Proj_05_Naive_Bayes_Email_Spam_Full_5

August 4, 2024

1 EE915: Week-5 - Project-5 - Naive Bayes - Email Spam Detection and Purchase Prediction

Name: Venkateswar Reddy Melachervu **Roll No.**: 23156022 **Email**: vmela23@iitk.ac.in **Dataset**: Breast Cancer Data Set from sklearn **Professor**: Prof. Aditya K. Jagganatham **TAs**: Meesam and Priyanka

This project implements naive Bayes algorithm for email spam classification on the data sets provided. The code includes pre-processing steps such as data normalization, feature selection, splitting the dataset into training. The Gaussian Naive Bayes classifier is used for this classification.

The evaluation metrics used include accuracy, F1 score, precision, recall, RoC/AuC, decision boundary etc.

Additionally, Laplace smoothing is applied for better accuracy in prediction - particularly in the spam and ham classification of the emails dataset.

[ ]:
```
%pip install seaborn
%pip install wordcloud
```

[12]:
```
# Importing necessary libraries for breast cancer SVM classifier
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import confusion_matrix, classification_report,␣
↪accuracy_score, roc_curve, roc_auc_score, precision_recall_curve, f1_score,␣
↪average_precision_score
from sklearn.preprocessing import LabelEncoder
from collections import Counter
from wordcloud import WordCloud
from sklearn.manifold import TSNE
from sklearn.decomposition import PCA
```

[13]:
```
# Define roll number, name, email
roll_number = "23156022"
name = "Venkateswar Reddy Melachervu"
email = "vmela23@iitk.ac.in"

# Load the dataset
df = pd.read_csv('Kaggle_Email_Spam_Dataset.csv', encoding='latin-1')

# Remove unnecessary columns
df.drop(['Unnamed: 2', 'Unnamed: 3', 'Unnamed: 4'], axis=1, inplace=True)

# Rename columns for clarity
df.columns = ['label', 'message']

# Verify column names and data
print(df.head())
```

```
  label                                            message
0   ham  Go until jurong point, crazy.. Available only …
1   ham                      Ok lar… Joking wif u oni…
2  spam  Free entry in 2 a wkly comp to win FA Cup fina…
3   ham  U dun say so early hor… U c already then say…
4   ham  Nah I don't think he goes to usf, he lives aro…
```

[14]:
```
# function to display confusion matrix plot with watermark - wama
def print_confusion_matrix_wama(cm, labels, plot_name, Y_test, wama='23156022'):
    # Calculate counts for each class in the test set
    class_counts = Counter(Y_test)

    # Print the counts
    print("Data Count in Test Set:")
    for class_label, count in class_counts.items():
        if class_label == 0:
            print(f"\tHam: {count}")
        elif class_label == 1:
            print(f"\tSpam: {count}")
        else:
            pass

    # Plot the confusion matrix
    plt.figure(figsize=(8, 6))
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=labels,␣
 ↪yticklabels=labels)

    plt.xlabel('Predicted Label')
    plt.ylabel('True Label')
    plt.title(plot_name)
```

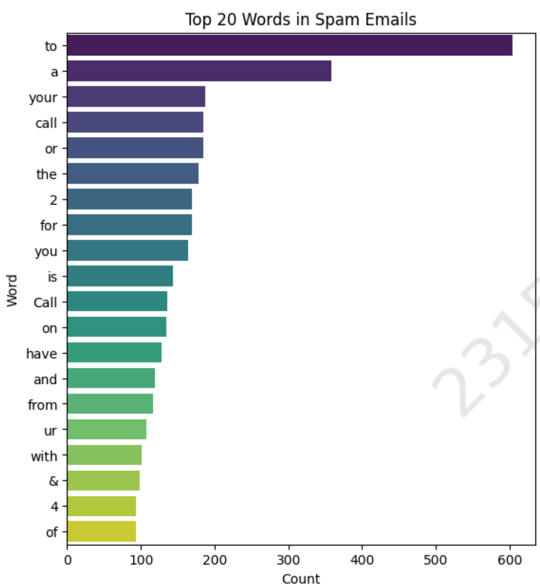
```
    # Add centered diagonal watermark
    plt.text(0.5, 0.5, wama, fontsize=50, color='gray', alpha=0.2,
             rotation=45, ha='center', va='center', transform=plt.gca().
 ↪transAxes)

    plt.show()
```

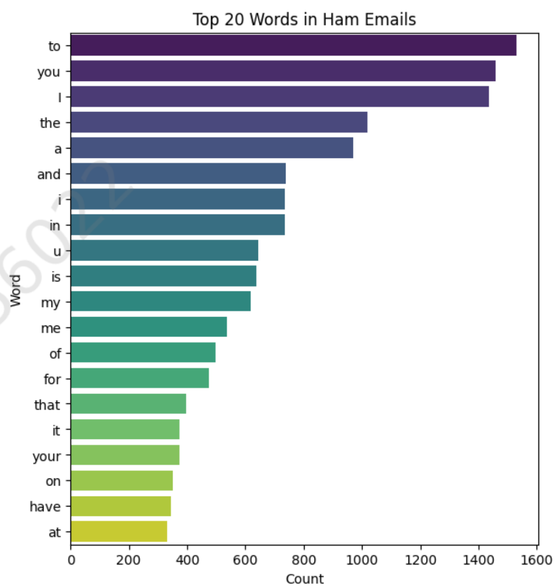
[15]:
```
# Plot word cloud, bar plot, and tSNE for the spam data set

# 1. Word Cloud Visualization
spam_text = " ".join(df[df['label'] == 'spam']['message'])
ham_text = " ".join(df[df['label'] == 'ham']['message'])

spam_wc = WordCloud(width=800, height=400, background_color='white').
 ↪generate(spam_text)
ham_wc = WordCloud(width=800, height=400, background_color='white').
 ↪generate(ham_text)

plt.figure(figsize=(14, 7))
plt.subplot(1, 2, 1)
plt.imshow(spam_wc, interpolation='bilinear')
plt.title('Spam Emails')
plt.axis('off')

plt.subplot(1, 2, 2)
plt.imshow(ham_wc, interpolation='bilinear')
plt.title('Ham Emails')
plt.axis('off')

# Add watermark
plt.text(0.5, 0.5, roll_number, fontsize=50, color='gray', alpha=0.2,
         rotation=45, ha='center', va='center', transform=plt.gcf().transFigure)

plt.show()

# 2. Bar Plot for Most Common Words
spam_words = " ".join(df[df['label'] == 'spam']['message']).split()
ham_words = " ".join(df[df['label'] == 'ham']['message']).split()

spam_counter = Counter(spam_words)
ham_counter = Counter(ham_words)

spam_df = pd.DataFrame(spam_counter.most_common(20), columns=['Word', 'Count'])
ham_df = pd.DataFrame(ham_counter.most_common(20), columns=['Word', 'Count'])

plt.figure(figsize=(14, 7))
```

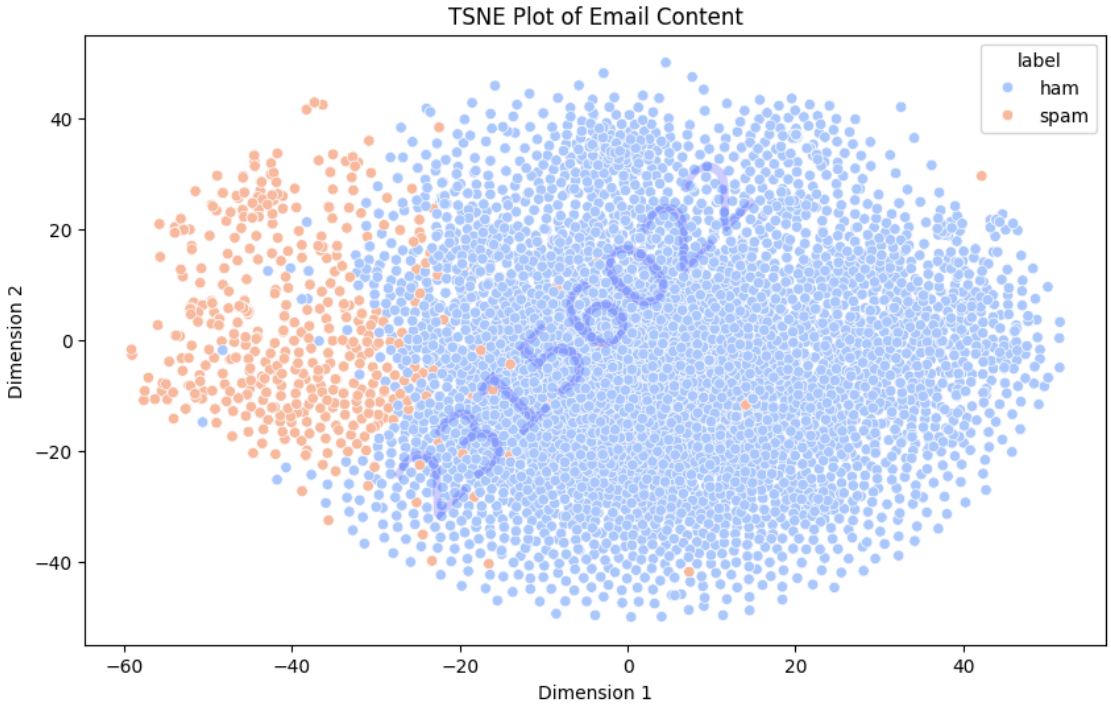
```
plt.subplot(1, 2, 1)
ax1 = sns.barplot(x='Count', y='Word', hue='Word', legend=False, data=spam_df,␣
 ↪palette='viridis', dodge=False)
plt.title('Top 20 Words in Spam Emails')
# Remove legend if it exists
if ax1.get_legend() is not None:
    ax1.get_legend().remove()

plt.subplot(1, 2, 2)
ax2 = sns.barplot(x='Count', y='Word', hue='Word', legend=False, data=ham_df,␣
 ↪palette='viridis', dodge=False)
plt.title('Top 20 Words in Ham Emails')
# Remove legend if it exists
if ax2.get_legend() is not None:
    ax2.get_legend().remove()

# Add watermark
plt.text(0.5, 0.5, roll_number, fontsize=50, color='gray', alpha=0.2,
         rotation=45, ha='center', va='center', transform=plt.gcf().transFigure)

plt.show()

# 3. TSNE Plot
vectorizer = TfidfVectorizer(max_features=10000)
X = vectorizer.fit_transform(df['message'])

tsne = TSNE(n_components=2, random_state=42)
X_tsne = tsne.fit_transform(X.toarray())

tsne_df = pd.DataFrame(X_tsne, columns=['Dimension 1', 'Dimension 2'])
tsne_df['label'] = df['label']

plt.figure(figsize=(10, 6))
sns.scatterplot(x='Dimension 1', y='Dimension 2', hue='label', data=tsne_df,␣
 ↪palette='coolwarm')
plt.title('TSNE Plot of Email Content')

# Add watermark
plt.text(0.5, 0.5, roll_number, fontsize=50, color='blue', alpha=0.2,
         rotation=45, ha='center', va='center', transform=plt.gcf().transFigure)

plt.show()
```


Spam Emails
FREE
call
text
txt
now
mobile
claim
won
reply
STOP
win
new
URGENT
prize
cash
tone
service
150p
chat
message
ringtone


Ham Emails
Ok
will
now
U
lt
gt
come
got
know
love
day
time
call
want
go
one
need
good
going
still


Top 20 Words in Spam Emails
to
a
your
call
or
the
2
for
you
is
Call
on
have
and
from
ur
with
&
4
of
Word
0
100
200
300
400
500
600
Count

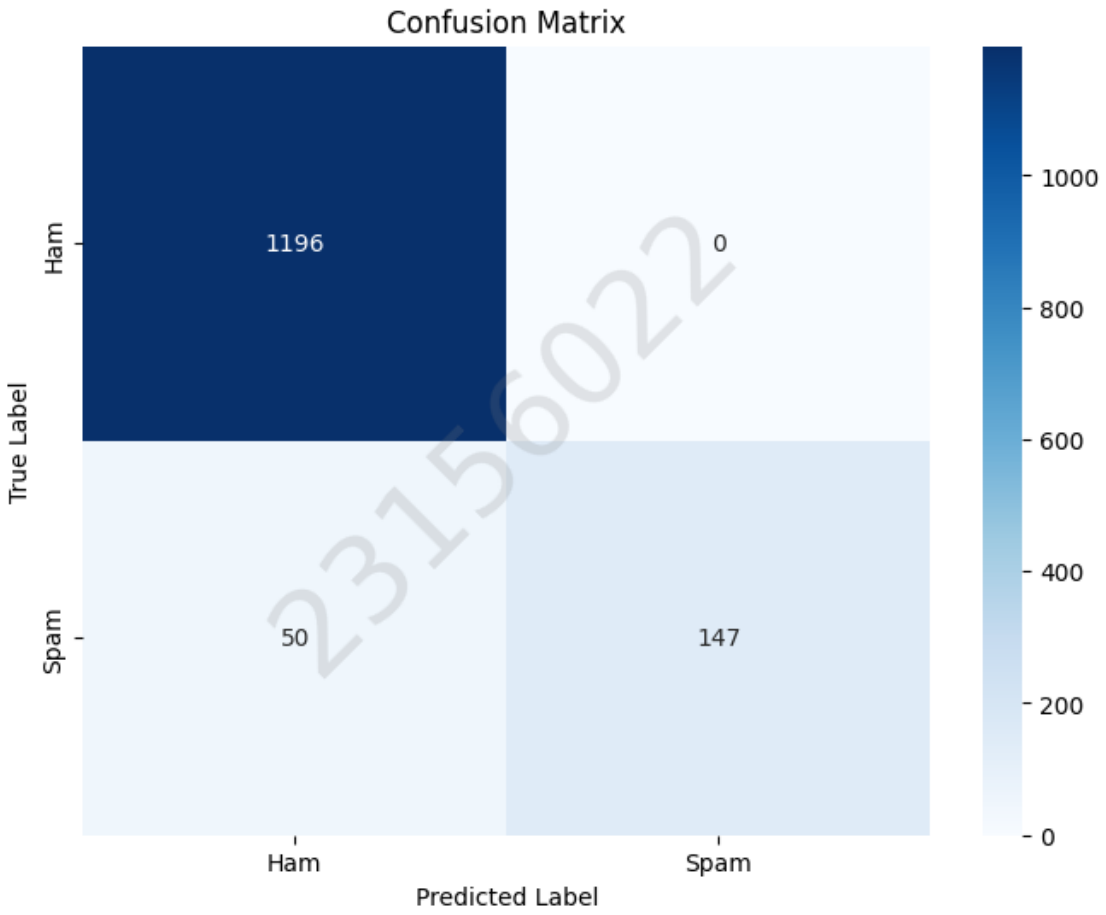

Top 20 Words in Ham Emails
to
you
I
the
a
and
i
in
u
is
my
me
of
for
that
it
your
on
have
at
Word
0
200
400
600
800
1000
1200
1400
1600
Count



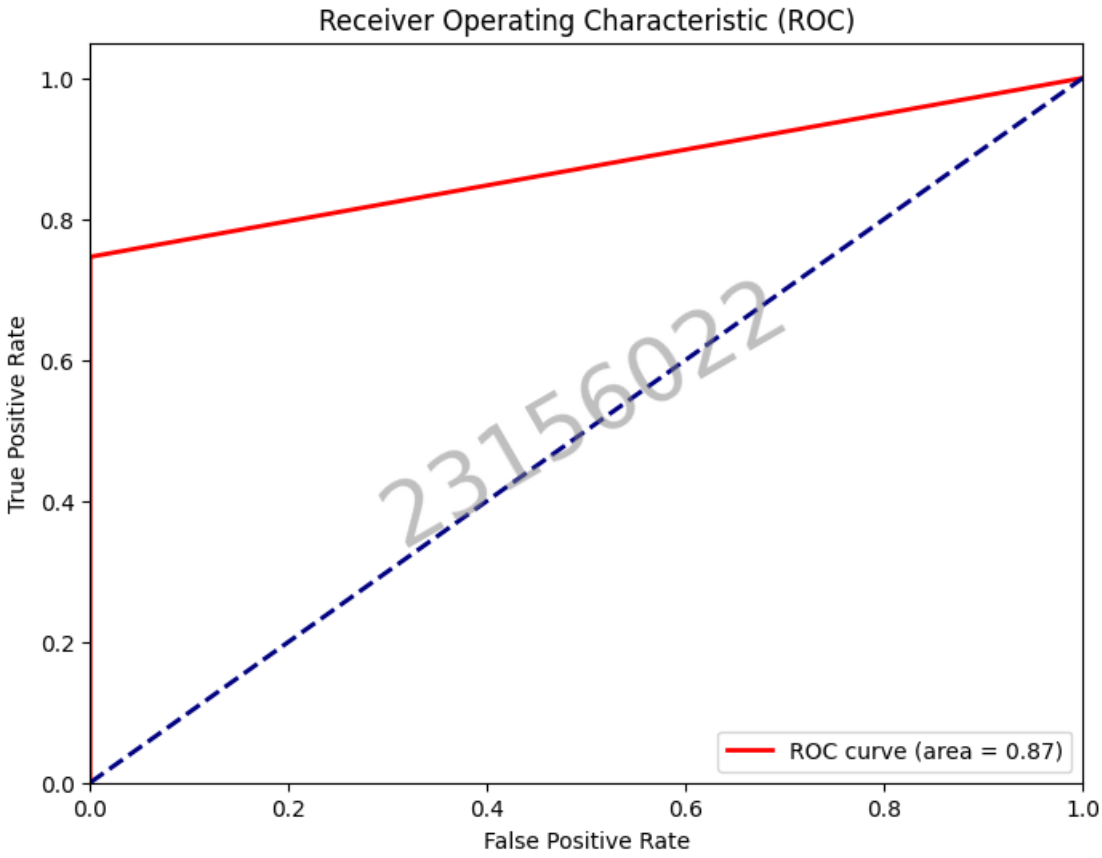
[17]:
```
# Build NB model, predict and plot curves

# Encode the target labels
label_encoder = LabelEncoder()

df['label'] = label_encoder.fit_transform(df['label'])  # Convert 'ham' and␣
 ↪'spam' to 0 and 1

# Check the unique values in the encoded target variable
print(f'Unique values in encoded labels: {df["label"].unique()}')

# Feature Extraction
vectorizer = TfidfVectorizer(stop_words='english')
X = vectorizer.fit_transform(df['message'])
y = df['label']

# Split the data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25,␣
 ↪random_state=0)

# Create and train the Naive Bayes classifier with Laplace smoothing
naive_bayes = MultinomialNB(alpha=1.0)  # alpha is the smoothing parameter
naive_bayes.fit(X_train, y_train)
```

```python
# Make predictions
y_pred = naive_bayes.predict(X_test)

# Evaluate the classifier
print(f'Spam Labeling Accuracy Score: {accuracy_score(y_test, y_pred)}')

cm = confusion_matrix(y_test, y_pred)

cr = classification_report(y_test, y_pred).upper()
print('Classification Report:\n', cr)

# Plot confusion matrix with print_confusion_matrix_wama
labels = ['Ham', 'Spam']
plot_name = 'Confusion Matrix'
print_confusion_matrix_wama(cm, labels, plot_name, y_test)
```

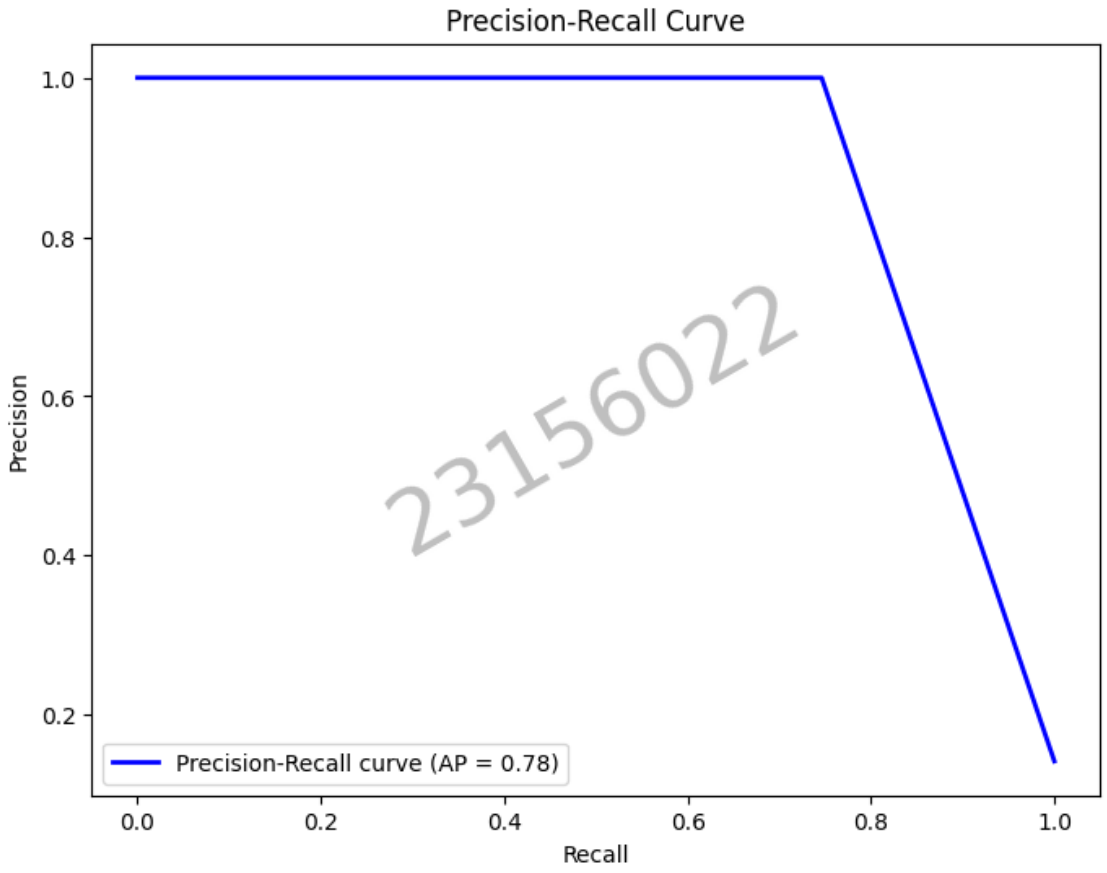
```
Unique values in encoded labels: [0 1]
Spam Labeling Accuracy Score: 0.9641062455132807
Classification Report:
               PRECISION    RECALL  F1-SCORE   SUPPORT

           0       0.96      1.00      0.98      1196
           1       1.00      0.75      0.85       197

    ACCURACY                           0.96      1393
   MACRO AVG       0.98      0.87      0.92      1393
WEIGHTED AVG       0.97      0.96      0.96      1393

Data Count in Test Set:
        Ham: 1196
        Spam: 197
```



[6]:
```
# Plot ROC curve
fpr, tpr, thresholds = roc_curve(y_test, y_pred)
roc_auc = roc_auc_score(y_test, y_pred)

# Plot ROC curve
plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='red', lw=2, label=f'ROC curve (area = {roc_auc:.2f})')
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC)')
plt.legend(loc='lower right')
plt.text(0.5, 0.5, roll_number, fontsize=40, color='gray', alpha=0.5,␣
 ↪ha='center', va='center', rotation=30, transform=plt.gca().transAxes)
plt.show()
```



[7]:
```
# Plot precision-recall curve
precision, recall, _ = precision_recall_curve(y_test, y_pred)
average_precision = average_precision_score(y_test, y_pred)

# Plot Precision-Recall curve
plt.figure(figsize=(8, 6))
plt.plot(recall, precision, color='blue', lw=2, label=f'Precision-Recall curve␣
 ↪(AP = {average_precision:.2f})')
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title('Precision-Recall Curve')
plt.legend(loc='lower left')
plt.text(0.5, 0.5, roll_number, fontsize=40, color='gray', alpha=0.5,␣
 ↪ha='center', va='center', rotation=30, transform=plt.gca().transAxes)
plt.show()
```



[10]:
```
# Plot decision boundary

# Feature Extraction
vectorizer = TfidfVectorizer(stop_words='english')
X = vectorizer.fit_transform(df['message'])
y = df['label']

# Reduce dimensions using PCA or t-SNE
reducer = PCA(n_components=2)  # Alternatively, use TSNE(n_components=2)
X_reduced = reducer.fit_transform(X.toarray())

# Split the data
X_train, X_test, y_train, y_test = train_test_split(X_reduced, y, test_size=0.
↪25, random_state=0)

# Create and train the Naive Bayes classifier
naive_bayes = MultinomialNB(alpha=1.0)
naive_bayes.fit(X_train, y_train)
```

```
# Create a mesh grid for plotting decision boundaries
x_min, x_max = X_reduced[:, 0].min() - 1, X_reduced[:, 0].max() + 1
y_min, y_max = X_reduced[:, 1].min() - 1, X_reduced[:, 1].max() + 1
xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.01),
                     np.arange(y_min, y_max, 0.01))

# Predict on the mesh grid
Z = naive_bayes.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)

# Plot decision boundaries
plt.figure(figsize=(10, 6))
plt.contourf(xx, yy, Z, alpha=0.3, cmap=plt.cm.coolwarm)
plt.scatter(X_reduced[:, 0], X_reduced[:, 1], c=y, edgecolor='k', cmap=plt.cm.
 ↪coolwarm)
plt.title('Naive Bayes Decision Boundary')
plt.xlabel('Component 1')
plt.ylabel('Component 2')
plt.colorbar()
plt.text(0.5, 0.5, roll_number, fontsize=40, color='gray', alpha=0.5,␣
 ↪ha='center', va='center', rotation=30, transform=plt.gca().transAxes)
plt.show()
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[10], line 17
     15 # Create and train the Naive Bayes classifier
     16 naive_bayes = MultinomialNB(alpha=1.0)
---> 17 naive_bayes.fit(X_train, y_train)
     19 # Create a mesh grid for plotting decision boundaries
     20 x_min, x_max = X_reduced[:, 0].min() - 1, X_reduced[:, 0].max() + 1

File ~\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.
 ↪11_qbz5n2kfra8p0\LocalCache\local-packages\Python311\site-packages\sklearn\ba e.
 ↪py:1473, in _fit_context.<locals>.decorator.<locals>.wrapper(estimator, *args ␣
 ↪**kwargs)
   1466     estimator._validate_params()
   1468 with config_context(
   1469     skip_parameter_validation=(
   1470         prefer_skip_nested_validation or global_skip_validation
   1471     )
   1472 ):
-> 1473     return fit_method(estimator, *args, **kwargs)

File ~\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.
 ↪11_qbz5n2kfra8p0\LocalCache\local-packages\Python311\site-packages\sklearn\na ve_bayes.
 ↪py:759, in _BaseDiscreteNB.fit(self, X, y, sample_weight)
    757 n_classes = Y.shape[1]
```

```
    758 self._init_counters(n_classes, n_features)
--> 759 self._count(X, Y)
    760 alpha = self._check_alpha()
    761 self._update_feature_log_prob(alpha)

File ~\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.
 ↪11_qbz5n2kfra8p0\LocalCache\local-packages\Python311\site-packages\sklearn\na .ve_bayes.
 ↪py:881, in MultinomialNB._count(self, X, Y)
    879 def _count(self, X, Y):
    880     """Count and smooth feature occurrences."""
--> 881     check_non_negative(X, "MultinomialNB (input X)")
    882     self.feature_count_ += safe_sparse_dot(Y.T, X)
    883     self.class_count_ += Y.sum(axis=0)

File ~\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.
 ↪11_qbz5n2kfra8p0\LocalCache\local-packages\Python311\site-packages\sklearn\ut .ls\validatio
 ↪py:1689, in check_non_negative(X, whom)
   1686     X_min = xp.min(X)
   1688 if X_min < 0:
-> 1689     raise ValueError("Negative values in data passed to %s" % whom)

ValueError: Negative values in data passed to MultinomialNB (input X)
```

[8]:
```
!jupyter nbconvert --to pdf Proj_05_Naive_Bayes_Email_Spam_Full_5.ipynb
```

```
[NbConvertApp] Converting notebook Proj_05_Naive_Bayes_Email_Spam_Full_5.ipynb
to pdf
[NbConvertApp] Support files will be in
Proj_05_Naive_Bayes_Email_Spam_Full_5_files\
[NbConvertApp] Making directory .\Proj_05_Naive_Bayes_Email_Spam_Full_5_files
[NbConvertApp] Writing 52316 bytes to notebook.tex
[NbConvertApp] Building PDF
[NbConvertApp] Running xelatex 3 times: ['xelatex', 'notebook.tex', '-quiet']
[NbConvertApp] Running bibtex 1 time: ['bibtex', 'notebook']
[NbConvertApp] WARNING | b had problems, most likely because there were no
citations
[NbConvertApp] PDF successfully created
[NbConvertApp] Writing 785553 bytes to Proj_05_Naive_Bayes_Email_Spam_Full_5.pdf
```