IS567 - Text Mining
Assignment 3

Task 1: Train and evaluate a unigram-based baseline classifier

Task 1: Train and evaluate a unigram-based baseline classifier

```
chatgpt_train = pd.read_csv("chatgpt_train.csv")
chatgpt_train.head()
```

	date	title	review	rating
0	5/21/2023 16:42	Much more accessible for blind users than the web version	Up to this point I?€?ve mostly been using ChatGPT on my windows desktop using Google Chrome. While it?€?s doable, screen reader navigation is pretty difficult on the desktop site and you really have to be an advanced user to find your way through it. I have submitted numerous feedbacks to open AI about this but nothing has changed on that front.\nWell, the good news ?€? the iOS app pretty much addresses all of those problems. The UI seems really clean, uncluttered and designed well to be compatible with voiceover, the screen reader built into iOS. I applaud the inclusivity of this design ?€? I only wish they would give the same attention and care to the accessibility experience of the desktop app.\nI would have given this review five stars but I have just a couple minor quibbles. First, once I submit my prompt, voiceover starts to read aloud ChatGPT?€?s response before that response is finished, so I will hear the first few words of the response followed by voiceover reading aloud the ?€?stop generating?€? button, which isn?€?t super helpful. It would be great if you could better coordinate this alert so that it didn?€?t start reading the message until it had been fully generated. The other thing I?€?d like is a Feedback button easily accessible from within the main screen of the app, to make it as easy as possible to get continuing suggestions and feedback from your users.\nOtherwise, fantastic app so far!	4
1	7/11/2023 12:24	Much anticipated, wasn?€?t let down.	I?€?ve been a user since it?€?s initial roll out and have been waiting for a mobile application ever since using the web app. For reference I?€?m a software engineering student while working in IT full time. I have to say GPT is an crucial tool. It takes far less time to get information quickly that you?€?d otherwise have to source from stack-overflow, various red-hat articles, Ubuntu articles, searching through software documentation, Microsoft documentation ect. Typically chat gpt can find the answer in a fraction of a second that google can. Obviously it is wrong, a lot. But to have the ability to get quick information on my phone like I can in the web browser I?€?m super excited about and have already been using the mobile app since download constantly. And I?€?m excited for the future of this program becoming more accurate and it seems to be getting more and more precise with every roll out. Gone are the days scouring the internet for obscure pieces of information, chat gpt can find it for you with 2 or 3 prompts. I love this app and I?€?m so happy it?€?s on mobile now. The UI is also very sleek, easy to use. My only complaint with the interface is the history tab at the top right. I actually prefer the conversation tabs on the left in the web app but I understand it would make the app kind of clunky especially on mobile since the screen size is smaller. Anyway, awesome app 5 stars.	4

```
chatgpt_train.columns
```

```
Index(['date', 'title', 'review', 'rating'], dtype='object')
```

```
chatgpt_test = pd.read_csv("chatgpt_test.csv")
chatgpt_test.head()
```

	date	title	review	rating
0	5/19/2023 6:09	error unsupported country	cant login	2
1	5/19/2023 9:39	Hype junk	More harm than help.	1
2	5/19/2023 4:12	your gpt 4 is fake	Fix it	1
3	5/20/2023 3:01	Please impose IPadOS	We need IPadOS!!!	5
4	5/19/2023 20:49	Amazing	Great	5

Number of instances in the training dataset with blank reviews

```
len(chatgpt_train)
```

```
1834
```

Number of instances in the test dataset with blank reviews

```
len(chatgpt_test)
```

```
458
```

```
chatgpt_train = chatgpt_train[~chatgpt_train["review"].isnull()]
chatgpt_test = chatgpt_test[~chatgpt_test["review"].isnull()]
```

Number of instances in the training dataset without blank reviews

```
len(chatgpt_train)
```

```
1829
```

Number of instances in the training dataset without blank reviews

```
len(chatgpt_test)
```

```
458
```

```
def three_way_classes(rating):
    if rating == 1 or rating == 2:
        val = 'negative'
    elif rating == 4 or rating == 5:
        val = 'positive'
    else:
        val = 'neutral'

    return val
```

```
chatgpt_train.loc[:, ('3_way_class')] = chatgpt_train.loc[:, ('rating')].apply(three_way_classes)
chatgpt_train.head()
```

```
chatgpt_train.loc[:, ('3_way_class')] = chatgpt_train.loc[:, ('rating')].apply(three_way_classes)
chatgpt_train.head()
```

	date	title	review	rating	3_way_class
0	5/21/2023 16:42	Much more accessible for blind users than the web version	Up to this point I?€?ve mostly been using ChatGPT on my windows desktop using Google Chrome. While it?€?s doable, screen reader navigation is pretty difficult on the desktop site and you really have to be an advanced user to find your way through it. I have submitted numerous feedbacks to open AI about this but nothing has changed on that front.\nWell, the good news ?€? the iOS app pretty much addresses all of those problems. The UI seems really clean, uncluttered and designed well to be compatible with voiceover, the screen reader built into iOS. I applaud the inclusivity of this design ?€? I only wish they would give the same attention and care to the accessibility experience of the desktop app.\nI would have given this review five stars but I have just a couple minor quibbles. First, once I submit my prompt, voiceover starts to read aloud ChatGPT?€?s response before that response is finished, so I will hear the first few words of the response followed by voiceover reading aloud the ?€?stop generating?€? button, which isn?€?t super helpful. It would be great if you could better coordinate this alert so that it didn?€?t start reading the message until it had been fully generated. The other thing I?€?d like is a Feedback button easily accessible from within the main screen of the app, to make it as easy as possible to get continuing suggestions and feedback from your users.\nOtherwise, fantastic app so far!	4	positive
1	7/11/2023 12:24	Much anticipated, wasn?€?t let down.	I?€?ve been a user since it?€?s initial roll out and have been waiting for a mobile application ever since using the web app. For reference I?€?m a software engineering student while working in IT full time. I have to say GPT is an crucial tool. It takes far less time to get information quickly that you?€?d otherwise have to source from stack-overflow, various red-hat articles, Ubuntu articles, searching through software documentation, Microsoft documentation ect. Typically chat gpt can find the answer in a fraction of a second that google can. Obviously it is wrong, a lot. But to have the ability to get quick information on my phone like I can in the web browser I?€?m super excited about and have already been using the mobile app since download constantly. And I?€?m excited for the future of this program becoming more accurate and it seems to be getting more and more precise with every roll out. Gone are the days scouring the internet for obscure pieces of information, chat gpt can find it for you with 2 or 3 prompts. I love this app and I?€?m so happy it?€?s on mobile now. The UI is also very sleek, easy to use. My only complaint with the interface is the history tab at the top right. I actually prefer the conversation tabs on the left in the web app but I understand it would make the app kind of clunky especially on mobile since the screen size is smaller. Anyway, awesome app 5 stars.	4	positive

Extract unigram features

```
# use review for model
train_text = chatgpt_train["review"]
test_text = chatgpt_test["review"]

# set the unigram range
vectorizer = CountVectorizer(ngram_range = (1,1))

# create training data representation
train_data_count = vectorizer.fit_transform(train_text.values.astype('U'))
print("NUMBER OF FEATURES IN TRAINING DATASET WITH UNIGRAMS")
print(train_data_count.shape,"\n")

# create test data representation
test_data_count = vectorizer.transform(test_text.values.astype('U'))
print("NUMBER OF FEATURES IN TEST DATASET WITH UNIGRAMS")
print(test_data_count.shape,"\n")
```

```
NUMBER OF FEATURES IN TRAINING DATASET WITH UNIGRAMS
(1829, 5551)

NUMBER OF FEATURES IN TEST DATASET WITH UNIGRAMS
(458, 5551)
```

```
# define true labels from train set

X_train = train_data_count
y_train = chatgpt_train["3_way_class"]
X_test = test_data_count
y_test = chatgpt_test["3_way_class"]
```

```
unigram_model = MultinomialNB()
unigram_model.fit(X_train, y_train)
```

```
MultinomialNB()
```

```
print ("Overall Accuracy score: ", accuracy_score(y_test, predictions), "\n")
print ("Overall Macro Recall score: ", recall_score(y_test, predictions, average='macro'))
print ("Overall Macro Precision score: ", precision_score(y_test, predictions, average='macro'))
print ("Overall Macro F1 score: ", f1_score(y_test, predictions, average='macro'), "\n")
print ("Overall Micro Recall score: ", recall_score(y_test, predictions, average='micro'))
print ("Overall Micro Precision score: ", precision_score(y_test, predictions, average='micro'))
print ("Overall Micro F1 score: ", f1_score(y_test, predictions, average='micro'), "\n")
print ("Overall Weighted Recall score: ", recall_score(y_test, predictions, average='weighted'))
print ("Overall Weighted Precision score: ", precision_score(y_test, predictions, average='weighted'))
print ("Overall Weighted F1 score: ", f1_score(y_test, predictions, average='weighted'), "\n")
print ("Individual label performance: ")
print (classification_report(y_test, predictions))
print ("Confusion Matrix: ")
print (confusion_matrix(y_test, predictions))
```

```
Overall Accuracy score:  0.7358078602620087

Overall Macro Recall score:  0.5007752841737103
Overall Macro Precision score:  0.5420844714259355
Overall Macro F1 score:  0.5116057233704292

Overall Micro Recall score:  0.7358078602620087
Overall Micro Precision score:  0.7358078602620087
Overall Micro F1 score:  0.7358078602620087

Overall Weighted Recall score:  0.7358078602620087
Overall Weighted Precision score:  0.7128098567692405
Overall Weighted F1 score:  0.7176894078769239

Individual label performance:
              precision    recall  f1-score   support

    negative       0.69      0.53      0.60       141
     neutral       0.17      0.08      0.11        25
    positive       0.77      0.89      0.83       292

    accuracy                           0.74       458
   macro avg       0.54      0.50      0.51       458
weighted avg       0.71      0.74      0.72       458

Confusion Matrix:
[[ 75   4  62]
 [  8   2  15]
 [ 26   6 260]]
```

Task 2: (feature selection 1): Remove features with low variance

A) Threshold = 0.001

Task 2: (feature selection 1): Remove features with low variance ¶

```
from sklearn.feature_selection import VarianceThreshold
```

```
low_threshold = 0.001
high_threshold = 0.005
```

```
feature_selector = VarianceThreshold(threshold = low_threshold)
```

```
# Low variance threshold = 0.001

feature_selector = VarianceThreshold(threshold = low_threshold)

X_train_low_variance_features_1 = feature_selector.fit(train_data_count).transform(train_data_count)
print ("Train feature space before low variance filtering (threshold=0.001): ", train_data_count.shape)
print ("Train feature space after low variance filtering (threshold=0.001): ", X_train_low_variance_features_1.shape)

print()

X_test_low_variance_features_1 = feature_selector.fit(train_data_count).transform(test_data_count)
print ("Test feature space before low variance filtering (threshold=0.001): ", test_data_count.shape)
print ("Test feature space after low variance filtering (threshold=0.001): ", X_test_low_variance_features_1.shape)
```

```
Train feature space before low variance filtering (threshold=0.001):  (1829, 5551)
Train feature space after low variance filtering (threshold=0.001):  (1829, 3054)

Test feature space before low variance filtering (threshold=0.001):  (458, 5551)
Test feature space after low variance filtering (threshold=0.001):  (458, 3054)
```

```
# define true labels from train set

X_train = X_train_low_variance_features_1
y_train = chatgpt_train["3_way_class"]
X_test = X_test_low_variance_features_1
y_test = chatgpt_test["3_way_class"]
```

```
low_variance_0001_model = MultinomialNB()
low_variance_0001_model.fit(X_train, y_train)
```

▼ MultinomialNB

MultinomialNB()

```
predictions = low_variance_0001_model.predict(X_test)
predictions[:5]
```

```
array(['negative', 'positive', 'negative', 'positive', 'positive'],
      dtype='<U8')
```

```
print ("Overall Accuracy score: ", accuracy_score(y_test, predictions), "\n")
print ("Overall Macro Recall score: ", recall_score(y_test, predictions, average='macro'))
print ("Overall Macro Precision score: ", precision_score(y_test, predictions, average='macro'))
print ("Overall Macro F1 score: ", f1_score(y_test, predictions, average='macro'), "\n")
print ("Overall Micro Recall score: ", recall_score(y_test, predictions, average='micro'))
print ("Overall Micro Precision score: ", precision_score(y_test, predictions, average='micro'))
print ("Overall Micro F1 score: ", f1_score(y_test, predictions, average='micro'), "\n")
print ("Overall Weighted Recall score: ", recall_score(y_test, predictions, average='weighted'))
print ("Overall Weighted Precision score: ", precision_score(y_test, predictions, average='weighted'))
print ("Overall Weighted F1 score: ", f1_score(y_test, predictions, average='weighted'), "\n")
print ("Individual label performance: ")
print (classification_report(y_test, predictions))
print ("Confusion Matrix: ")
print (confusion_matrix(y_test, predictions))
```

```
Overall Accuracy score:  0.7270742358078602

Overall Macro Recall score:  0.4974315878104861
Overall Macro Precision score:  0.5286054212610907
Overall Macro F1 score:  0.5065362940263409

Overall Micro Recall score:  0.7270742358078602
Overall Micro Precision score:  0.7270742358078602
Overall Micro F1 score:  0.7270742358078602

Overall Weighted Recall score:  0.7270742358078602
Overall Weighted Precision score:  0.7060217680304899
Overall Weighted F1 score:  0.7117444835328599

Individual label performance:
              precision    recall  f1-score   support

    negative       0.67      0.54      0.60       141
     neutral       0.14      0.08      0.10        25
    positive       0.77      0.87      0.82       292

    accuracy                           0.73       458
   macro avg       0.53      0.50      0.51       458
weighted avg       0.71      0.73      0.71       458

Confusion Matrix:
[[ 76   4  61]
 [  8   2  15]
 [ 29   8 255]]
```

B) Threshold = 0.005

```
# High variance threshold = 0.005

feature_selector = VarianceThreshold(threshold = high_threshold)

X_train_low_variance_features_5 = feature_selector.fit(train_data_count).transform(train_data_count)
print ("Train feature space before low variance filtering (threshold=0.005): ", train_data_count.shape)
print ("Train feature space after low variance filtering (threshold=0.005): ", X_train_low_variance_features_5.shape)

print()

X_test_low_variance_features_5 = feature_selector.fit(train_data_count).transform(test_data_count)
print ("Test feature space before low variance filtering (threshold=0.005): ", test_data_count.shape)
print ("Test feature space after low variance filtering (threshold=0.005): ", X_test_low_variance_features_5.shape)
```

```
Train feature space before low variance filtering (threshold=0.005):  (1829, 5551)
Train feature space after low variance filtering (threshold=0.005):  (1829, 980)

Test feature space before low variance filtering (threshold=0.005):  (458, 5551)
Test feature space after low variance filtering (threshold=0.005):  (458, 980)
```

```
# define true labels from train set

X_train = X_train_low_variance_features_5
y_train = chatgpt_train["3_way_class"]
X_test = X_test_low_variance_features_5
y_test = chatgpt_test["3_way_class"]
```

```
low_variance_0005_model = MultinomialNB()
low_variance_0005_model.fit(X_train, y_train)
```

▾ MultinomialNB

MultinomialNB()

```
predictions = low_variance_0005_model.predict(X_test)
predictions[:5]
```

```
array(['negative', 'positive', 'negative', 'positive', 'positive'],
      dtype='<U8')
```

```
print ("Overall Accuracy score: ", accuracy_score(y_test, predictions), "\n")
print ("Overall Macro Recall score: ", recall_score(y_test, predictions, average='macro'))
print ("Overall Macro Precision score: ", precision_score(y_test, predictions, average='macro'))
print ("Overall Macro F1 score: ", f1_score(y_test, predictions, average='macro'), "\n")
print ("Overall Micro Recall score: ", recall_score(y_test, predictions, average='micro'))
print ("Overall Micro Precision score: ", precision_score(y_test, predictions, average='micro'))
print ("Overall Micro F1 score: ", f1_score(y_test, predictions, average='micro'), "\n")
print ("Overall Weighted Recall score: ", recall_score(y_test, predictions, average='weighted'))
print ("Overall Weighted Precision score: ", precision_score(y_test, predictions, average='weighted'))
print ("Overall Weighted F1 score: ", f1_score(y_test, predictions, average='weighted'), "\n")
print ("Individual label performance: ")
print (classification_report(y_test, predictions))
print ("Confusion Matrix: ")
print (confusion_matrix(y_test, predictions))
```

```
Overall Accuracy score:  0.7358078602620087

Overall Macro Recall score:  0.4995527704912724
Overall Macro Precision score:  0.5362426035502958
Overall Macro F1 score:  0.5100713456114153

Overall Micro Recall score:  0.7358078602620087
Overall Micro Precision score:  0.7358078602620087
Overall Micro F1 score:  0.7358078602620087

Overall Weighted Recall score:  0.7358078602620087
Overall Weighted Precision score:  0.7181904214361385
Overall Weighted F1 score:  0.7195576238803115

Individual label performance:
              precision    recall  f1-score   support

    negative       0.71      0.52      0.60       141
     neutral       0.12      0.08      0.10        25
    positive       0.77      0.89      0.83       292

    accuracy                           0.74       458
   macro avg       0.54      0.50      0.51       458
weighted avg       0.72      0.74      0.72       458

Confusion Matrix:
[[ 74   4  63]
 [  9   2  14]
 [ 21  10 261]]
```

In this experiment, using the threshold of 0.001 gives us 3054 features, whereas the threshold of 0.005 gives us 980 features.

The model with threshold of 0.005 gives us fewer features and has shown an accuracy of 73.58% whereas the threshold of 0.001 gives us an accuracy of 72.7%.

The model with threshold of 0.005 has better accuracy and F1 score.

There may be reasons for this:

- When we use a higher threshold value for the filtering, the features with lower variance are excluded and hence we obtain less features reducing the feature subspace as compared to the low threshold values. Due to this, the model learns better.
- With lower features, the model may complexity is reduced and it avoids the chances of overfitting. The model may be able to generalise better.

Task 3: Task 3 (feature selection 2): Select top k-best features using information gain (mutual information)

A) k = 1000

```
from sklearn.feature_selection import SelectKBest, mutual_info_classif
```

```
# k = 1000

selector = SelectKBest(mutual_info_classif, k=1000)
X_train_mutual_info_features_1000 = selector.fit_transform(train_data_count, y_train)
print ("Train feature space before filtering with 1000 best features: ", train_data_count.shape)
print ("Train feature space after filtering with 1000 best features: ", X_train_mutual_info_features_1000.shape)

print()

X_test_mutual_info_features_1000 = selector.transform(test_data_count)
print ("Test feature space before filtering with 1000 best features: ", test_data_count.shape)
print ("Test feature space after filtering with 1000 best features: ", X_test_mutual_info_features_1000.shape)
```

```
Train feature space before filtering with 1000 best features:  (1829, 5551)
Train feature space after filtering with 1000 best features:  (1829, 1000)

Test feature space before filtering with 1000 best features:  (458, 5551)
Test feature space after filtering with 1000 best features:  (458, 1000)
```

```
# define true labels from train set

X_train = X_train_mutual_info_features_1000
y_train = chatgpt_train["3_way_class"]
X_test = X_test_mutual_info_features_1000
y_test = chatgpt_test["3_way_class"]
```

```
best_features_1000_model = MultinomialNB()
best_features_1000_model.fit(X_train, y_train)
```

▾ MultinomialNB

MultinomialNB()

```
predictions = best_features_1000_model.predict(X_test_mutual_info_features_1000)
predictions[:5]
```

```
array(['negative', 'positive', 'negative', 'positive', 'positive'],
      dtype='<U8')
```

```
print ("Overall Accuracy score: ", accuracy_score(y_test, predictions), "\n")
print ("Overall Macro Recall score: ", recall_score(y_test, predictions, average='macro'))
print ("Overall Macro Precision score: ", precision_score(y_test, predictions, average='macro'))
print ("Overall Macro F1 score: ", f1_score(y_test, predictions, average='macro'), "\n")
print ("Overall Micro Recall score: ", recall_score(y_test, predictions, average='micro'))
print ("Overall Micro Precision score: ", precision_score(y_test, predictions, average='micro'))
print ("Overall Micro F1 score: ", f1_score(y_test, predictions, average='micro'), "\n")
print ("Overall Weighted Recall score: ", recall_score(y_test, predictions, average='weighted'))
print ("Overall Weighted Precision score: ", precision_score(y_test, predictions, average='weighted'))
print ("Overall Weighted F1 score: ", f1_score(y_test, predictions, average='weighted'), "\n")
print ("Individual label performance: ")
print (classification_report(y_test, predictions))
print ("Confusion Matrix: ")
print (confusion_matrix(y_test, predictions))
```

```
Overall Accuracy score:  0.740174672489083

Overall Macro Recall score:  0.5030583891965413
Overall Macro Precision score:  0.5389112618341105
Overall Macro F1 score:  0.5134618118836196

Overall Micro Recall score:  0.740174672489083
Overall Micro Precision score:  0.740174672489083
Overall Micro F1 score:  0.740174672489083

Overall Weighted Recall score:  0.740174672489083
Overall Weighted Precision score:  0.7223889137762837
Overall Weighted F1 score:  0.7241716069662414

Individual label performance:
              precision    recall  f1-score   support

    negative       0.71      0.53      0.61       141
     neutral       0.12      0.08      0.10        25
    positive       0.78      0.90      0.83       292

    accuracy                           0.74       458
   macro avg       0.54      0.50      0.51       458
weighted avg       0.72      0.74      0.72       458

Confusion Matrix:
[[ 75   5  61]
 [  9   2  14]
 [ 21   9 262]]
```

B) k = 2000

```
# k = 2000

selector = SelectKBest(mutual_info_classif, k=2000)
X_train_mutual_info_features_2000 = selector.fit_transform(train_data_count, y_train)
print ("Train feature space before filtering with 2000 best features: ", train_data_count.shape)
print ("Train feature space after filtering with 2000 best features: ", X_train_mutual_info_features_2000.shape)

print()

X_test_mutual_info_features_2000 = selector.transform(test_data_count)
print ("Test feature space before filtering with 2000 best features: ", test_data_count.shape)
print ("Test feature space after filtering with 2000 best features: ", X_test_mutual_info_features_2000.shape)
```

```
Train feature space before filtering with 2000 best features:  (1829, 5551)
Train feature space after filtering with 2000 best features:  (1829, 2000)

Test feature space before filtering with 2000 best features:  (458, 5551)
Test feature space after filtering with 2000 best features:  (458, 2000)
```

```
# define true labels from train set

X_train = X_train_mutual_info_features_2000
y_train = chatgpt_train["3_way_class"]
X_test = X_test_mutual_info_features_2000
y_test = chatgpt_test["3_way_class"]
```

```
best_features_2000_model = MultinomialNB()
best_features_2000_model.fit(X_train, y_train)
```

▾ MultinomialNB

MultinomialNB()

```
predictions = best_features_2000_model.predict(X_test_mutual_info_features_2000)
predictions[:5]
```

```
array(['negative', 'positive', 'negative', 'positive', 'positive'],
      dtype='<U8')
```

```
print ("Overall Accuracy score: ", accuracy_score(y_test, predictions), "\n")
print ("Overall Macro Recall score: ", recall_score(y_test, predictions, average='macro'))
print ("Overall Macro Precision score: ", precision_score(y_test, predictions, average='macro'))
print ("Overall Macro F1 score: ", f1_score(y_test, predictions, average='macro'), "\n")
print ("Overall Micro Recall score: ", recall_score(y_test, predictions, average='micro'))
print ("Overall Micro Precision score: ", precision_score(y_test, predictions, average='micro'))
print ("Overall Micro F1 score: ", f1_score(y_test, predictions, average='micro'), "\n")
print ("Overall Weighted Recall score: ", recall_score(y_test, predictions, average='weighted'))
print ("Overall Weighted Precision score: ", precision_score(y_test, predictions, average='weighted'))
print ("Overall Weighted F1 score: ", f1_score(y_test, predictions, average='weighted'), "\n")
print ("Individual label performance: ")
print (classification_report(y_test, predictions))
print ("Confusion Matrix: ")
print (confusion_matrix(y_test, predictions))
```

```
Overall Accuracy score:  0.7379912663755459

Overall Macro Recall score:  0.49094756954564595
Overall Macro Precision score:  0.5183691642643176
Overall Macro F1 score:  0.4976540194190983

Overall Micro Recall score:  0.7379912663755459
Overall Micro Precision score:  0.7379912663755459
Overall Micro F1 score:  0.7379912663755459

Overall Weighted Recall score:  0.7379912663755459
Overall Weighted Precision score:  0.7146149936633656
Overall Weighted F1 score:  0.7199030187879334

Individual label performance:
              precision    recall  f1-score   support

    negative       0.70      0.54      0.61       141
     neutral       0.08      0.04      0.05        25
    positive       0.77      0.89      0.83       292

    accuracy                           0.74       458
   macro avg       0.52      0.49      0.50       458
weighted avg       0.71      0.74      0.72       458

Confusion Matrix:
[[ 76   5  60]
 [  8   1  16]
 [ 24   7 261]]
```

In this experiment, using the k=1000 gives us 1000 best features, whereas using the k=2000 gives us 2000 best features.

The model with using the k=1000 shows an accuracy of 74.01% whereas the k=2000 gives us an accuracy of 73.79%.

The model with k=1000 has better accuracy and F1 score.

There may be reasons for this:

- When we use a lower ‘k’ value for the filtering of best features, the features that have the highest mutual information are only retained and hence we obtain less features reducing the feature subspace as compared to the low threshold values. Due to this, the model learns better.
- With lower features, the model may complexity is reduced and it avoids the chances of overfitting. The model may be able to generalise better.

Task 4: Task 4: (feature selection 3): Lexicon-based feature selection

```
f = open("positive_words_list.txt", "r")
positive_contents = f.read()

positive_words = positive_contents.split("\n")
positive_words
```

```
['a+',
 'abound',
 'abounds',
 'abundance',
 'abundant',
 'accessable',
 'accessible',
 'acclaim',
 'acclaimed',
 'acclamation',
 'accolade',
 'accolades',
 'accommodative',
 'accomodative',
 'accomplish',
 'accomplished',
 'accomplishment',
 'accomplishments',
 'accurate',
```

```
f = open("negative_words_list.txt", "r")
negative_contents = f.read()

negative_words = negative_contents.split("\n")
negative_words
```

```
['2-faced',
 '2-faces',
 'abnormal',
 'abolish',
 'abominable',
 'abominably',
 'abominate',
 'abomination',
 'abort',
 'aborted',
 'aborts',
 'abrade',
```

```
# Function to extract lexicon features

def extract_lexicon_features(text):
    words = set(text.split())
    positive = set([word for word in words if word in positive_words])
    negative = set([word for word in words if word in negative_words])
    joined = positive.union(negative)
    return " ".join(joined)
```

```
chatgpt_train['lexicon_features'] = chatgpt_train['review'].apply(extract_lexicon_features)
chatgpt_test['lexicon_features'] = chatgpt_test['review'].apply(extract_lexicon_features)
```

```
# define true labels from train set

X_train = chatgpt_train['lexicon_features']
y_train = chatgpt_train["3_way_class"]
X_test = chatgpt_test['lexicon_features']
y_test = chatgpt_test["3_way_class"]
```

```
# Using CountVectorizer
vectorizer = CountVectorizer(ngram_range = (1,1))

# create training data representation
X_train_lexicon_features_count_vec = vectorizer.fit_transform(X_train.values.astype('U'))
print("Number of features in training dataset with lexicons")
print(X_train_lexicon_features_count_vec.shape,"\n")

# create test data representation
X_test_data_count_vec = vectorizer.transform(X_test.values.astype('U'))
print("Number of features in test dataset with lexicons")
print(test_data_count.shape,"\n")
```

```
Number of features in training dataset with lexicons
(1829, 915)

Number of features in test dataset with lexicons
(458, 915)
```

```
lexicon_based_model = MultinomialNB()
lexicon_based_model.fit(X_train_lexicon_features_count_vec, y_train)
```

▾ MultinomialNB

MultinomialNB()

```
predictions = lexicon_based_model.predict(X_test_data_count_vec)
predictions[:5]
```

```
array(['positive', 'positive', 'positive', 'positive', 'positive'],
      dtype='<U8')
```

```
print ("Overall Accuracy score: ", accuracy_score(y_test, predictions), "\n")
print ("Overall Macro Recall score: ", recall_score(y_test, predictions, average='macro'))
print ("Overall Macro Precision score: ", precision_score(y_test, predictions, average='macro'))
print ("Overall Macro F1 score: ", f1_score(y_test, predictions, average='macro'), "\n")
print ("Overall Micro Recall score: ", recall_score(y_test, predictions, average='micro'))
print ("Overall Micro Precision score: ", precision_score(y_test, predictions, average='micro'))
print ("Overall Micro F1 score: ", f1_score(y_test, predictions, average='micro'), "\n")
print ("Overall Weighted Recall score: ", recall_score(y_test, predictions, average='weighted'))
print ("Overall Weighted Precision score: ", precision_score(y_test, predictions, average='weighted'))
print ("Overall Weighted F1 score: ", f1_score(y_test, predictions, average='weighted'), "\n")
print ("Individual label performance: ")
print (classification_report(y_test, predictions))
print ("Confusion Matrix: ")
print (confusion_matrix(y_test, predictions))
```

```
Overall Accuracy score:  0.6615720524017468

Overall Macro Recall score:  0.36422811619547263
Overall Macro Precision score:  0.44745621351125936
Overall Macro F1 score:  0.3250859569877975

Overall Micro Recall score:  0.6615720524017468
Overall Micro Precision score:  0.6615720524017468
Overall Micro F1 score:  0.6615720524017468

Overall Weighted Recall score:  0.6615720524017468
Overall Weighted Precision score:  0.631041697775803
Overall Weighted F1 score:  0.5611001905641175

Individual label performance:
              precision    recall  f1-score   support

    negative       0.68      0.11      0.18       141
     neutral       0.00      0.00      0.00        25
    positive       0.66      0.99      0.79       292

    accuracy                           0.66       458
   macro avg       0.45      0.36      0.33       458
weighted avg       0.63      0.66      0.56       458

Confusion Matrix:
[[ 15   0 126]
 [  3   0  22]
 [  4   0 288]]
```

In this experiment, using the lexicon features:

I have used the approach of first just selecting the words from the review that are present in the lexicon lists and then combining them. Based on this, I then extract the unigram features.

The model shows an accuracy of 66.15% and the weighted F1 score is 0.5611.

This model does not perform as good as the other feature selection techniques. Even though it successfully helps in reducing the dimensionality of the feature space, the overall performance of the model is very low. We also observe that there are no samples that have been classified as neutral and the F1 score for neutral class is 0 because both the precision and recall values are 0 for the neutral class. Hence this is acting just like a binary classifier and is worse.

The reasons may be that the lexicons may be domain specific or they include words that are rare. When we filter out using lexicons, there is a high probability that we lose uncommon words and hence this results in a drop in the accuracy, F1 score and the overall performance.

Comparison of feature selection techniques.

Model	Feature Space	Micro			Macro			Weighted		
		Precision	Recall	F1 Score	Precision	Recall	F1 Score	Precision	Recall	F1 Score
Baseline (Naive Bayes)	5551	0.7358	0.7358	0.7358	0.5421	0.5008	0.5116	0.7128	0.7358	0.7172
Low variance (threshold = 0.001)	3054	0.7270	0.7270	0.7270	0.5286	0.4974	0.5065	0.7060	0.7270	0.7117
Low variance (threshold = 0.005)	980	0.7358	0.7358	0.7358	0.5362	0.4996	0.5101	0.7182	0.7358	0.7196
K-best using mutual information with k=1000	1000	0.7402	0.7402	0.7402	0.5389	0.5031	0.5135	0.7224	0.7402	0.7242
K-best using mutual information with k=1000	2000	0.7379	0.7379	0.7379	0.5184	0.4909	0.4977	0.7146	0.7379	0.7199
Lexicon-based feature selection	915	0.6616	0.6616	0.6616	0.4475	0.3642	0.325	0.631	0.6616	0.5611

Task 5: Discussion on feature selection

Strengths and weaknesses of the feature selection techniques:

A) Variance Threshold Feature Selection:

Pros:

The technique is simple to straightforward to implement and easy to understand. It is computationally efficient as we just need to calculate the variance of the features and exclude features that are below the threshold value. Hence, this is suitable for large datasets. It effectively removes features with low variance, reducing dimensionality and the chances of overfitting.

Cons:

From my experience, firstly since variance is involved it deals with numbers and hence the textual representations need to be converted to numeric form. Secondly, it does not take into consideration text semantics and may sometimes remove rare words that may have low variance but may be significant to the domain.

B) K-Best Feature Selection:

Pros:

The technique is simple and scalable. Hence, this is suitable for large datasets. It provides the different criteria or scoring techniques like mutual information, chi-square to select the best features dependent on domain. It helps identify and retain the most relevant features for the predictive model.

Cons:

Since there are multiple scoring options available, the choice may not be suitable for the applications. Also, the technique may miss features that are collectively informative as it treats the features as independent.

C) Lexicon-based Feature Selection:

Pros:

The technique is suitable when the tasks are domain specific as it the lexicons can be defined that are relevant to the domain. It allows for interpretability as we can customise the lexicons and adds selection easy to justify.

Cons:

The list of lexicons for the tasks needs to be available and may require manual effort for text annotations. Sometimes, words that are not domain specific may be missed and common words may be ignored. As a result, the model may have poor accuracy and only suitable for the domain.

To improve the performance, we can try the ensemble technique where we can generate the features using each of the techniques. At the end, we can combine the set of features obtained by all the techniques by performing union operations for the unique features to get a coherent and rich set of features.

Task 6: Extract and select best unigrams

```
# use review for model
train_text = chatgpt_train["review"]
test_text = chatgpt_test["review"]

# set the unigram range
vectorizer = CountVectorizer(ngram_range = (1,1))

# create training data representation
train_data_count = vectorizer.fit_transform(train_text.values.astype('U'))
print("NUMBER OF FEATURES IN TRAINING DATASET WITH UNIGRAMS")
print(train_data_count.shape,"\n")

# create test data representation
test_data_count = vectorizer.transform(test_text.values.astype('U'))
print("NUMBER OF FEATURES IN TEST DATASET WITH UNIGRAMS")
print(test_data_count.shape,"\n")
```

```
NUMBER OF FEATURES IN TRAINING DATASET WITH UNIGRAMS
(1829, 5551)

NUMBER OF FEATURES IN TEST DATASET WITH UNIGRAMS
(458, 5551)
```

```
# k = 2000

selector = SelectKBest(mutual_info_classif, k=2000)

X_train_mutual_info_best_2000_features = selector.fit_transform(train_data_count, y_train)
print ("Train feature space before filtering with 2000 best features: ", train_data_count.shape)
print ("Train feature space after filtering with 2000 best features: ", X_train_mutual_info_best_2000_features.shape)

print()

X_test_mutual_info_best_2000_features = selector.transform(test_data_count)
print ("Test feature space before filtering with 2000 best features: ", test_data_count.shape)
print ("Test feature space after filtering with 2000 best features: ", X_test_mutual_info_best_2000_features.shape)
```

```
Train feature space before filtering with 2000 best features:  (1829, 5551)
Train feature space after filtering with 2000 best features:  (1829, 2000)

Test feature space before filtering with 2000 best features:  (458, 5551)
Test feature space after filtering with 2000 best features:  (458, 2000)
```

Task 7: Train and evaluate a Naive Bayes classifier using cross-validation

```
from sklearn.model_selection import StratifiedKFold, cross_val_score
```

```
cross_validation = StratifiedKFold(n_splits=5)
cross_validation
```

```
StratifiedKFold(n_splits=5, random_state=None, shuffle=False)
```

```
# get data columns

y_label = chatgpt_train['3_way_class']
y_label = np.array(y_label)

print("Train feature space with 2000 best features: ", X_train_mutual_info_best_2000_features.shape)
print("Test labels shape: ", y_label.shape)
```

```
Train feature space with 2000 best features:  (1829, 2000)
Test labels shape:  (1829,)
```

```
# counter i
i = 0
f1_nb = []

for train_index, test_index in cross_validation.split(X_train_mutual_info_best_2000_features, y_label):

    print(f"CV with Multinomial Naive Bayes: {i+1}")

    print(train_index)

    X_train = X_train_mutual_info_best_2000_features[train_index]
    X_test = X_train_mutual_info_best_2000_features[test_index]
    y_train = y_label[train_index]
    y_test = y_label[test_index]

    multinomial_naive_bayes = MultinomialNB()

    multinomial_naive_bayes.fit(X_train, y_train)

    # to see all the hyper parameters
    print()
    # print(pipeline.get_params())

    predictions = multinomial_naive_bayes.predict(X_test)

    print ("Overall Accuracy score: ", accuracy_score(y_test, predictions))
    print ("Overall Weighted Recall score: ", recall_score(y_test, predictions, average='weighted'))
    print ("Overall Weighted Precision score: ", precision_score(y_test, predictions, average='weighted'))
    print ("Overall Weighted F1 score: ", f1_score(y_test, predictions, average='weighted'), "\n")
    f1_nb.append(f1_score(y_test, predictions, average='weighted'))
    print ("................................................\n\n")
    i += 1
```

Output of the cross validation splits:

```
CV with Multinomial Naive Bayes: 1

Overall Accuracy score:  0.7622950819672131
Overall Weighted Recall score:  0.7622950819672131
Overall Weighted Precision score:  0.7463243649496859
Overall Weighted F1 score:  0.7353849453441034

..............................................


CV with Multinomial Naive Bayes: 2

Overall Accuracy score:  0.7404371584699454
Overall Weighted Recall score:  0.7404371584699454
Overall Weighted Precision score:  0.7478418553213636
Overall Weighted F1 score:  0.7316207081945378

..............................................


CV with Multinomial Naive Bayes: 3

Overall Accuracy score:  0.7404371584699454
Overall Weighted Recall score:  0.7404371584699454
Overall Weighted Precision score:  0.7522655202678737
Overall Weighted F1 score:  0.7344702865676813

..............................................



CV with Multinomial Naive Bayes: 4

Overall Accuracy score:  0.7650273224043715
Overall Weighted Recall score:  0.7650273224043715
Overall Weighted Precision score:  0.7761611402021165
Overall Weighted F1 score:  0.7573191566719791

..............................................


CV with Multinomial Naive Bayes: 5

Overall Accuracy score:  0.8
Overall Weighted Recall score:  0.8
Overall Weighted Precision score:  0.7448149958090665
Overall Weighted F1 score:  0.7603869670326174

..............................................
```

CV Split - 5 Training Dataset Indexes:

```
CV with Multinomial Naive Bayes: 5
[   0    1    2    3    4    5    6    7    8    9   10   11   12   13
   14   15   16   17   18   19   20   21   22   23   24   25   26   27
   28   29   30   31   32   33   34   35   36   37   38   39   40   41
   42   43   44   45   46   47   48   49   50   51   52   53   54   55
   56   57   58   59   60   61   62   63   64   65   66   67   68   69
   70   71   72   73   74   75   76   77   78   79   80   81   82   83
   84   85   86   87   88   89   90   91   92   93   94   95   96   97
   98   99  100  101  102  103  104  105  106  107  108  109  110  111
  112  113  114  115  116  117  118  119  120  121  122  123  124  125
  126  127  128  129  130  131  132  133  134  135  136  137  138  139
  140  141  142  143  144  145  146  147  148  149  150  151  152  153
  154  155  156  157  158  159  160  161  162  163  164  165  166  167
  168  169  170  171  172  173  174  175  176  177  178  179  180  181
  182  183  184  185  186  187  188  189  190  191  192  193  194  195
```

196 197 198 199 200 201 202 203 204 205 206 207 208 209
210 211 212 213 214 215 216 217 218 219 220 221 222 223
224 225 226 227 228 229 230 231 232 233 234 235 236 237
238 239 240 241 242 243 244 245 246 247 248 249 250 251
252 253 254 255 256 257 258 259 260 261 262 263 264 265
266 267 268 269 270 271 272 273 274 275 276 277 278 279
280 281 282 283 284 285 286 287 288 289 290 291 292 293
294 295 296 297 298 299 300 301 302 303 304 305 306 307
308 309 310 311 312 313 314 315 316 317 318 319 320 321
322 323 324 325 326 327 328 329 330 331 332 333 334 335
336 337 338 339 340 341 342 343 344 345 346 347 348 349
350 351 352 353 354 355 356 357 358 359 360 361 362 363
364 365 366 367 368 369 370 371 372 373 374 375 376 377
378 379 380 381 382 383 384 385 386 387 388 389 390 391
392 393 394 395 396 397 398 399 400 401 402 403 404 405
406 407 408 409 410 411 412 413 414 415 416 417 418 419
420 421 422 423 424 425 426 427 428 429 430 431 432 433
434 435 436 437 438 439 440 441 442 443 444 445 446 447
448 449 450 451 452 453 454 455 456 457 458 459 460 461
462 463 464 465 466 467 468 469 470 471 472 473 474 475
476 477 478 479 480 481 482 483 484 485 486 487 488 489
490 491 492 493 494 495 496 497 498 499 500 501 502 503
504 505 506 507 508 509 510 511 512 513 514 515 516 517
518 519 520 521 522 523 524 525 526 527 528 529 530 531
532 533 534 535 536 537 538 539 540 541 542 543 544 545
546 547 548 549 550 551 552 553 554 555 556 557 558 559
560 561 562 563 564 565 566 567 568 569 570 571 572 573
574 575 576 577 578 579 580 581 582 583 584 585 586 587
588 589 590 591 592 593 594 595 596 597 598 599 600 601
602 603 604 605 606 607 608 609 610 611 612 613 614 615
616 617 618 619 620 621 622 623 624 625 626 627 628 629
630 631 632 633 634 635 636 637 638 639 640 641 642 643
644 645 646 647 648 649 650 651 652 653 654 655 656 657
658 659 660 661 662 663 664 665 666 667 668 669 670 671
672 673 674 675 676 677 678 679 680 681 682 683 684 685
686 687 688 689 690 691 692 693 694 695 696 697 698 699
700 701 702 703 704 705 706 707 708 709 710 711 712 713
714 715 716 717 718 719 720 721 722 723 724 725 726 727
728 729 730 731 732 733 734 735 736 737 738 739 740 741
742 743 744 745 746 747 748 749 750 751 752 753 754 755
756 757 758 759 760 761 762 763 764 765 766 767 768 769
770 771 772 773 774 775 776 777 778 779 780 781 782 783
784 785 786 787 788 789 790 791 792 793 794 795 796 797
798 799 800 801 802 803 804 805 806 807 808 809 810 811
812 813 814 815 816 817 818 819 820 821 822 823 824 825
826 827 828 829 830 831 832 833 834 835 836 837 838 839
840 841 842 843 844 845 846 847 848 849 850 851 852 853
854 855 856 857 858 859 860 861 862 863 864 865 866 867
868 869 870 871 872 873 874 875 876 877 878 879 880 881
882 883 884 885 886 887 888 889 890 891 892 893 894 895
896 897 898 899 900 901 902 903 904 905 906 907 908 909
910 911 912 913 914 915 916 917 918 919 920 921 922 923
924 925 926 927 928 929 930 931 932 933 934 935 936 937
938 939 940 941 942 943 944 945 946 947 948 949 950 951
952 953 954 955 956 957 958 959 960 961 962 963 964 965
966 967 968 969 970 971 972 973 974 975 976 977 978 979
980 981 982 983 984 985 986 987 988 989 990 991 992 993
994 995 996 997 998 999 1000 1001 1002 1003 1004 1005 1006 1007
1008 1009 1010 1011 1012 1013 1014 1015 1016 1017 1018 1019 1020 1021
1022 1023 1024 1025 1026 1027 1028 1029 1030 1031 1032 1033 1034 1035
1036 1037 1038 1039 1040 1041 1042 1043 1044 1045 1046 1047 1048 1049
1050 1051 1052 1053 1054 1055 1056 1057 1058 1059 1060 1061 1062 1063
1064 1065 1066 1067 1068 1069 1070 1071 1072 1073 1074 1075 1076 1077
1078 1079 1080 1081 1082 1083 1084 1085 1086 1087 1088 1089 1090 1091
1092 1093 1094 1095 1096 1097 1098 1099 1100 1101 1102 1103 1104 1105
1106 1107 1108 1109 1110 1111 1112 1113 1114 1115 1116 1117 1118 1119
1120 1121 1122 1123 1124 1125 1126 1127 1128 1129 1130 1131 1132 1133
1134 1135 1136 1137 1138 1139 1140 1141 1142 1143 1144 1145 1146 1147
1148 1149 1150 1151 1152 1153 1154 1155 1156 1157 1158 1159 1160 1161
1162 1163 1164 1165 1166 1167 1168 1169 1170 1171 1172 1173 1174 1175
1176 1177 1178 1179 1180 1181 1182 1183 1184 1185 1186 1187 1188 1189
1190 1191 1192 1193 1194 1195 1196 1197 1198 1199 1200 1201 1202 1203

```
1204 1205 1206 1207 1208 1209 1210 1211 1212 1213 1214 1215 1216 1217
1218 1219 1220 1221 1222 1223 1224 1225 1226 1227 1228 1229 1230 1231
1232 1233 1234 1235 1236 1237 1238 1239 1240 1241 1242 1243 1244 1245
1246 1247 1248 1249 1250 1251 1252 1253 1254 1255 1256 1257 1258 1259
1260 1261 1262 1263 1264 1265 1266 1267 1268 1269 1270 1271 1272 1273
1274 1275 1276 1277 1278 1279 1280 1281 1282 1283 1284 1285 1286 1287
1288 1289 1290 1291 1292 1293 1294 1295 1296 1297 1298 1299 1300 1301
1302 1303 1304 1305 1306 1307 1308 1309 1310 1311 1312 1313 1314 1315
1316 1317 1318 1319 1320 1321 1322 1323 1324 1325 1326 1327 1328 1330
1331 1332 1333 1334 1335 1336 1337 1338 1339 1340 1341 1342 1343 1344
1345 1346 1347 1348 1349 1350 1351 1352 1353 1354 1355 1356 1359 1362
1365 1367 1370 1375 1376 1379 1380 1383 1387 1389 1392 1393 1398 1399
1400 1401 1402 1404 1406 1409 1413 1414 1415 1416 1419 1420 1422 1424
1425 1428 1431 1433 1436 1438 1439 1443 1444 1445 1446 1447 1449 1450
1455 1456 1459 1460 1461 1464 1466 1468 1469 1470 1471 1476 1478 1479
1483 1484 1486 1487 1489 1490 1494 1495 1496 1503 1504 1507 1513 1514
1521 1522 1523 1525 1530 1532 1534 1539 1541 1543 1545 1546 1548 1549
1550 1552 1553 1554 1555 1557 1558 1563 1565 1566 1567 1570 1571 1573
1575 1577 1580 1581 1582 1583 1584 1585]
```

Here, we can see that the CV Split 5 is run on the train data subset and has the highest accuracy and F1 score. The accuracy is 0.8 and the F1 score is 0.7604 which is the best across all the CV split results. Hence, we should use this model (trained on the fifth split of the dataset) on the test dataset. But, the model we would evaluate on the test set is the one trained on the entire training dataset after the 5-fold cross-validation process, taking into account the average performance metrics and the stability by the confidence interval. This model represents your best estimate of how the classifier will perform on new, unseen data.

```python
from scipy.stats import sem, t
```

```python
confidence_interval = 0.95
```

```python
accuracy_nb
```

```
[0.7622950819672131,
 0.7404371584699454,
 0.7404371584699454,
 0.7650273224043715,
 0.8]
```

```python
def calculate_confidence_interval(accuracy_scores, confidence_interval):

    mean_accuracy = np.mean(accuracy_scores)
    standard_error = sem(accuracy_scores)
    print(standard_error)

    z_score = 1.959964
    margin_of_error = z_score * standard_error

    # Bounds
    lower_bound = mean_accuracy - margin_of_error
    upper_bound = mean_accuracy + margin_of_error

    print(f"Mean Accuracy: {mean_accuracy:.4f}")
    print(f"95% Confidence Interval: [{lower_bound:.4f}, {upper_bound:.4f}]")
```

```python
calculate_confidence_interval(accuracy_nb, confidence_interval)
```

```
0.010914471268005118
Mean Accuracy: 0.7616
95% Confidence Interval: [0.7402, 0.7830]
```

The accuracy confidence interval for the accuracy score with cross validation for Naive Bayes is [0.7402, 0.7830] and this interval provides a range of values within which the true accuracy of the classifier model is likely to fall with a confidence of 95%. It is a measure of the uncertainty associated with the model's performance. A narrower interval as in this case indicates greater confidence in the model's performance.
(I have considered that this is a sample and hence n-1 is used in calculation of standard deviation)

Task 8: Train a linear SVM classifier

```
from sklearn.svm import LinearSVC
```

```
# counter i
i = 0
f1_svc = []
accuracy_svc = []

for train_index, test_index in cross_validation.split(X_train_mutual_info_best_2000_features, y_label):

    print(f"CV with Linear SVC: {i+1}")

    X_train = X_train_mutual_info_best_2000_features[train_index]
    X_test = X_train_mutual_info_best_2000_features[test_index]
    y_train = y_label[train_index]
    y_test = y_label[test_index]

    linear_svc = LinearSVC(dual=True)

    linear_svc.fit(X_train, y_train)

    print()

    predictions = linear_svc.predict(X_test)

    print ("Overall Accuracy score: ", accuracy_score(y_test, predictions))
    print ("Overall Weighted Recall score: ", recall_score(y_test, predictions, average='weighted'))
    print ("Overall Weighted Precision score: ", precision_score(y_test, predictions, average='weighted'))
    print ("Overall Weighted F1 score: ", f1_score(y_test, predictions, average='weighted'), "\n")
    f1_svc.append(f1_score(y_test, predictions, average='weighted'))
    accuracy_svc.append(accuracy_score(y_test, predictions))
    print (".................................................\n\n")
    i += 1
```

Cross validation output:

```
CV with Linear SVC: 1

Overall Accuracy score:  0.6475409836065574
Overall Weighted Recall score:  0.6475409836065574
Overall Weighted Precision score:  0.6656190102027453
Overall Weighted F1 score:  0.6558938057219318

.................................................


CV with Linear SVC: 2

Overall Accuracy score:  0.674863387978142
Overall Weighted Recall score:  0.674863387978142
Overall Weighted Precision score:  0.7037675219767167
Overall Weighted F1 score:  0.685178843364506

.................................................


CV with Linear SVC: 3

Overall Accuracy score:  0.7021857923497268
Overall Weighted Recall score:  0.7021857923497268
Overall Weighted Precision score:  0.6846943624206632
Overall Weighted F1 score:  0.6913392989892845

.................................................
```

```
CV with Linear SVC: 4

Overall Accuracy score:  0.7349726775956285
Overall Weighted Recall score:  0.7349726775956285
Overall Weighted Precision score:  0.7104527451716034
Overall Weighted F1 score:  0.7195614618227334

..............................................


CV with Linear SVC: 5

Overall Accuracy score:  0.7205479452054795
Overall Weighted Recall score:  0.7205479452054795
Overall Weighted Precision score:  0.6738760300215189
Overall Weighted F1 score:  0.6733759130628014

..............................................
```

CV Split - 4 Training Dataset Indexes:

```
CV with Linear SVC: 4
[   0    1    2    3    4    5    6    7    8    9   10   11   12   13
   14   15   16   17   18   19   20   21   22   23   24   25   26   27
   28   29   30   31   32   33   34   35   36   37   38   39   40   41
   42   43   44   45   46   47   48   49   50   51   52   53   54   55
   56   57   58   59   60   61   62   63   64   65   66   67   68   69
   70   71   72   73   74   75   76   77   78   79   80   81   82   83
   84   85   86   87   88   89   90   91   92   93   94   95   96   97
   98   99  100  101  102  103  104  105  106  107  108  109  110  111
  112  113  114  115  116  117  118  119  120  121  122  123  124  125
  126  127  128  129  130  131  132  133  134  135  136  137  138  139
  140  141  142  143  144  145  146  147  148  149  150  151  152  153
  154  155  156  157  158  159  160  161  162  163  164  165  166  167
  168  169  170  171  172  173  174  175  176  177  178  179  180  181
  182  183  184  185  186  187  188  189  190  191  192  193  194  195
  196  197  198  199  200  201  202  203  204  205  206  207  208  209
  210  211  212  213  214  215  216  217  218  219  220  221  222  223
  224  225  226  227  228  229  230  231  232  233  234  235  236  237
  238  239  240  241  242  243  244  245  246  247  248  249  250  251
  252  253  254  255  256  257  258  259  260  261  262  263  264  265
  266  267  268  269  270  271  272  273  274  275  276  277  278  279
  280  281  282  283  284  285  286  287  288  289  290  291  292  293
  294  295  296  297  298  299  300  301  302  303  304  305  306  307
  308  309  310  311  312  313  314  315  316  317  318  319  320  321
  322  323  324  325  326  327  328  329  330  331  332  333  334  335
  336  337  338  339  340  341  342  343  344  345  346  347  348  349
  350  351  352  353  354  355  356  357  358  359  360  361  362  363
  364  365  366  367  368  369  370  371  372  373  374  375  376  377
  378  379  380  381  382  383  384  385  386  387  388  389  390  391
  392  393  394  395  396  397  398  399  400  401  402  403  404  405
  406  407  408  409  410  411  412  413  414  415  416  417  418  419
  420  421  422  423  424  425  426  427  428  429  430  431  432  433
  434  435  436  437  438  439  440  441  442  443  444  445  446  447
  448  449  450  451  452  453  454  455  456  457  458  459  460  461
  462  463  464  465  466  467  468  469  470  471  472  473  474  475
  476  477  478  479  480  481  482  483  484  485  486  487  488  489
  490  491  492  493  494  495  496  497  498  499  500  501  502  503
  504  505  506  507  508  509  510  511  512  513  514  515  516  517
  518  519  520  521  522  523  524  525  526  527  528  529  530  531
  532  533  534  535  536  537  538  539  540  541  542  543  544  545
  546  547  548  549  550  551  552  553  554  555  556  557  558  559
  560  561  562  563  564  565  566  567  568  569  570  571  572  573
  574  575  576  577  578  579  580  581  582  583  584  585  586  587
  588  589  590  591  592  593  594  595  596  597  598  599  600  601
  602  603  604  605  606  607  608  609  610  611  612  613  614  615
  616  617  618  619  620  621  622  623  624  625  626  627  628  629
  630  631  632  633  634  635  636  637  638  639  640  641  642  643
```

```
 644  645  646  647  648  649  650  651  652  653  654  655  656  657
 658  659  660  661  662  663  664  665  666  667  668  669  670  671
 672  673  674  675  676  677  678  679  680  681  682  683  684  685
 686  687  688  689  690  691  692  693  694  695  696  697  698  699
 700  701  702  703  704  705  706  707  708  709  710  711  712  713
 714  715  716  717  718  719  720  721  722  723  724  725  726  727
 728  729  730  731  732  733  734  735  736  737  738  739  740  741
 742  743  744  745  746  747  748  749  750  751  752  753  754  755
 756  757  758  759  760  761  762  763  764  765  766  767  768  769
 770  771  772  773  774  775  776  777  778  779  780  781  782  783
 784  785  786  787  788  789  790  791  792  793  794  795  796  797
 798  799  800  801  802  803  804  805  806  807  808  809  810  811
 812  813  814  815  816  817  818  819  820  821  822  823  824  825
 826  827  828  829  830  831  832  833  834  835  836  837  838  839
 840  841  842  843  844  845  846  847  848  849  850  851  852  853
 854  855  856  857  858  859  860  861  862  863  864  865  866  867
 868  869  870  871  872  873  874  875  876  877  878  879  880  881
 882  883  884  885  886  887  888  889  890  891  892  893  894  895
 896  897  898  899  900  901  902  903  904  905  906  907  908  909
 910  911  912  913  914  915  916  917  918  919  920  921  922  923
 924  925  926  927  928  929  930  931  932  933  934  935  936  937
 938  939  942  947  952  953  958  960  961  964  965  966  969  970
 975  988  991  993  996  998 1001 1005 1007 1014 1016 1019 1020 1021
1023 1027 1033 1040 1041 1043 1047 1052 1054 1055 1056 1060 1064 1065
1067 1068 1070 1072 1087 1088 1091 1092 1093 1094 1096 1101 1102 1109
1111 1122 1123 1127 1130 1138 1140 1142 1145 1146 1150 1152 1153 1154
1155 1160 1164 1165 1167 1168 1169 1171 1172 1173 1178 1179 1180 1181
1182 1184 1186 1187 1191 1193 1195 1201 1202 1205 1210 1211 1214 1215
1216 1218 1227 1232 1233 1235 1236 1238 1241 1246 1249 1250 1251 1252
1258 1261 1262 1264 1266 1268 1271 1275 1279 1283 1288 1292 1293 1295
1298 1300 1306 1309 1312 1314 1315 1323 1328 1329 1331 1332 1334 1339
1340 1341 1343 1344 1345 1346 1350 1351 1352 1353 1354 1356 1357 1358
1359 1360 1361 1362 1363 1364 1365 1366 1367 1368 1369 1370 1371 1372
1373 1374 1375 1376 1377 1378 1379 1380 1381 1382 1384 1385 1386 1388
1390 1391 1394 1395 1396 1397 1403 1405 1407 1408 1410 1411 1412 1417
1418 1421 1423 1426 1427 1429 1430 1432 1434 1435 1437 1440 1441 1442
1448 1451 1452 1453 1454 1457 1458 1462 1463 1465 1467 1472 1473 1474
1475 1477 1480 1481 1482 1485 1488 1491 1492 1493 1497 1498 1499 1500
1501 1502 1505 1506 1508 1509 1510 1511 1512 1515 1516 1517 1518 1519
1520 1524 1526 1527 1528 1529 1531 1533 1535 1536 1537 1538 1540 1542
1544 1547 1551 1556 1559 1560 1561 1562 1564 1568 1569 1572 1574 1576
1578 1579 1586 1587 1588 1589 1590 1591 1592 1593 1594 1595 1596 1597
1598 1599 1600 1601 1602 1603 1604 1605 1606 1607 1608 1609 1610 1611
1612 1613 1614 1615 1616 1617 1618 1619 1620 1621 1622 1623 1624 1625
1626 1627 1628 1629 1630 1631 1632 1633 1634 1635 1636 1637 1638 1639
1640 1641 1642 1643 1644 1645 1646 1647 1648 1649 1650 1651 1652 1653
1654 1655 1656 1657 1658 1659 1660 1661 1662 1663 1664 1665 1666 1667
1668 1669 1670 1671 1672 1673 1674 1675 1676 1677 1678 1679 1680 1681
1682 1683 1684 1685 1686 1687 1688 1689 1690 1691 1692 1693 1694 1695
1696 1697 1698 1699 1700 1701 1702 1703 1704 1705 1706 1707 1708 1709
1710 1711 1712 1713 1714 1715 1716 1717 1718 1719 1720 1721 1722 1723
1724 1725 1726 1727 1728 1729 1730 1731 1732 1733 1734 1735 1736 1737
1738 1739 1740 1741 1742 1743 1744 1745 1746 1747 1748 1749 1750 1751
1752 1753 1754 1755 1756 1757 1758 1759 1760 1761 1762 1763 1764 1765
1766 1767 1768 1769 1770 1771 1772 1773 1774 1775 1776 1777 1778 1779
1780 1781 1782 1783 1784 1785 1786 1787 1788 1789 1790 1791 1792 1793
1794 1795 1796 1797 1798 1799 1800 1801 1802 1803 1804 1805 1806 1807
1808 1809 1810 1811 1812 1813 1814 1815 1816 1817 1818 1819 1820 1821
1822 1823 1824 1825 1826 1827 1828]
```

Here, we can see that the CV Split 4 is run on the train data subset and has the highest accuracy and F1 score. The accuracy is 0.7349 and the F1 score is 0.7196 which is the best across all the CV split results. Hence, we should use this model (trained on the fourth split of the dataset) on the test dataset. But, the model we would evaluate on the test set is the one trained on the entire training dataset after the 5-fold cross-validation process, taking into account the average performance metrics and the stability by the confidence interval. This model represents your best estimate of how the classifier will perform on new, unseen data.

```
accuracy_svc
```

```
[0.6475409836065574,
 0.674863387978142,
 0.7021857923497268,
 0.7349726775956285,
 0.7205479452054795]
```

```
calculate_confidence_interval(accuracy_svc, confidence_interval)
```

```
Mean Accuracy: 0.6960
95% Confidence Interval: [0.6652, 0.7269]
```

The accuracy confidence interval for the accuracy score with cross validation for Linear SVC is [0.6652, 0.7269] and this interval provides a range of values within which the true accuracy of the classifier model is likely to fall with a confidence of 95%. It is a measure of the uncertainty associated with the model's performance. A broader interval as in this case indicates the model's performance might be poor on unseen data.

(I have considered that this is a sample and hence n-1 is used in calculation of standard deviation)

Task 9: Train a logistic regression classifier

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

```
# counter i
i = 0
f1_lr = []
accuracy_lr = []

for train_index, test_index in cross_validation.split(X_train_mutual_info_best_2000_features, y_label):

    print(f"CV with Logistic Regression: {i+1}")

    X_train = X_train_mutual_info_best_2000_features[train_index]
    X_test = X_train_mutual_info_best_2000_features[test_index]
    y_train = y_label[train_index]
    y_test = y_label[test_index]

    logistic_regression = LogisticRegression(max_iter=50000)

    logistic_regression.fit(X_train, y_train)

    print()

    predictions = logistic_regression.predict(X_test)

    print ("Overall Accuracy score: ", accuracy_score(y_test, predictions))
    print ("Overall Weighted Recall score: ", recall_score(y_test, predictions, average='weighted'))
    print ("Overall Weighted Precision score: ", precision_score(y_test, predictions, average='weighted'))
    print ("Overall Weighted F1 score: ", f1_score(y_test, predictions, average='weighted'), "\n")
    f1_lr.append(f1_score(y_test, predictions, average='weighted'))
    accuracy_lr.append(accuracy_score(y_test, predictions))
    print (".................................................\n\n")
    i += 1
```

Cross validation outputs:

```
CV with Logistic Regression: 1

Overall Accuracy score:  0.6721311475409836
Overall Weighted Recall score:  0.6721311475409836
Overall Weighted Precision score:  0.686377586245145
Overall Weighted F1 score:  0.6788006388200585

.................................................


CV with Logistic Regression: 2

Overall Accuracy score:  0.7158469945355191
Overall Weighted Recall score:  0.7158469945355191
Overall Weighted Precision score:  0.7116041781054723
Overall Weighted F1 score:  0.7133182685241422

.................................................


CV with Logistic Regression: 3

Overall Accuracy score:  0.7131147540983607
Overall Weighted Recall score:  0.7131147540983607
Overall Weighted Precision score:  0.6759404979844112
Overall Weighted F1 score:  0.6899667912456394

.................................................
```

```
CV with Logistic Regression: 4

Overall Accuracy score:  0.7513661202185792
Overall Weighted Recall score:  0.7513661202185792
Overall Weighted Precision score:  0.7221007893139041
Overall Weighted F1 score:  0.7253747229354577

.............................................


CV with Logistic Regression: 5

Overall Accuracy score:  0.7205479452054795
Overall Weighted Recall score:  0.7205479452054795
Overall Weighted Precision score:  0.6749802977039077
Overall Weighted F1 score:  0.6672201439683422

.............................................
```

CV Split - 4 Training Dataset Indexes:

```
CV with Logistic Regression: 4
[   0    1    2    3    4    5    6    7    8    9   10   11   12   13
   14   15   16   17   18   19   20   21   22   23   24   25   26   27
   28   29   30   31   32   33   34   35   36   37   38   39   40   41
   42   43   44   45   46   47   48   49   50   51   52   53   54   55
   56   57   58   59   60   61   62   63   64   65   66   67   68   69
   70   71   72   73   74   75   76   77   78   79   80   81   82   83
   84   85   86   87   88   89   90   91   92   93   94   95   96   97
   98   99  100  101  102  103  104  105  106  107  108  109  110  111
  112  113  114  115  116  117  118  119  120  121  122  123  124  125
  126  127  128  129  130  131  132  133  134  135  136  137  138  139
  140  141  142  143  144  145  146  147  148  149  150  151  152  153
  154  155  156  157  158  159  160  161  162  163  164  165  166  167
  168  169  170  171  172  173  174  175  176  177  178  179  180  181
  182  183  184  185  186  187  188  189  190  191  192  193  194  195
  196  197  198  199  200  201  202  203  204  205  206  207  208  209
  210  211  212  213  214  215  216  217  218  219  220  221  222  223
  224  225  226  227  228  229  230  231  232  233  234  235  236  237
  238  239  240  241  242  243  244  245  246  247  248  249  250  251
  252  253  254  255  256  257  258  259  260  261  262  263  264  265
  266  267  268  269  270  271  272  273  274  275  276  277  278  279
  280  281  282  283  284  285  286  287  288  289  290  291  292  293
  294  295  296  297  298  299  300  301  302  303  304  305  306  307
  308  309  310  311  312  313  314  315  316  317  318  319  320  321
  322  323  324  325  326  327  328  329  330  331  332  333  334  335
  336  337  338  339  340  341  342  343  344  345  346  347  348  349
  350  351  352  353  354  355  356  357  358  359  360  361  362  363
  364  365  366  367  368  369  370  371  372  373  374  375  376  377
  378  379  380  381  382  383  384  385  386  387  388  389  390  391
  392  393  394  395  396  397  398  399  400  401  402  403  404  405
  406  407  408  409  410  411  412  413  414  415  416  417  418  419
  420  421  422  423  424  425  426  427  428  429  430  431  432  433
  434  435  436  437  438  439  440  441  442  443  444  445  446  447
  448  449  450  451  452  453  454  455  456  457  458  459  460  461
  462  463  464  465  466  467  468  469  470  471  472  473  474  475
  476  477  478  479  480  481  482  483  484  485  486  487  488  489
  490  491  492  493  494  495  496  497  498  499  500  501  502  503
  504  505  506  507  508  509  510  511  512  513  514  515  516  517
  518  519  520  521  522  523  524  525  526  527  528  529  530  531
  532  533  534  535  536  537  538  539  540  541  542  543  544  545
  546  547  548  549  550  551  552  553  554  555  556  557  558  559
  560  561  562  563  564  565  566  567  568  569  570  571  572  573
  574  575  576  577  578  579  580  581  582  583  584  585  586  587
  588  589  590  591  592  593  594  595  596  597  598  599  600  601
  602  603  604  605  606  607  608  609  610  611  612  613  614  615
```

```
 616  617  618  619  620  621  622  623  624  625  626  627  628  629
 630  631  632  633  634  635  636  637  638  639  640  641  642  643
 644  645  646  647  648  649  650  651  652  653  654  655  656  657
 658  659  660  661  662  663  664  665  666  667  668  669  670  671
 672  673  674  675  676  677  678  679  680  681  682  683  684  685
 686  687  688  689  690  691  692  693  694  695  696  697  698  699
 700  701  702  703  704  705  706  707  708  709  710  711  712  713
 714  715  716  717  718  719  720  721  722  723  724  725  726  727
 728  729  730  731  732  733  734  735  736  737  738  739  740  741
 742  743  744  745  746  747  748  749  750  751  752  753  754  755
 756  757  758  759  760  761  762  763  764  765  766  767  768  769
 770  771  772  773  774  775  776  777  778  779  780  781  782  783
 784  785  786  787  788  789  790  791  792  793  794  795  796  797
 798  799  800  801  802  803  804  805  806  807  808  809  810  811
 812  813  814  815  816  817  818  819  820  821  822  823  824  825
 826  827  828  829  830  831  832  833  834  835  836  837  838  839
 840  841  842  843  844  845  846  847  848  849  850  851  852  853
 854  855  856  857  858  859  860  861  862  863  864  865  866  867
 868  869  870  871  872  873  874  875  876  877  878  879  880  881
 882  883  884  885  886  887  888  889  890  891  892  893  894  895
 896  897  898  899  900  901  902  903  904  905  906  907  908  909
 910  911  912  913  914  915  916  917  918  919  920  921  922  923
 924  925  926  927  928  929  930  931  932  933  934  935  936  937
 938  939  942  947  952  953  958  960  961  964  965  966  969  970
 975  988  991  993  996  998 1001 1005 1007 1014 1016 1019 1020 1021
1023 1027 1033 1040 1041 1043 1047 1052 1054 1055 1056 1060 1064 1065
1067 1068 1070 1072 1087 1088 1091 1092 1093 1094 1096 1101 1102 1109
1111 1122 1123 1127 1130 1138 1140 1142 1145 1146 1150 1152 1153 1154
1155 1160 1164 1165 1167 1168 1169 1171 1172 1173 1178 1179 1180 1181
1182 1184 1186 1187 1191 1193 1195 1201 1202 1205 1210 1211 1214 1215
1216 1218 1227 1232 1233 1235 1236 1238 1241 1246 1249 1250 1251 1252
1258 1261 1262 1264 1266 1268 1271 1275 1279 1283 1288 1292 1293 1295
1298 1300 1306 1309 1312 1314 1315 1323 1328 1329 1331 1332 1334 1339
1340 1341 1343 1344 1345 1346 1350 1351 1352 1353 1354 1356 1357 1358
1359 1360 1361 1362 1363 1364 1365 1366 1367 1368 1369 1370 1371 1372
1373 1374 1375 1376 1377 1378 1379 1380 1381 1382 1384 1385 1386 1388
1390 1391 1394 1395 1396 1397 1403 1405 1407 1408 1410 1411 1412 1417
1418 1421 1423 1426 1427 1429 1430 1432 1434 1435 1437 1440 1441 1442
1448 1451 1452 1453 1454 1457 1458 1462 1463 1465 1467 1472 1473 1474
1475 1477 1480 1481 1482 1485 1488 1491 1492 1493 1497 1498 1499 1500
1501 1502 1505 1506 1508 1509 1510 1511 1512 1515 1516 1517 1518 1519
1520 1524 1526 1527 1528 1529 1531 1533 1535 1536 1537 1538 1540 1542
1544 1547 1551 1556 1559 1560 1561 1562 1564 1568 1569 1572 1574 1576
1578 1579 1586 1587 1588 1589 1590 1591 1592 1593 1594 1595 1596 1597
1598 1599 1600 1601 1602 1603 1604 1605 1606 1607 1608 1609 1610 1611
1612 1613 1614 1615 1616 1617 1618 1619 1620 1621 1622 1623 1624 1625
1626 1627 1628 1629 1630 1631 1632 1633 1634 1635 1636 1637 1638 1639
1640 1641 1642 1643 1644 1645 1646 1647 1648 1649 1650 1651 1652 1653
1654 1655 1656 1657 1658 1659 1660 1661 1662 1663 1664 1665 1666 1667
1668 1669 1670 1671 1672 1673 1674 1675 1676 1677 1678 1679 1680 1681
1682 1683 1684 1685 1686 1687 1688 1689 1690 1691 1692 1693 1694 1695
1696 1697 1698 1699 1700 1701 1702 1703 1704 1705 1706 1707 1708 1709
1710 1711 1712 1713 1714 1715 1716 1717 1718 1719 1720 1721 1722 1723
1724 1725 1726 1727 1728 1729 1730 1731 1732 1733 1734 1735 1736 1737
1738 1739 1740 1741 1742 1743 1744 1745 1746 1747 1748 1749 1750 1751
1752 1753 1754 1755 1756 1757 1758 1759 1760 1761 1762 1763 1764 1765
1766 1767 1768 1769 1770 1771 1772 1773 1774 1775 1776 1777 1778 1779
1780 1781 1782 1783 1784 1785 1786 1787 1788 1789 1790 1791 1792 1793
1794 1795 1796 1797 1798 1799 1800 1801 1802 1803 1804 1805 1806 1807
1808 1809 1810 1811 1812 1813 1814 1815 1816 1817 1818 1819 1820 1821
1822 1823 1824 1825 1826 1827 1828]
```

Here, we can see that the CV Split 4 is run on the train data subset and has the highest accuracy and F1 score. The accuracy is 0.7514 and the F1 score is 0.7254 which is the best across all the CV split results. Hence, we should use this model (trained on the fourth split of the dataset) on the test dataset. But, the model we would evaluate on the test set is the one trained on the entire training dataset after the 5-fold cross-validation process, taking into account the average performance

metrics and the stability by the confidence interval. This model represents your best estimate of how the classifier will perform on new, unseen data.

```
accuracy_lr
```

```
[0.6721311475409836,
 0.7158469945355191,
 0.7131147540983607,
 0.7513661202185792,
 0.7205479452054795]
```

```
calculate_confidence_interval(accuracy_lr, confidence_interval)
```

```
Mean Accuracy: 0.7146
95% Confidence Interval: [0.6898, 0.7394]
```

The accuracy confidence interval for the accuracy score with cross validation for Logistic Regression is [0.6898, 0.7394] and this interval provides a range of values within which the true accuracy of the classifier model is likely to fall with a confidence of 95%. It is a measure of the uncertainty associated with the model's performance. A broader interval as in this case indicates the model's performance might be poor on unseen data.

(I have considered that this is a sample and hence n-1 is used in calculation of standard deviation)

Task 10: Model comparison using paired t-test

A) Naive Bayes vs Linear SVC

For this, we have the list of F1 scores for both models from the 5 fold cross validation runs ans we will use them to perform the paired t-tests.

Null Hypothesis:

The 2 independent samples of the F1 scores of Naive Bayes and F1 scores of Linear SVC have identical average values. ($\mu_{nb} = \mu_{svc}$)

Alternate Hypothesis:

The mean of the samples of F1 scores of Naive Bayes is greater than the means of the samples of F1 scores of Linear SVC. ($\mu_{nb} > \mu_{svc}$)

```
# Student's t-test for Linear SVC vs Naive Bayes

mnb_svc_ttest = stats.ttest_ind(f1_nb, f1_svc, alternative='greater')
print("Student's t-test result for Linear SVC vs Naive Bayes: ", mnb_svc_ttest)
```

```
Student's t-test result for Linear SVC vs Naive Bayes:  Ttest_indResult(statistic=4.813487870897465, pvalue=0.0006661
935588968813)
```

The t-test statistic = 4.8135
P-value = 0.0006
Level of significance α = 0.05

Since we are considering greater than hypothesis, we need to check if the P-value < α
In our case, P-value = 0.0006 < 0.05

Hence, we can reject the Null hypothesis and conclude that the mean of F1 scores of Naive Bayes is greater than the mean of F1 scores of Linear SVC.

B) Naive Bayes vs Logistic Regression

For this, we have the list of F1 scores for both models from the 5 fold cross validation runs ans we will use them to perform the paired t-tests.

Null Hypothesis:

The 2 independent samples of the F1 scores of Naive Bayes and F1 scores of Logistic Regression have identical average values. ($\mu_{nb} = \mu_{lr}$)

Alternate Hypothesis:

The mean of the samples of F1 scores of Naive Bayes is greater than the means of the samples of F1 scores of Logistic Regression. ($\mu_{nb} > \mu_{lr}$)

The t-test statistic = 3.939
P-value = 0.002
Level of significance α = 0.05

Since we are considering greater than test, we need to check if the P-value < α

In our case, P-value = 0.002 < 0.05

Hence, we can reject the Null hypothesis and conclude that the mean of F1 scores of Naive Bayes is greater than the mean of F1 scores of Logistic Regression.

C) Linear SVC vs Logistic Regression

For this, we have the list of F1 scores for both models from the 5 fold cross validation runs ans we will use them to perform the paired t-tests.

Null Hypothesis:

The 2 independent samples of the F1 scores of Linear SVC and F1 scores of Logistic Regression have identical average values. ($\mu_{svc} = \mu_{lr}$)

Alternate Hypothesis:

The mean of the samples of F1 scores of Linear SVC is greater than the means of the samples of F1 scores of Logistic Regression. ($\mu_{svc} > \mu_{lr}$)

```
# Student's t-test for Linear SVC vs Logistic Regression

lvc_lr_ttest = stats.ttest_ind(f1_svc, f1_lr, alternative='greater')
print("Student's t-test result for Linear SVC vs Logistic Regression: ", lvc_lr_ttest)
```

```
Student's t-test result for Linear SVC vs Logistic Regression:  Ttest_indResult(statistic=-0.6552346866913336, pvalue
=0.7346487315731108)
```

The t-test statistic = -0.6552
P-value = 0.7345
Level of significance α = 0.05

Since we are considering greater than test, we need to check if the P-value < α

In our case, P-value = 0.7345 > 0.05

Hence, we cannot reject the Null hypothesis and conclude that the mean of F1 scores of Linear SVC is identical to the mean of F1 scores of Logistic Regression.

Comparing all the results, we can say that the Naive Bayes model has the best performance as it has a far more greater F1 Score as compared to the Linear SVC and Logistic Regression. Also the confidence interval for the accuracy is narrower for Naive Bayes indicating that with 95% confidence we can conclude that the average accuracy will lie in the narrow interval range and it improves the model's predictive capability.

But when we compare the Linear SVC and Logistic Regression models as per the T-Test we concluded that the means of the F1 scores for them are identical and hence we can say that the predictive performance of both is equally comparable. But when compared with the Naive Bayes, we concluded that the Naive Bayes has greater average F1 score than both these models. This concludes that Naive Bayes has a better performance in prediction.

Task 11: Hyperparameter tuning

```
from sklearn.model_selection import GridSearchCV
```

```
X_train = X_train_mutual_info_best_2000_features[train_index]
X_test = X_train_mutual_info_best_2000_features[test_index]
y_train = y_label[train_index]
y_test = y_label[test_index]
```

```
svc_C_params = [
    {'C': [0.01, 0.1,1,10,100,1000]}
]
```

```
linear_svc_raw = LinearSVC(dual=True, max_iter=50000)
```

```
grid_cv = GridSearchCV(estimator=linear_svc_raw, param_grid=svc_C_params)
grid_cv.fit(X_train, y_train)
```

▸ GridSearchCV
▸ estimator: LinearSVC
▸ LinearSVC

```
grid_cv.best_params_
```

```
{'C': 0.1}
```

```
grid_cv.best_score_
```

```
0.7261045397166768
```

The best value of regularisation parameter (C) obtained from Grid Search is 0.1

```
linear_SVC_tuned = LinearSVC(dual=True, max_iter=50000, C=0.1)
```

```
linear_SVC_tuned.fit(X_train, y_train)
```

▾ LinearSVC

```
LinearSVC(C=0.1, dual=True, max_iter=50000)
```

```
predictions = linear_SVC_tuned.predict(X_test)
predictions[:5]
```

```
array(['negative', 'positive', 'positive', 'positive', 'positive'],
      dtype=object)
```

```
print ("Overall Accuracy score: ", accuracy_score(y_test, predictions), "\n")
print ("Overall Weighted Recall score: ", recall_score(y_test, predictions, average='weighted'))
print ("Overall Weighted Precision score: ", precision_score(y_test, predictions, average='weighted'))
print ("Overall Weighted F1 score: ", f1_score(y_test, predictions, average='weighted'), "\n")
print ("Individual label performance: ")
print (classification_report(y_test, predictions))
print ("Confusion Matrix: ")
print (confusion_matrix(y_test, predictions))
```

```
Overall Accuracy score:  0.7123287671232876

Overall Weighted Recall score:  0.7123287671232876
Overall Weighted Precision score:  0.6526870389884089
Overall Weighted F1 score:  0.6524200913242011

Individual label performance:
              precision    recall  f1-score   support

    negative       0.78      0.39      0.52        98
     neutral       0.00      0.00      0.00        39
    positive       0.71      0.97      0.82       228

    accuracy                           0.71       365
   macro avg       0.50      0.45      0.45       365
weighted avg       0.65      0.71      0.65       365

Confusion Matrix:
[[ 38   1  59]
 [  8   0  31]
 [  3   3 222]]
```

The weighted precision, recall, and F1 score of the best Linear SVC model are 0.6527, 0.0.7123 and 0.6524 respectively.

The best value of C = 0.1 and is provided by Grid Search hyperparameter tuning technique after performing training and evaluation on an exhaustive set of the parameters provided. It does trial and error for each and every combination of the parameters provided and then provides the best value for the final model.

A smaller C indicates heavy penalty and regularisation which can lead to underfitting. On the other hand, a smaller C indicates least regularisation and can lead to overfitting. This value of C = 0.1 strikes a perfect balance between underfitting and overfitting and can lead to optimal results on unseen data.

Task 12: Discussion on model selection

A) Cross-Validation:

Cross-validation helps assess a model's performance on different subsets of the data. It provides a more robust estimate of how well the model generalizes to unseen data. This helps us in using the part of the training dataset as unseen data and improves model selection. For example in the experiments we ran the 5-fold CV on the training dataset and got 5 different models with varying accuracy and metrics. The average estimate is done.
In text classification, this is essential as the model should perform consistently across various text samples.

B) Hyperparameter Tuning:

Hyperparameter tuning, via techniques like grid search, allows us to find the best hyperparameter values for your text classification model. This ensures that the model is tuned to perform well on your specific text dataset. The different combinations of parameters are tried and the best values are returned for optimal performance.
Proper hyperparameter selection can help avoid overfitting. For example, when we trained the Linear SVC for identifying the best value for the inverse regularization parameter C. An optimal value balances the tradeoff between bias and variance.

C) T-Tests (or Model Comparison):

T-tests for model comparison provide a way for doing statistical validation. It can help understand whether one model is significantly better than another. In text classification, this helps in selecting the best-performing model based on statistical evidence. T-tests offers evidence in the form of hypothesis and p-values. If one model consistently outperforms another with statistical significance, we have a more confident basis for selecting the model. For example, in out T-tests, we found out that the Naive Bayes model has greater F1 score than both Linear SVC and Logistic Regression. It also helped us understand that out of the two models of Linear SVC and Logistic Regression, both of them had equivalent F1 score and similar predictive performance. Hence T-Tests can be useful in understanding which model is better based on statistical tests.

References:

1) geeksforgeeks.com

2) Canvas Class Slides: https://canvas.illinois.edu/courses/37566/pages/course-schedule

3) www.google.com

4) https://www.omnicalculator.com/statistics/confidence-interval

5) https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.ttest_ind.html