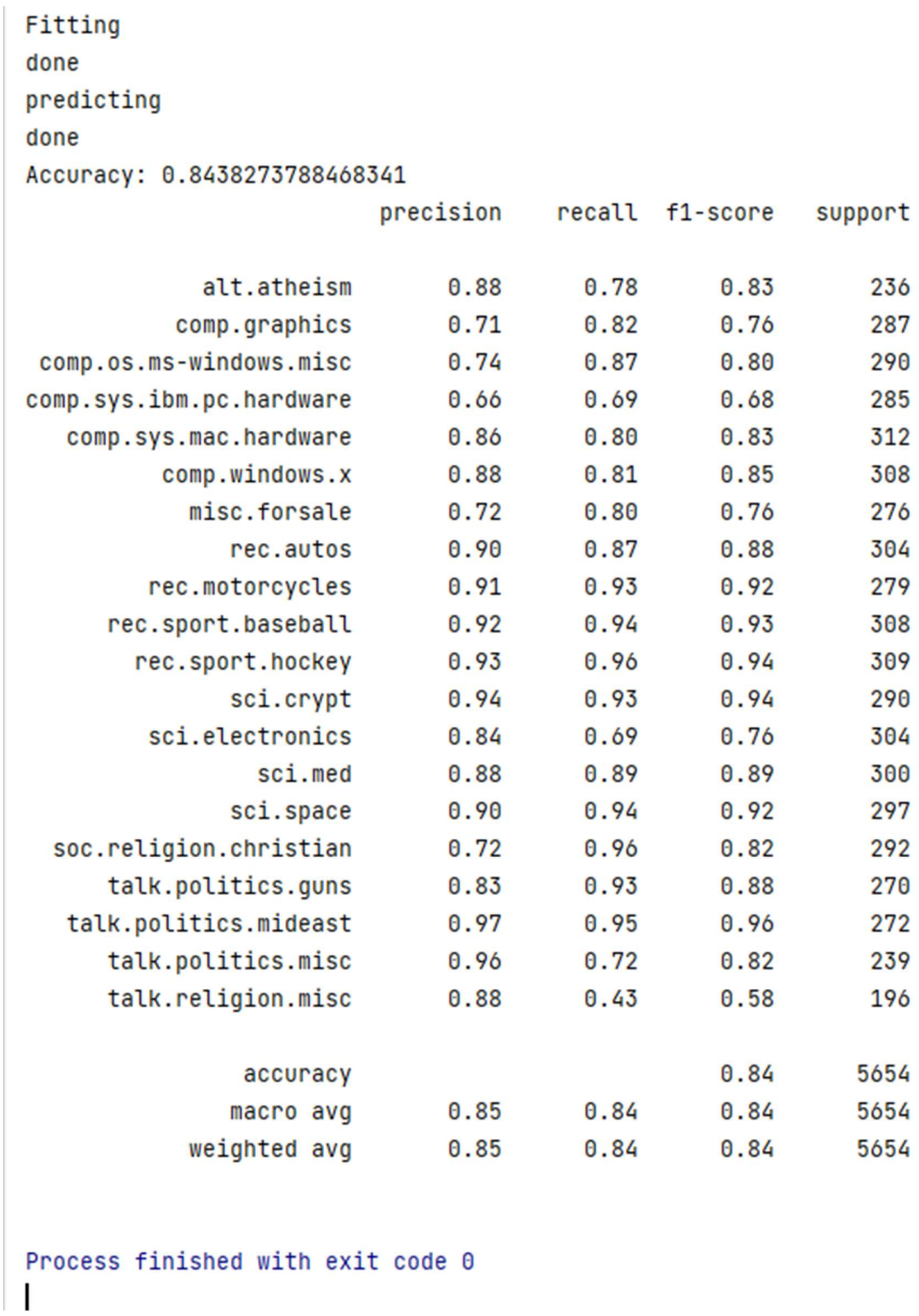
```
Fitting the pipeline
Fitting completed
Predicting test data
prediction completed
Accuracy: 0.9299610894941635
                          precision    recall  f1-score   support

             alt.atheism       0.94      0.93      0.93       236
           comp.graphics       0.79      0.88      0.83       287
 comp.os.ms-windows.misc       0.91      0.89      0.90       290
comp.sys.ibm.pc.hardware       0.78      0.84      0.81       285
   comp.sys.mac.hardware       0.92      0.92      0.92       312
          comp.windows.x       0.91      0.89      0.90       308
            misc.forsale       0.88      0.86      0.87       276
               rec.autos       0.96      0.95      0.95       304
         rec.motorcycles       1.00      0.97      0.98       279
      rec.sport.baseball       0.98      0.98      0.98       308
        rec.sport.hockey       0.98      0.98      0.98       309
               sci.crypt       0.99      0.96      0.97       290
         sci.electronics       0.87      0.89      0.88       304
                 sci.med       0.98      0.96      0.97       300
               sci.space       0.97      0.97      0.97       297
  soc.religion.christian       0.95      0.99      0.97       292
      talk.politics.guns       0.95      0.96      0.95       270
   talk.politics.mideast       1.00      0.97      0.99       272
      talk.politics.misc       0.94      0.92      0.93       239
      talk.religion.misc       0.94      0.85      0.90       196

                accuracy                           0.93      5654
               macro avg       0.93      0.93      0.93      5654
            weighted avg       0.93      0.93      0.93      5654


Process finished with exit code 0
```

Random Forest Classifier Results:



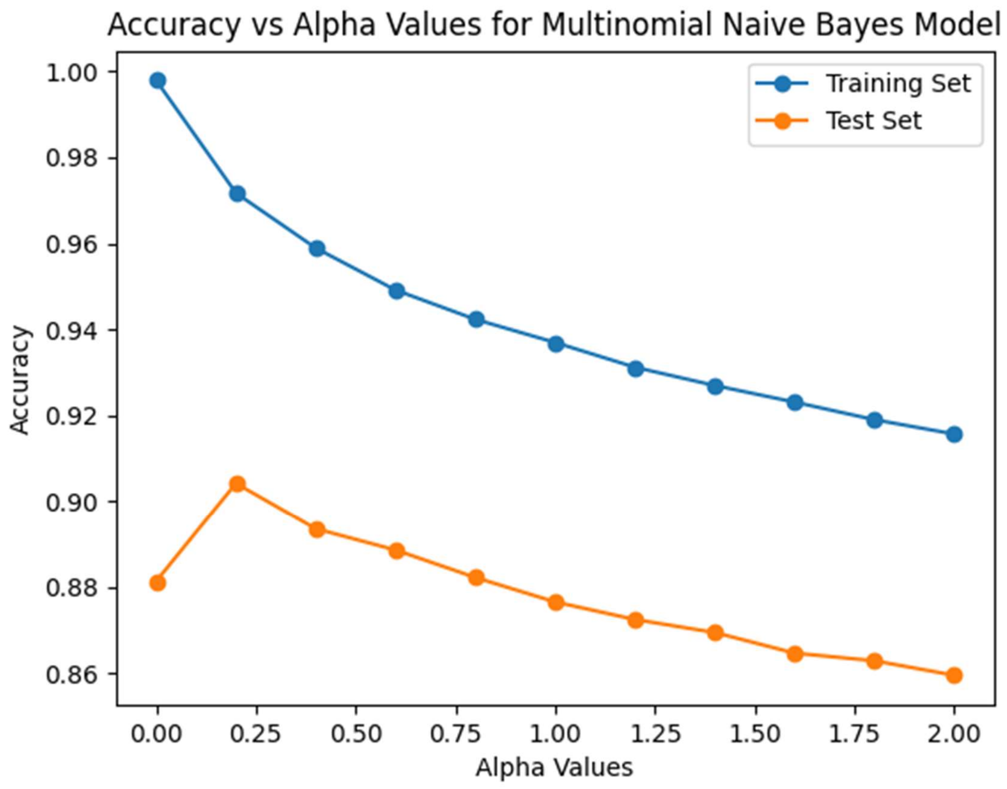
```
Fitting
done
predicting
done
Accuracy: 0.8417049876193845
                          precision    recall  f1-score   support

             alt.atheism       0.90      0.78      0.84       236
           comp.graphics       0.69      0.80      0.74       287
 comp.os.ms-windows.misc       0.74      0.84      0.79       290
comp.sys.ibm.pc.hardware       0.69      0.69      0.69       285
   comp.sys.mac.hardware       0.88      0.81      0.85       312
          comp.windows.x       0.85      0.80      0.82       308
            misc.forsale       0.74      0.80      0.77       276
               rec.autos       0.89      0.87      0.88       304
         rec.motorcycles       0.93      0.94      0.94       279
      rec.sport.baseball       0.90      0.94      0.92       308
        rec.sport.hockey       0.91      0.94      0.93       309
               sci.crypt       0.95      0.92      0.94       290
         sci.electronics       0.83      0.72      0.77       304
                 sci.med       0.86      0.89      0.88       300
               sci.space       0.89      0.93      0.91       297
  soc.religion.christian       0.73      0.97      0.84       292
      talk.politics.guns       0.82      0.90      0.86       270
   talk.politics.mideast       0.96      0.94      0.95       272
      talk.politics.misc       0.92      0.72      0.81       239
      talk.religion.misc       0.92      0.46      0.61       196

                accuracy                           0.84      5654
               macro avg       0.85      0.83      0.84      5654
            weighted avg       0.85      0.84      0.84      5654


Process finished with exit code 0
```

Logistic Regression Results:

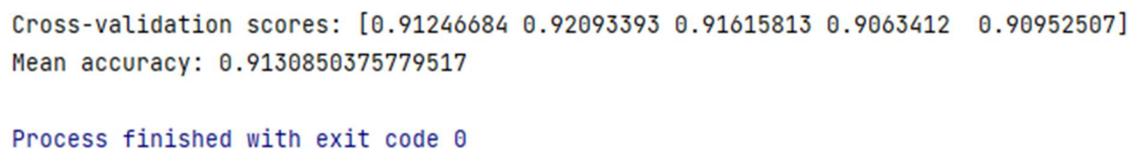


```
Fitting
done
predicting
done
Accuracy: 0.8438273788468341
                          precision    recall  f1-score   support

             alt.atheism       0.88      0.78      0.83       236
           comp.graphics       0.71      0.82      0.76       287
 comp.os.ms-windows.misc       0.74      0.87      0.80       290
comp.sys.ibm.pc.hardware       0.66      0.69      0.68       285
   comp.sys.mac.hardware       0.86      0.80      0.83       312
          comp.windows.x       0.88      0.81      0.85       308
            misc.forsale       0.72      0.80      0.76       276
               rec.autos       0.90      0.87      0.88       304
         rec.motorcycles       0.91      0.93      0.92       279
      rec.sport.baseball       0.92      0.94      0.93       308
        rec.sport.hockey       0.93      0.96      0.94       309
               sci.crypt       0.94      0.93      0.94       290
         sci.electronics       0.84      0.69      0.76       304
                 sci.med       0.88      0.89      0.89       300
               sci.space       0.90      0.94      0.92       297
  soc.religion.christian       0.72      0.96      0.82       292
      talk.politics.guns       0.83      0.93      0.88       270
   talk.politics.mideast       0.97      0.95      0.96       272
      talk.politics.misc       0.96      0.72      0.82       239
      talk.religion.misc       0.88      0.43      0.58       196

                accuracy                           0.84      5654
               macro avg       0.85      0.84      0.84      5654
            weighted avg       0.85      0.84      0.84      5654


Process finished with exit code 0
```

Naïve Bayes Alpha vs Accuracy Graph:



Naïve Bayes cross validation score:



```
Cross-validation scores: [0.91246684 0.92093393 0.91615813 0.9063412  0.90952507]
Mean accuracy: 0.9130850375779517

Process finished with exit code 0
```

Cross Validation score same as testing scores obtained previously. Hence model is not overfit.

Conclusions made from the results:

- The order of computational complexity, i.e., training time from lowest to highest is as follows:
    - Naïve Bayes
    - Logistic Regression
    - Random Forest Classifier
    - SVM

- The order of accuracy that I have achieved from highest to lowest is as follows:
    - SVM
    - Naïve Bayes
    - Logistic Regression
    - Random Forest Classifier (almost same as LR)
- Therefore, the best model to use for text classification is SVM. However, if you prefer shorter computational times while training the model, use Naïve Bayes.
- Here is a table showing the accuracies and Computational time ranks of each model that has been used for our Text classification problem.

	Accuracy	**Computational Efficiency**
Naïve Bayes	0.9108	1
SVM	0.9299	4
Random Forest	0.8412	3
Logistic Regression	0.8438	2

How good was the hyperparameter tuning? Have any of the models overfitted?

Do note that the hyperparameter tuning of the models have been done to the best of my ability. The order may vary in practice. Also note that I have gone for the most balanced accuracy, i.e., an accuracy with the least amount of overfitting. Have been able to achieve higher training accuracy but it was overfitted and hence not selected. None of the above models have been overfit. The cross_val_score method has been used for each of them once and the cross validation accuracy and testing accuracy was around the same, hence not overfit. I have not displayed the cross_val_score outputs for all the models as it is computationally expensive. However, it has been checked for every single one. Have displayed cross_val_score with Naïve Bayes and mean cross validation score and prediction scores were similar which shows there was no overfitting.