nlp-spam-detection

January 4, 2024

[228]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

import warnings
warnings.filterwarnings("ignore")
```

[229]:
```
data = pd.read_csv('/kaggle/input/spamdata/spam.csv', encoding='ISO-8859-1')
data.head()
```

[229]:
```
     v1                                                 v2 Unnamed: 2  \
0   ham  Go until jurong point, crazy.. Available only …         NaN
1   ham                      Ok lar… Joking wif u oni…         NaN
2  spam  Free entry in 2 a wkly comp to win FA Cup fina…         NaN
3   ham  U dun say so early hor… U c already then say…         NaN
4   ham  Nah I don't think he goes to usf, he lives aro…         NaN

  Unnamed: 3 Unnamed: 4
0        NaN        NaN
1        NaN        NaN
2        NaN        NaN
3        NaN        NaN
4        NaN        NaN
```

[230]:
```
data.describe()
```

[230]:
```
          v1                      v2  \
count   5572                    5572
unique     2                    5169
top      ham  Sorry, I'll call later
freq    4825                      30

                                                Unnamed: 2  \
count                                                   50
unique                                                  43
top      bt not his girlfrnd… G o o d n i g h t . . .@"
```

```
freq                                                    3

                    Unnamed: 3 Unnamed: 4
count                       12          6
unique                      10          5
top      MK17 92H. 450Ppw 16"    GNT:-)"
freq                         2          2
```

[231]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5572 entries, 0 to 5571
Data columns (total 5 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   v1          5572 non-null   object
 1   v2          5572 non-null   object
 2   Unnamed: 2  50 non-null     object
 3   Unnamed: 3  12 non-null     object
 4   Unnamed: 4  6 non-null      object
dtypes: object(5)
memory usage: 217.8+ KB
```

[232]:
```
data.shape
```

[232]:
```
(5572, 5)
```

[233]:
```
data.isnull().sum()
```

[233]:
```
v1              0
v2              0
Unnamed: 2    5522
Unnamed: 3    5560
Unnamed: 4    5566
dtype: int64
```

droping the columns that are : Unnamed:2, Unnamed:3, Unnamed:4

[234]:
```
data=data.drop(['Unnamed: 2','Unnamed: 3','Unnamed: 4'],axis=1)
data.head()
```

[234]:

	v1	v2
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…

(v1,v2) rename

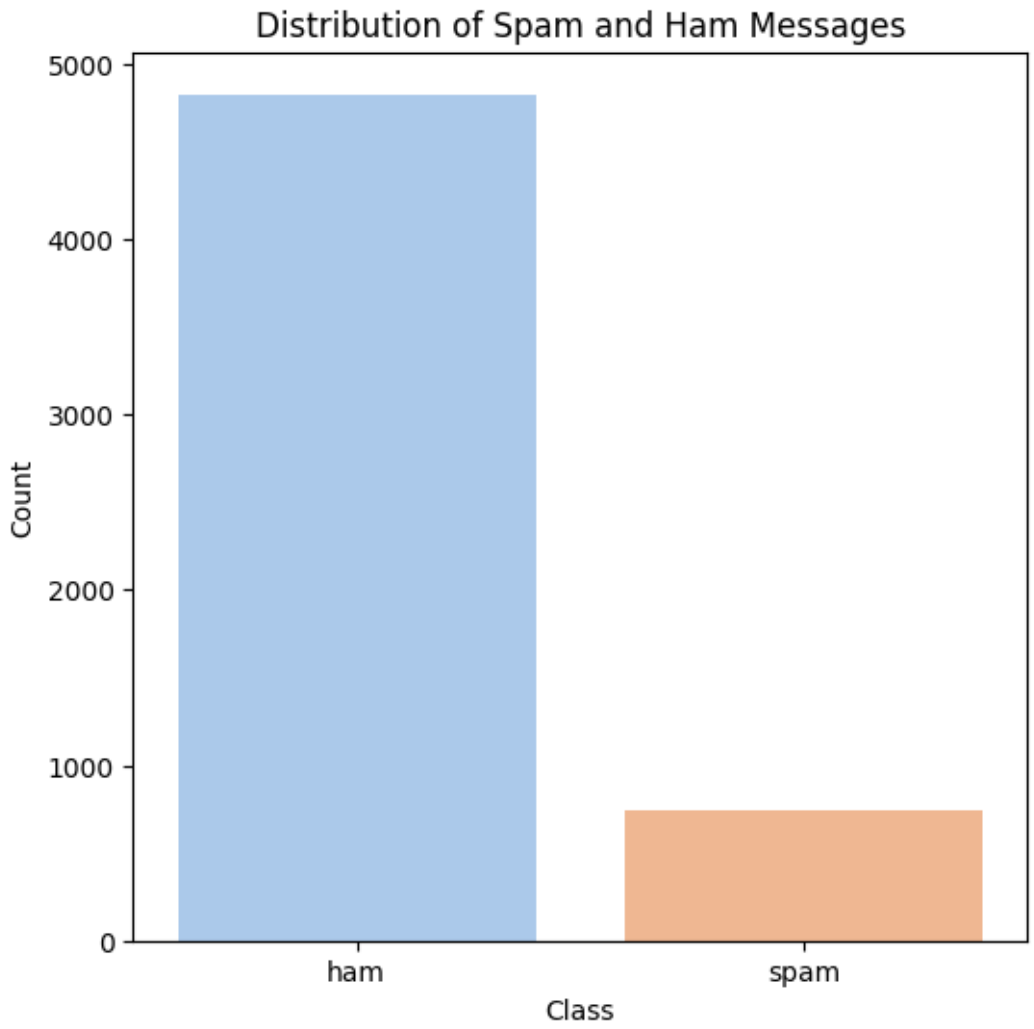
[235]:
```
data=data.rename({'v1':'Class','v2':'Message'},axis=1)
data.head()
```

[235]:

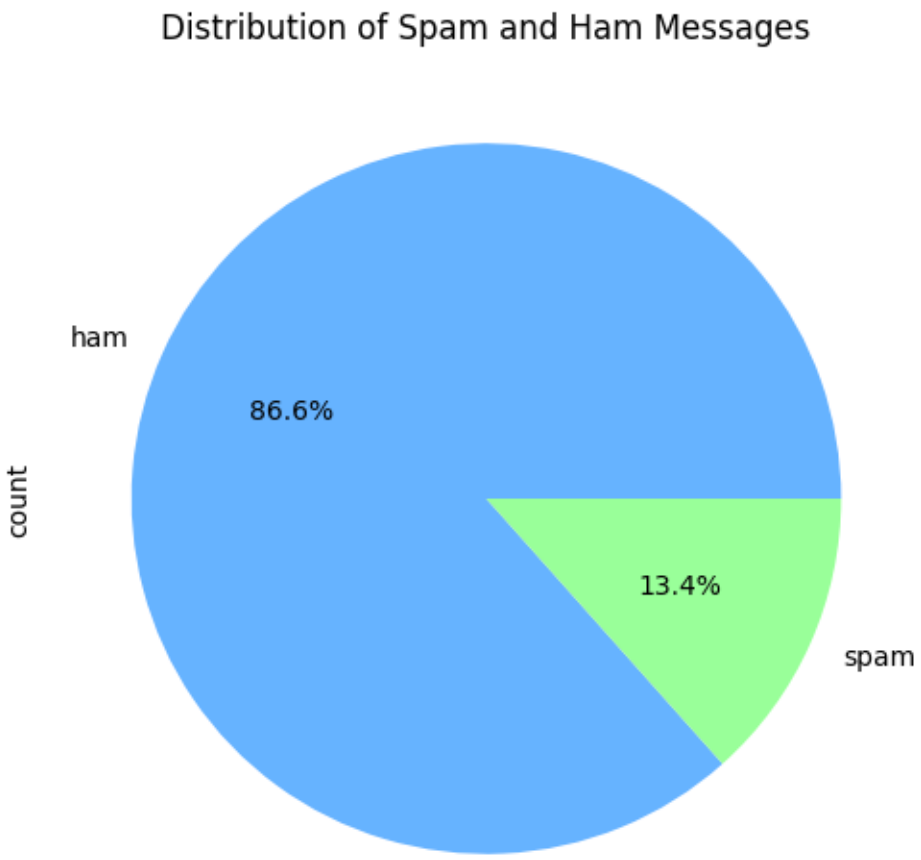
	Class	Message
0	ham	Go until jurong point, crazy.. Available only …
1	ham	Ok lar… Joking wif u oni…
2	spam	Free entry in 2 a wkly comp to win FA Cup fina…
3	ham	U dun say so early hor… U c already then say…
4	ham	Nah I don't think he goes to usf, he lives aro…

0.0.1 Exploratory Data Analysis

[236]:
```
# Plot the count of each class
plt.figure(figsize=(6, 6))
sns.countplot(x='Class', data=data, palette='pastel')
plt.xlabel('Class')
plt.ylabel('Count')
plt.title('Distribution of Spam and Ham Messages')
plt.show()
```



[237]:
```
plt.figure(figsize=(6, 6))
data['Class'].value_counts().plot(kind='pie', autopct='%1.1f%%',␣
↪colors=['#66b3ff', '#99ff99'])
plt.title('Distribution of Spam and Ham Messages')
plt.show()
```



[238]:
```
import nltk
import re
from nltk.corpus import stopwords
nltk.download('stopwords')
STOPWORDS = stopwords.words('english')
```

```
[nltk_data] Downloading package stopwords to /usr/share/nltk_data…
[nltk_data]   Package stopwords is already up-to-date!
```

[239]:
```
def clean_text(text):
    text = text.lower()
    text = re.sub(r'[^0-9a-zA-Z]', ' ', text)
    text = re.sub(r'\s+', ' ', text)
    text = " ".join(word for word in text.split() if word not in STOPWORDS)
    return text
```

[240]:
```
data['clean_text'] = data['Message'].apply(clean_text)
data.head()
```

[240]:
```
  Class                                            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…

                                          clean_text
0  go jurong point crazy available bugis n great …
1                            ok lar joking wif u oni
2  free entry 2 wkly comp win fa cup final tkts 2…
3                u dun say early hor u c already say
4             nah think goes usf lives around though
```

[241]:
```
X = data['clean_text']
y = data['Class']
```

[242]:
```
from sklearn.feature_extraction.text import CountVectorizer
from nltk.stem import PorterStemmer
from nltk.tokenize import sent_tokenize,word_tokenize
import string
ps=PorterStemmer
words=word_tokenize('clean_text')
```

[243]:
```
#define a function to get rid of stopwords present in the messages
def message_text_process(mess):
    # Check characters to see if there are punctuations
    no_punctuation=[char for char in mess if char not in string.punctuation]
    # now form the sentence
    no_punctuation=''.join(no_punctuation)
    # Now eliminate any stopwords
    return[word for word in no_punctuation.split() if word.lower() not in␣
 ↪stopwords.words('english')]
```

[244]:
```
data['Message'].head(5).apply(message_text_process)
```

[244]:
```
0    [Go, jurong, point, crazy, Available, bugis, n…
1                        [Ok, lar, Joking, wif, u, oni]
2    [Free, entry, 2, wkly, comp, win, FA, Cup, fin…
3        [U, dun, say, early, hor, U, c, already, say]
4    [Nah, dont, think, goes, usf, lives, around, t…
Name: Message, dtype: object
```

[245]:
```
# bag of words by applying the function and fit the data(message) into it
bag_of_words_transformer=CountVectorizer(analyzer=message_text_process).
 ↪fit(data['Message'])
# print the length of bag of words stored in vocabulary_attribute
print(len(bag_of_words_transformer.vocabulary_))
```

```
11304
```

[246]:
```
#store bag of words for messages using transform method
message_bagofwords=bag_of_words_transformer.transform(data['Message'])
```

[247]:
```
#apply tfidf transformer and fit the bag of words into it(transformed version)
from sklearn.feature_extraction.text import TfidfTransformer
tfidf_transformer=TfidfTransformer().fit(message_bagofwords)
```

[248]:
```
#print shape of tfidf
message_tfidf=tfidf_transformer.transform(message_bagofwords)
print(message_tfidf.shape)
```

```
(5572, 11304)
```

[249]:
```
# choose naive bayes model to detect the spam and fit the tfidf data into it
from sklearn.naive_bayes import MultinomialNB
spam_detection_model=MultinomialNB().fit(message_tfidf,data['Class'])
```

[250]:
```
message = data['Message'][3]
bag_of_words_for_message = bag_of_words_transformer.transform([message])
tfidf = tfidf_transformer.transform(bag_of_words_for_message)

predicted_class = spam_detection_model.predict(tfidf)[0]
result_message = f'The message is predicted as {predicted_class.upper()}.'
print(result_message)
```

```
The message is predicted as HAM.
```

[251]:
```
#importing PCA for the dimensionality reduction
from sklearn.pipeline import Pipeline
from sklearn.linear_model import LinearRegression
from sklearn.decomposition import PCA
from sklearn.model_selection import train_test_split
import scikitplot as skplt
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
from sklearn.metrics import accuracy_score
```

[252]:
```
#function for the model building and prediction
def Model(model, X, y):
```

```
#training and testing the data
    x_train, x_test, y_train, y_test = train_test_split(X, y, test_size=0.25,␣
 ↪random_state=30)
    # model building using CountVectorizer and TfidfTransformer
    pipeline_model = Pipeline([('vect', CountVectorizer()),
                               ('tfidf', TfidfTransformer()),
                               ('clf', model)])
    pipeline_model.fit(x_train, y_train)

    y_pred = pipeline_model.predict(x_test)
    y_probas =pipeline_model.predict_proba(x_test)
    skplt.metrics.
 ↪plot_roc(y_test,y_probas,figsize=(12,8),title_fontsize=12,text_fontsize=16)
    plt.show()
    skplt.metrics.
 ↪plot_precision_recall(y_test,y_probas,figsize=(12,8),title_fontsize=12,text_fontsize=16)
    plt.show()

    print("Confusion Matrix:\n",confusion_matrix(y_test,y_pred))
    print("Classification Report is:\n",classification_report(y_test, y_pred))
    print('Accuracy:', pipeline_model.score(x_test, y_test)*100)
    print("Training Score:\n",pipeline_model.score(x_train,y_train)*100)
```

1 Model Building

1.0.1 1. Logistic Regression

1.0.2 2. KNeighborsClassifier

1.0.3 3. SVM

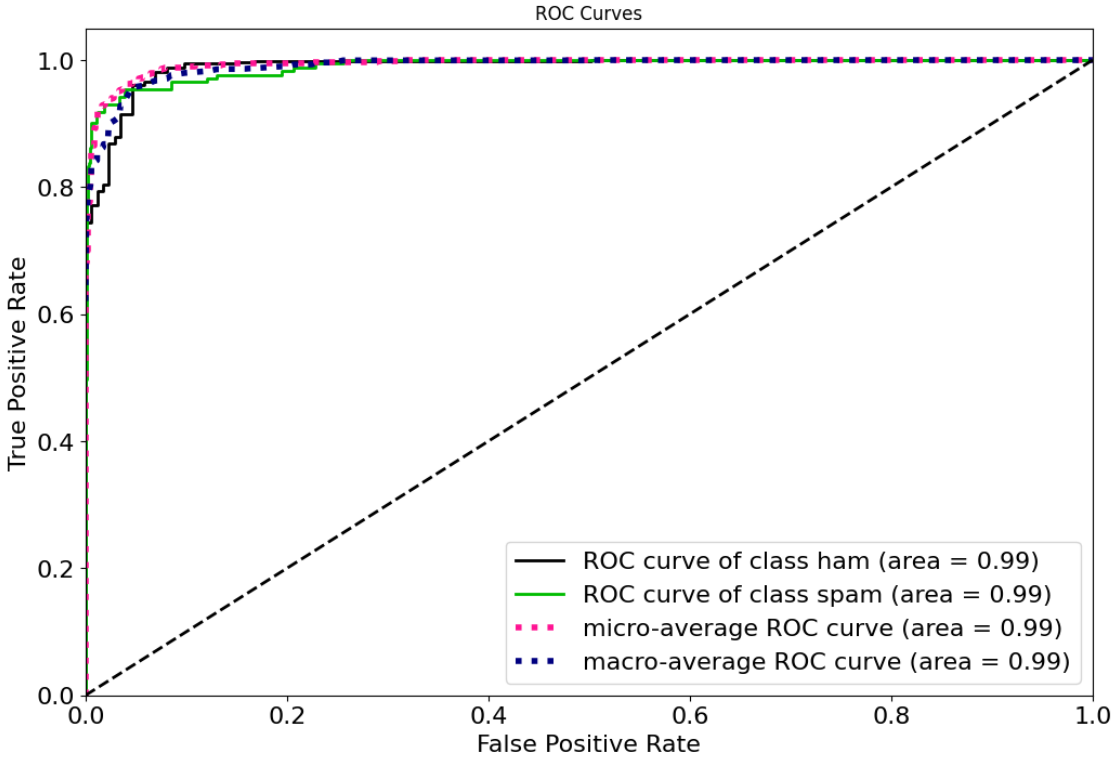
1.0.4 4. Naive Bayes

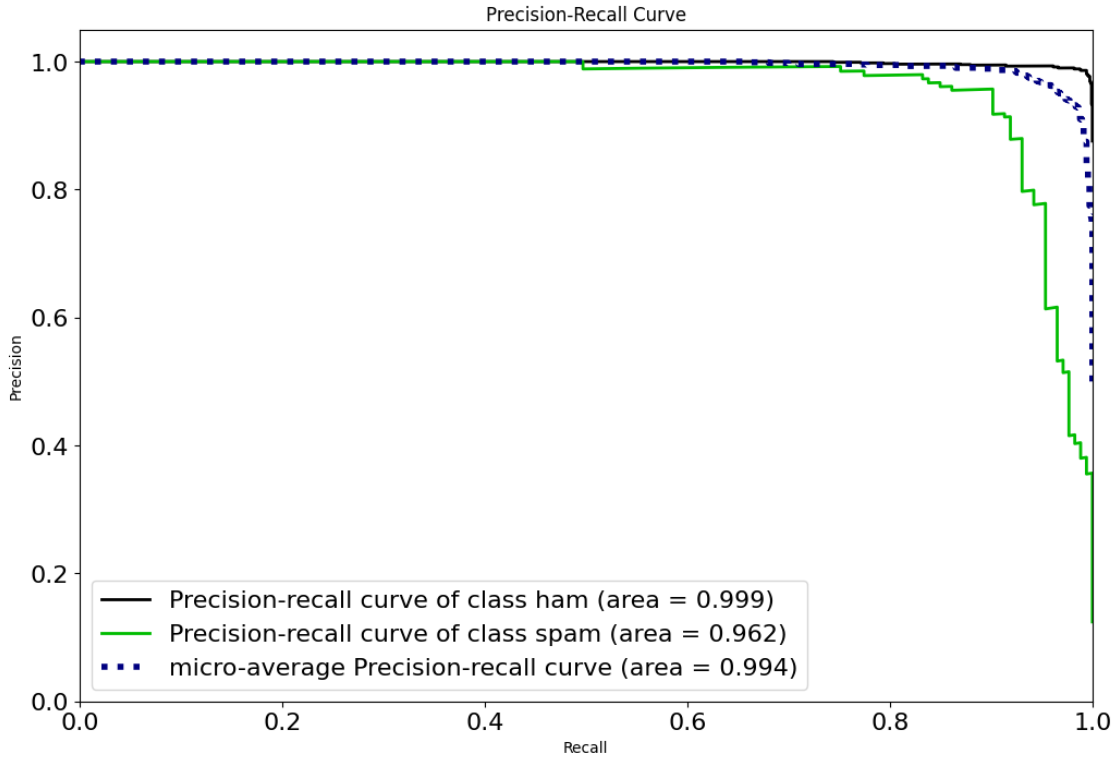
1.0.5 5. DECISION TREE CLASSIFIER

1.0.6 6. RandomForestClassifier

1.1 1. Logistic Regression

[253]:
```
from sklearn.linear_model import LogisticRegression
model = LogisticRegression()
Model(model, X, y)
```

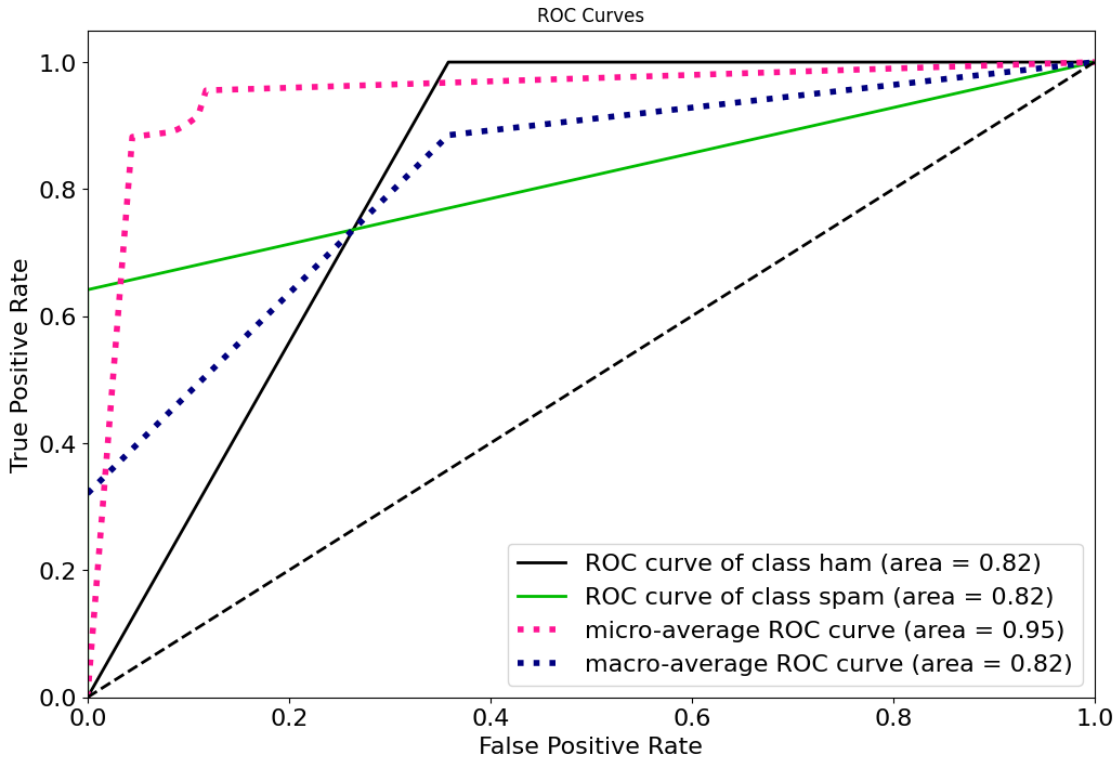

ROC Curves
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
ROC curve of class ham (area = 0.99)
ROC curve of class spam (area = 0.99)
micro-average ROC curve (area = 0.99)
macro-average ROC curve (area = 0.99)

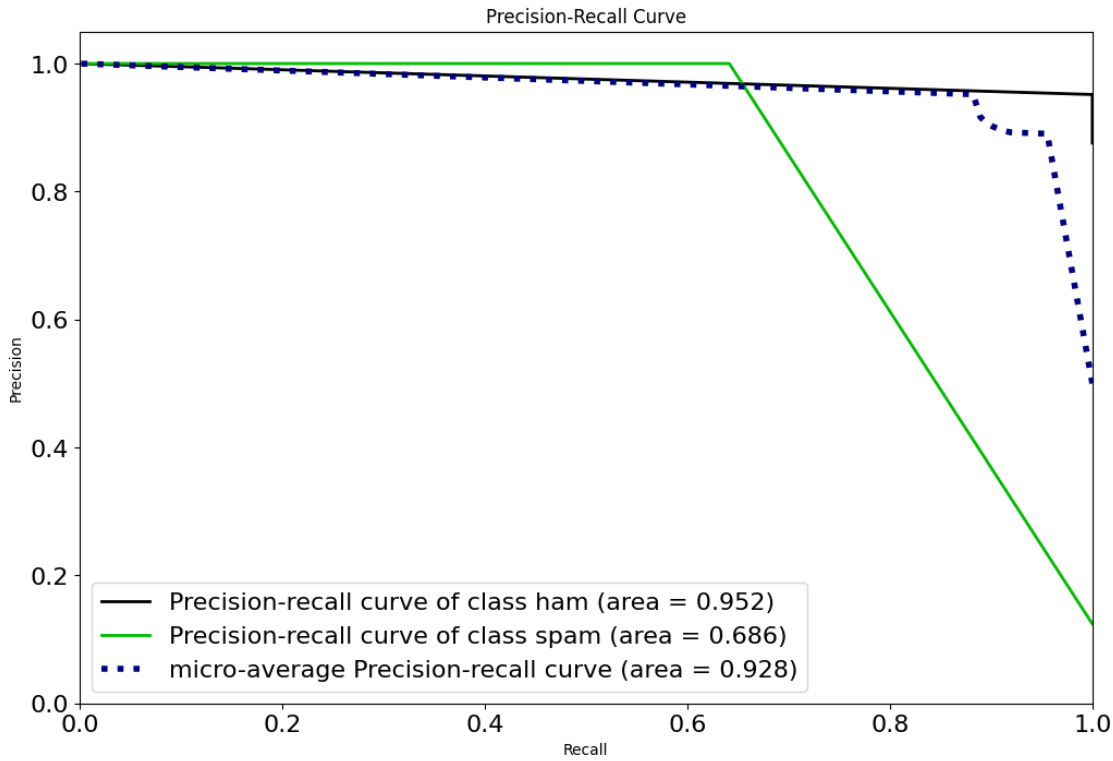

Precision-Recall Curve
Precision
Recall
0.0
0.2
0.4
0.6
0.8
1.0
Precision-recall curve of class ham (area = 0.999)
Precision-recall curve of class spam (area = 0.962)
micro-average Precision-recall curve (area = 0.994)

```
Confusion Matrix:
 [[1219    1]
 [  52  121]]
Classification Report is:
              precision    recall  f1-score   support

         ham       0.96      1.00      0.98      1220
        spam       0.99      0.70      0.82       173

    accuracy                           0.96      1393
   macro avg       0.98      0.85      0.90      1393
weighted avg       0.96      0.96      0.96      1393

Accuracy: 96.19526202440775
Training Score:
 97.00885379277338
```

1.2 2. KNeighborsClassifier

[254]:
```
from sklearn.neighbors import KNeighborsClassifier
model=KNeighborsClassifier(n_neighbors=7)
Model(model,X,y)
```

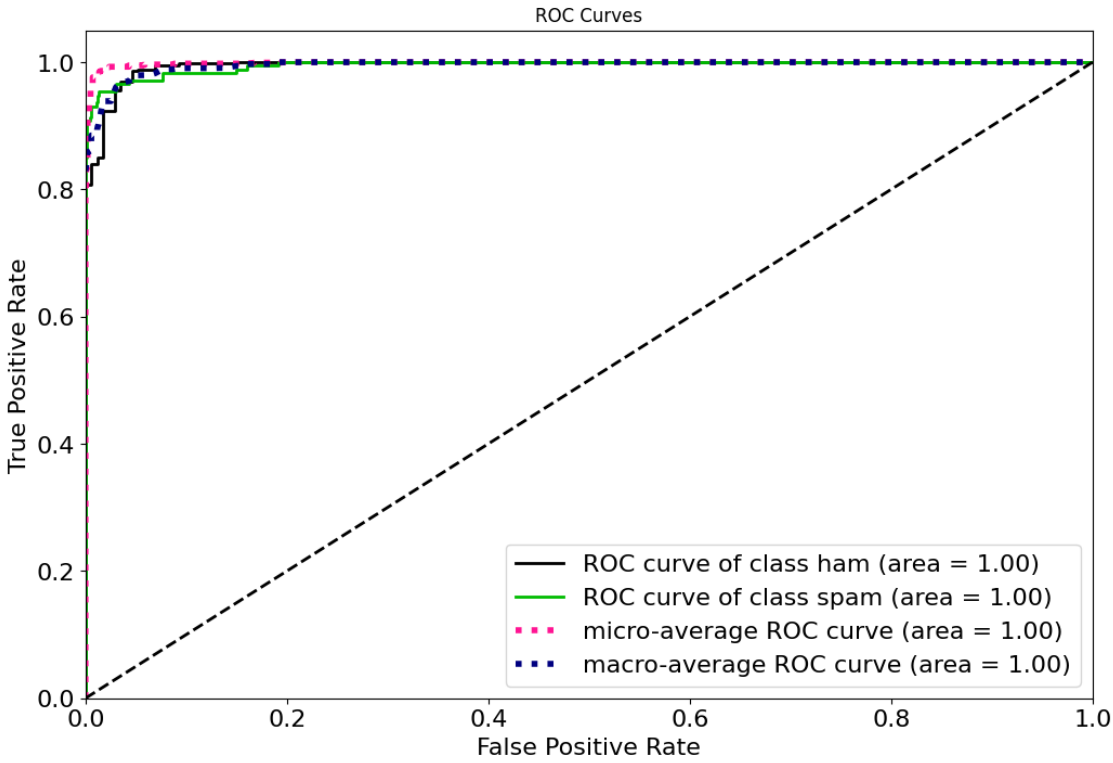


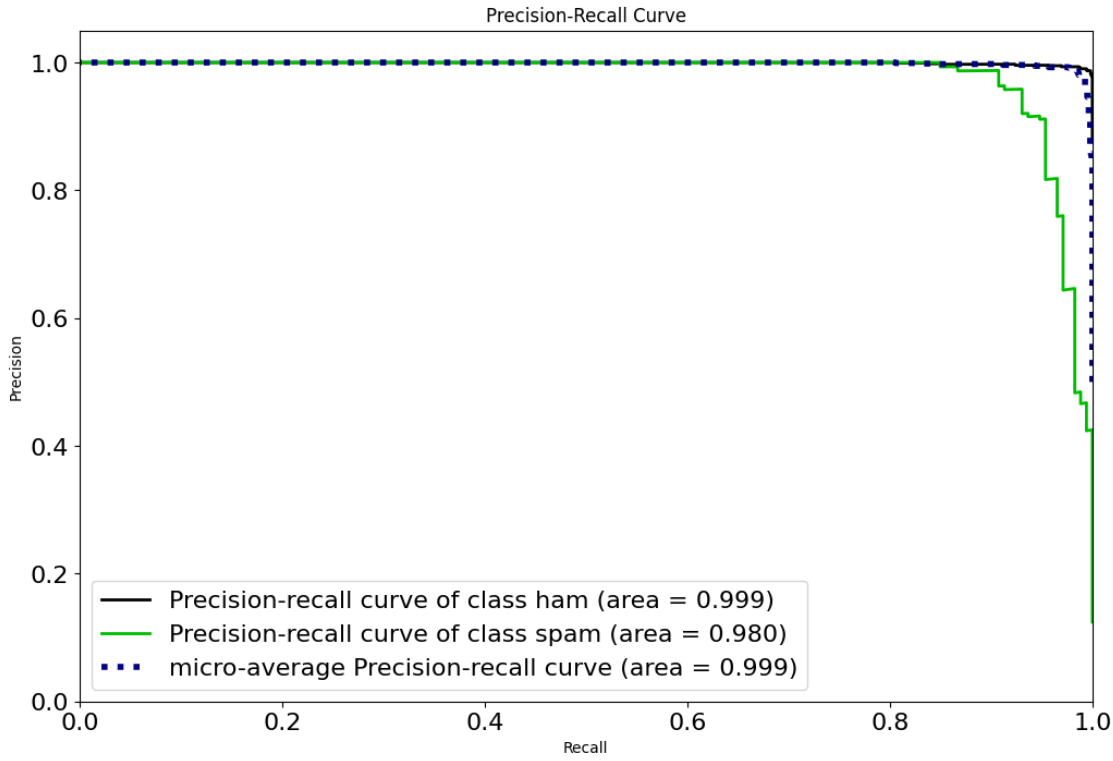



```
Confusion Matrix:
 [[1220    0]
 [ 137   36]]
Classification Report is:
              precision    recall  f1-score   support

         ham       0.90      1.00      0.95      1220
        spam       1.00      0.21      0.34       173

    accuracy                           0.90      1393
   macro avg       0.95      0.60      0.65      1393
weighted avg       0.91      0.90      0.87      1393

Accuracy: 90.1651112706389
Training Score:
 90.45226130653266
```

1.3 3. SVM

[255]:
```
from sklearn.svm import SVC
model = SVC(probability=True )
Model(model, X, y)
```

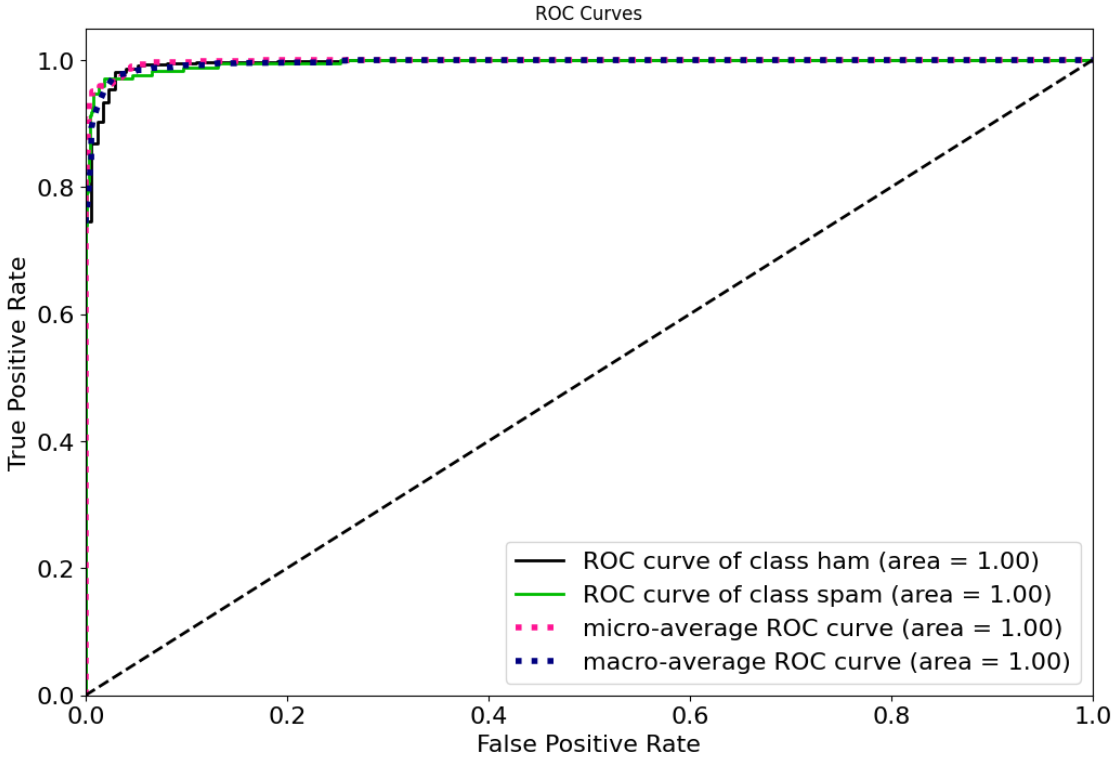


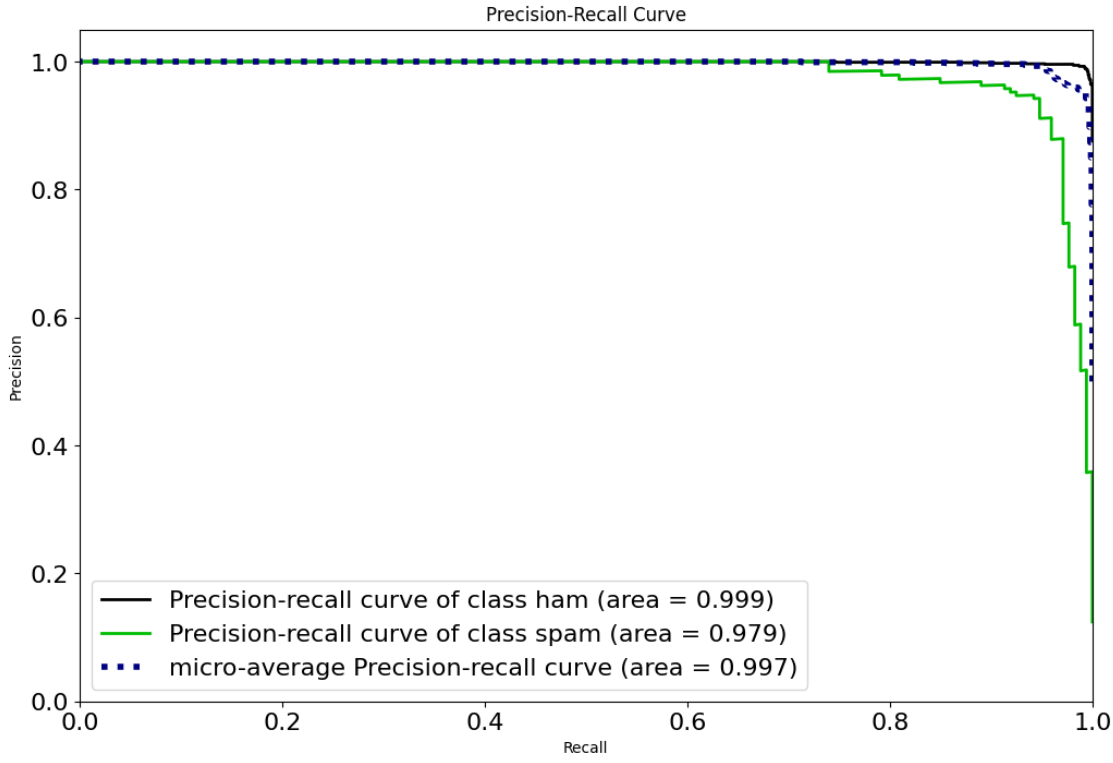



```
Confusion Matrix:
 [[1220    0]
 [  30  143]]
Classification Report is:
               precision    recall  f1-score   support

          ham       0.98      1.00      0.99      1220
         spam       1.00      0.83      0.91       173

    accuracy                           0.98      1393
   macro avg       0.99      0.91      0.95      1393
weighted avg       0.98      0.98      0.98      1393

Accuracy: 97.84637473079684
Training Score:
 99.83249581239531
```

1.4 4. Naive Bayes

[256]:
```
from sklearn.naive_bayes import BernoulliNB
model = BernoulliNB()
Model(model, X, y)
```

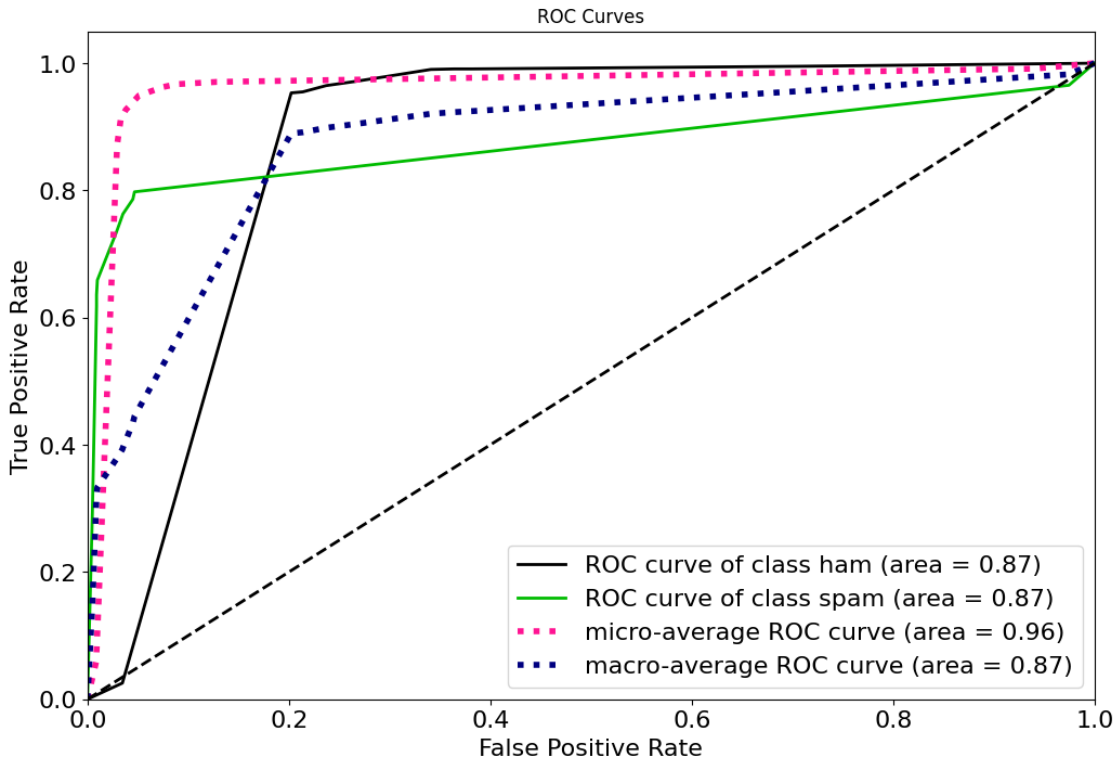

ROC Curves
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
ROC curve of class ham (area = 1.00)
ROC curve of class spam (area = 1.00)
micro-average ROC curve (area = 1.00)
macro-average ROC curve (area = 1.00)

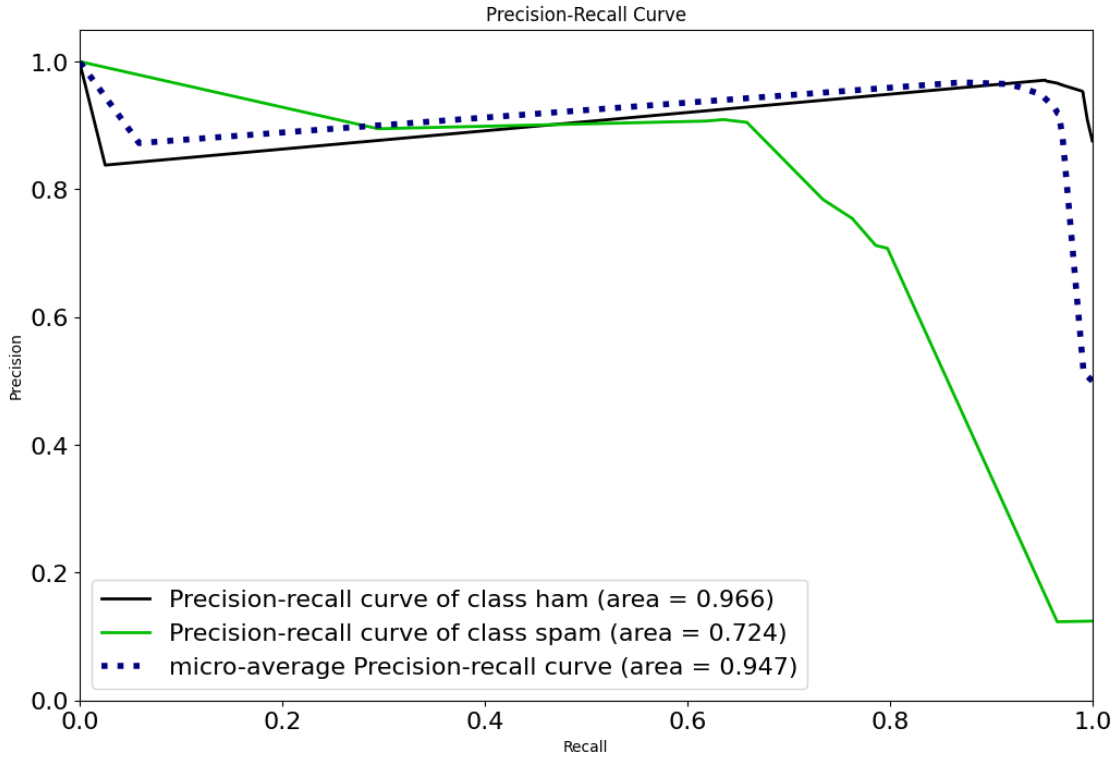

Precision-Recall Curve
Precision
Recall
0.0
0.2
0.4
0.6
0.8
1.0
Precision-recall curve of class ham (area = 0.999)
Precision-recall curve of class spam (area = 0.979)
micro-average Precision-recall curve (area = 0.997)

```
Confusion Matrix:
 [[1218    2]
 [  44  129]]
Classification Report is:
              precision    recall  f1-score   support

         ham       0.97      1.00      0.98      1220
        spam       0.98      0.75      0.85       173

    accuracy                           0.97      1393
   macro avg       0.97      0.87      0.92      1393
weighted avg       0.97      0.97      0.96      1393

Accuracy: 96.69777458722182
Training Score:
 98.82747068676717
```

1.5 5. DECISION TREE CLASSIFIER

[257]:
```
from sklearn import tree
tree_clf = tree.DecisionTreeClassifier(max_depth=6,␣
 ↪random_state=123,criterion='entropy')
Model(tree_clf,X,y)
```

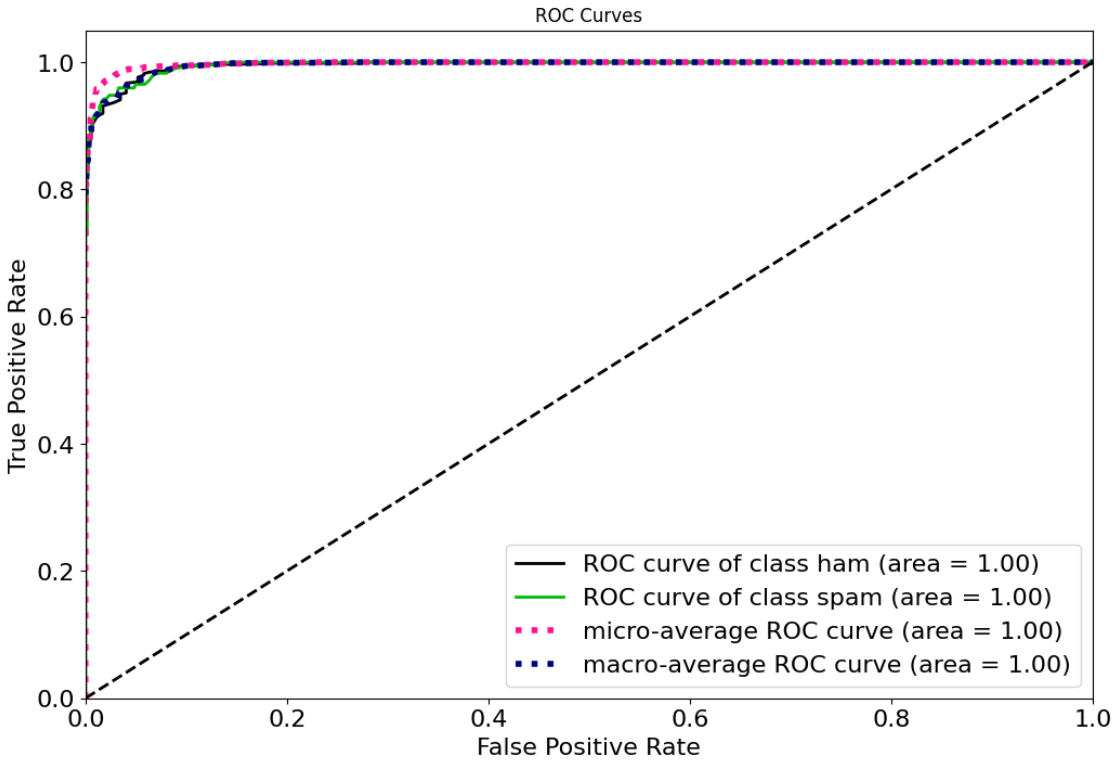


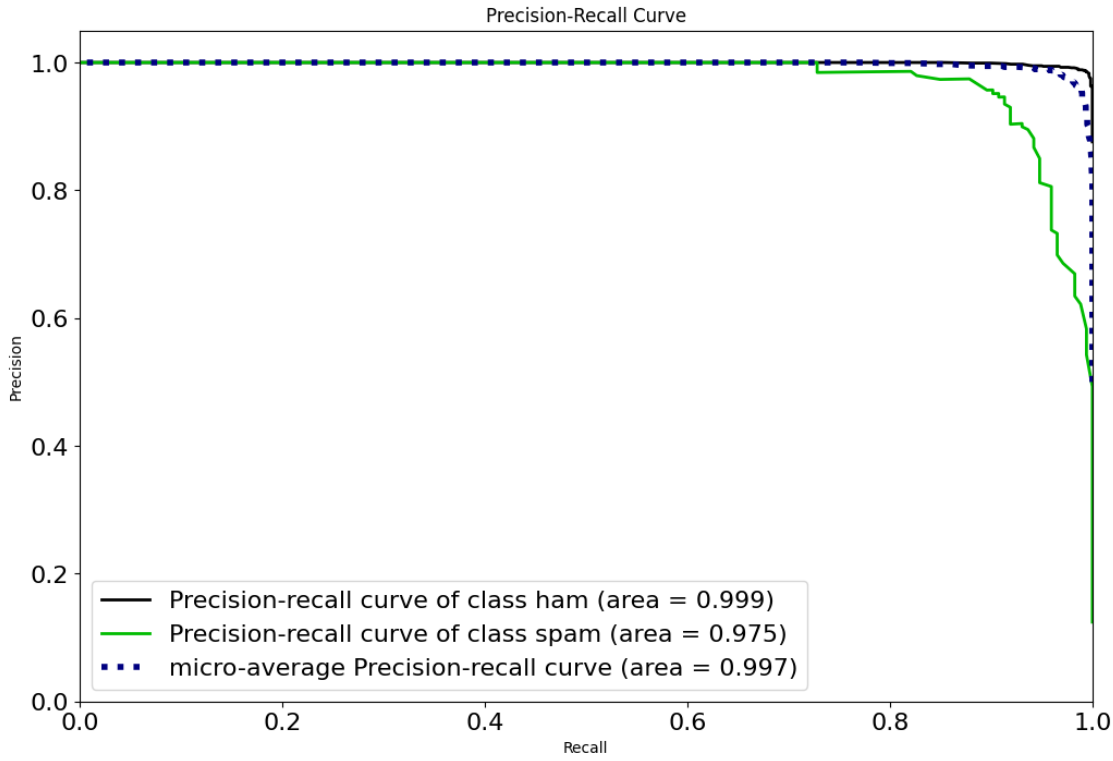



```
Confusion Matrix:
 [[1208   12]
 [  59  114]]
Classification Report is:
              precision    recall  f1-score   support

         ham       0.95      0.99      0.97      1220
        spam       0.90      0.66      0.76       173

    accuracy                           0.95      1393
   macro avg       0.93      0.82      0.87      1393
weighted avg       0.95      0.95      0.95      1393

Accuracy: 94.90308686288586
Training Score:
 96.21919119406557
```

1.6 6. RandomForestClassifier

[258]:
```
from sklearn.ensemble import RandomForestClassifier
model = RandomForestClassifier()
Model(model, X, y)
```






```
Confusion Matrix:
 [[1218    2]
 [  32  141]]
Classification Report is:
              precision    recall  f1-score   support

         ham       0.97      1.00      0.99      1220
        spam       0.99      0.82      0.89       173

    accuracy                           0.98      1393
   macro avg       0.98      0.91      0.94      1393
weighted avg       0.98      0.98      0.97      1393

Accuracy: 97.5592246949031
Training Score:
 100.0
```

We get a good accuracy score of 98 % using, Random Forest and SVM.