```
import pandas as pd
import numpy as np
%matplotlib inline
import matplotlib.pyplot as plt
import seaborn as sns
import string
from nltk.corpus import stopwords
import os
from sklearn.metrics import roc_auc_score
from matplotlib import pyplot
from sklearn.metrics import ConfusionMatrixDisplay
from wordcloud import WordCloud, STOPWORDS, ImageColorGenerator
from PIL import Image
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.naive_bayes import MultinomialNB
from sklearn.tree import DecisionTreeClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import roc_curve, auc
from sklearn import metrics
from sklearn import model_selection
from sklearn import svm
from nltk import word_tokenize
```

```
import nltk
nltk.download('punkt')
nltk.download('stopwords')
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
True
```

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

```
data  = pd.read_csv("mail_data.csv")
data.head()
```

	Category	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...

```
data.describe()
```

	Category	Message
count	5572	5572
unique	2	5157
top	ham	Sorry, I'll call later
freq	4825	30

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5572 entries, 0 to 5571
Data columns (total 2 columns):
 #   Column    Non-Null Count  Dtype 
---  ------    --------------  ----- 
 0   Category  5572 non-null   object
 1   Message   5572 non-null   object
dtypes: object(2)
memory usage: 87.2+ KB
```

```
data.isnull().sum()
```

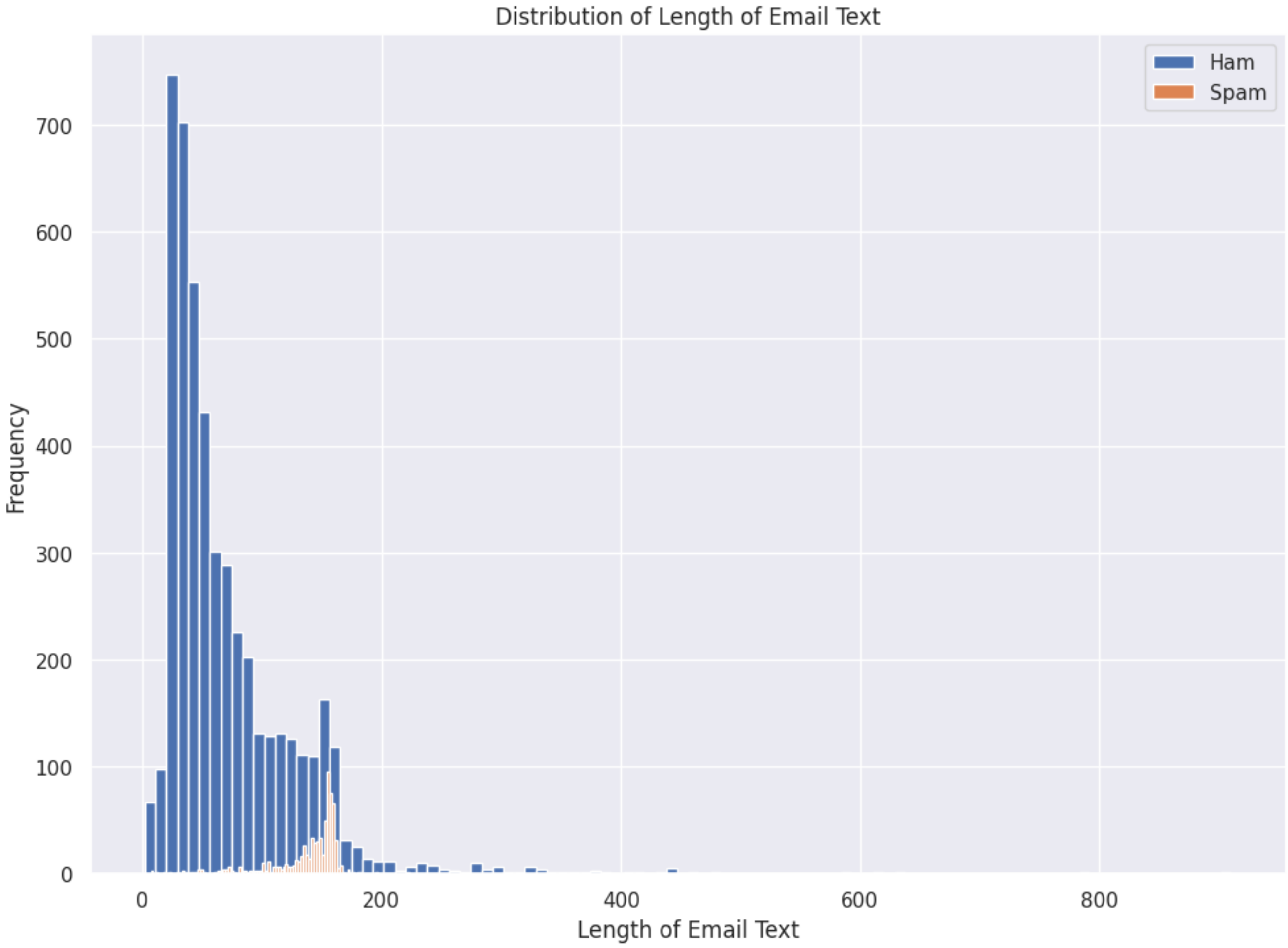
```
Category    0
Message     0
dtype: int64
```

```
data['Category']=data['Category'].replace({'ham': 0, 'spam': 1})
```

```
data['length']=data['Message'].apply(len)
data["length"].max()
```

```
910
```

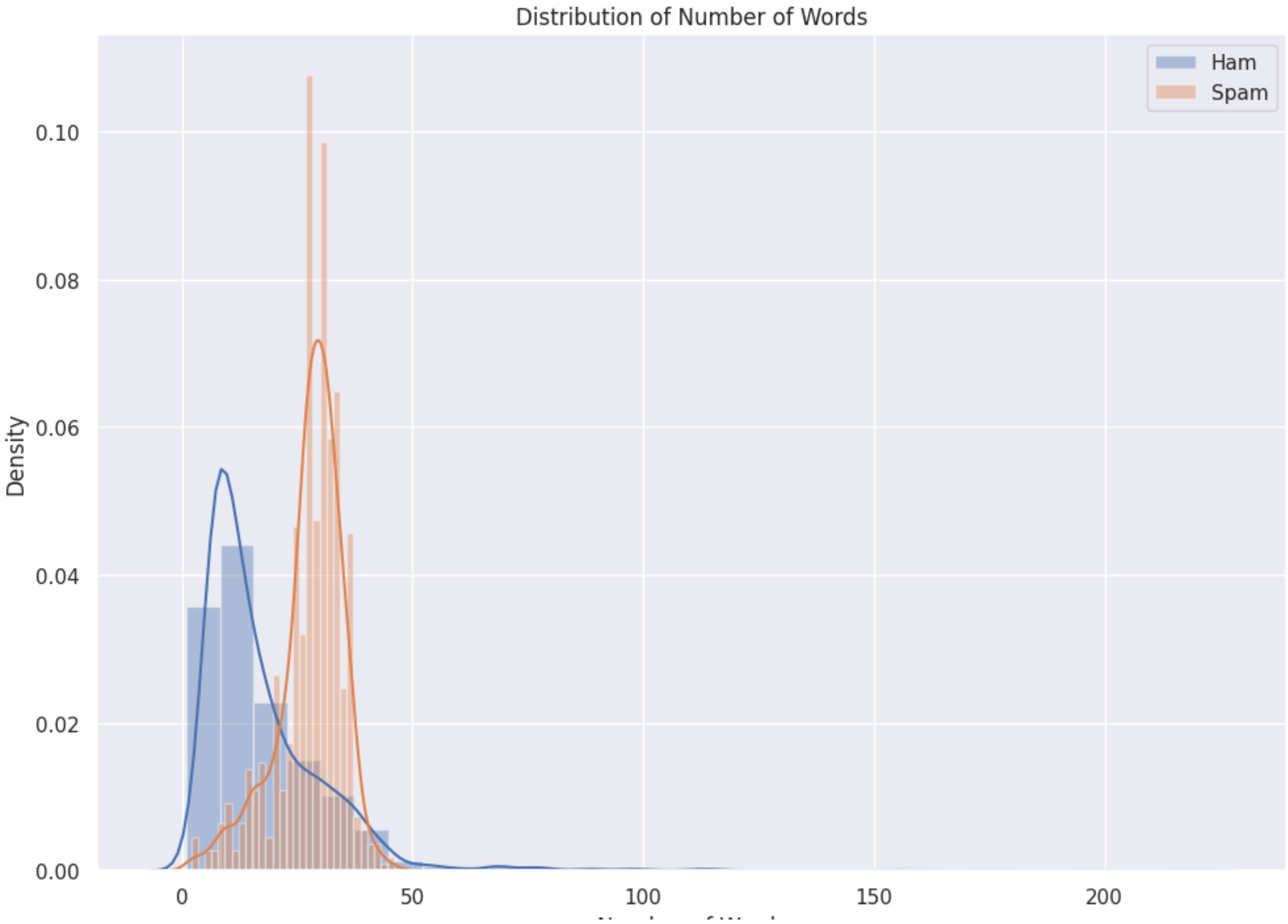
```
sns.set(rc={'figure.figsize':(11.7,8.27)})
ham_messages_length =  data[data['Category']==0]
spam_messages_length =  data[data['Category']==1]
ham_messages_length['length'].plot(bins=100, kind='hist',label = 'Ham')
spam_messages_length['length'].plot(bins=100, kind='hist',label = 'Spam')
plt.title('Distribution of Length of Email Text')
plt.xlabel('Length of Email Text')
plt.legend();
```



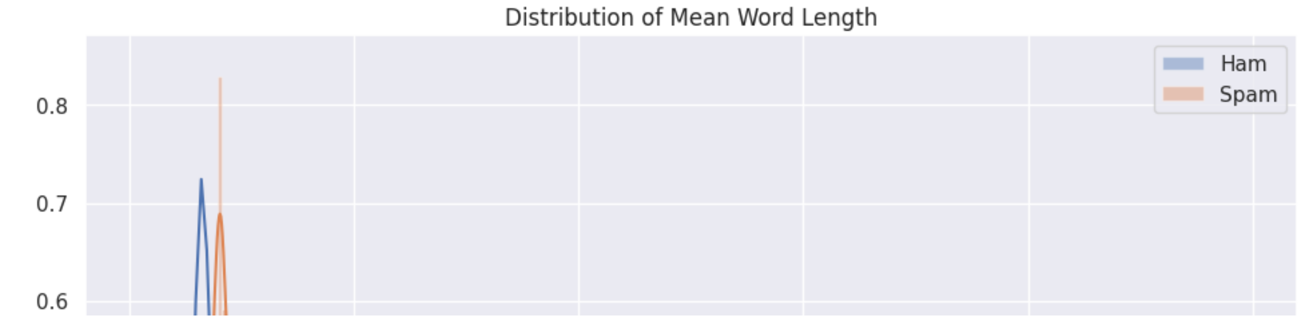
```
from nltk import word_tokenize
ham_words_length = [len(word_tokenize(title)) for title in data[data['Category']==0].Message.values]
spam_words_length = [len(word_tokenize(title)) for title in data[data['Category']==1].Message.values]
print(max(ham_words_length))
print(max(spam_words_length))
```

```
220
46
```

```
sns.set(rc={'figure.figsize':(11.7,8.27)})
ax = sns.distplot(ham_words_length, norm_hist = True, bins = 30, label = 'Ham')
ax = sns.distplot(spam_words_length, norm_hist = True, bins = 30, label = 'Spam')
plt.title('Distribution of Number of Words')
plt.xlabel('Number of Words')
plt.legend()

plt.show()
```

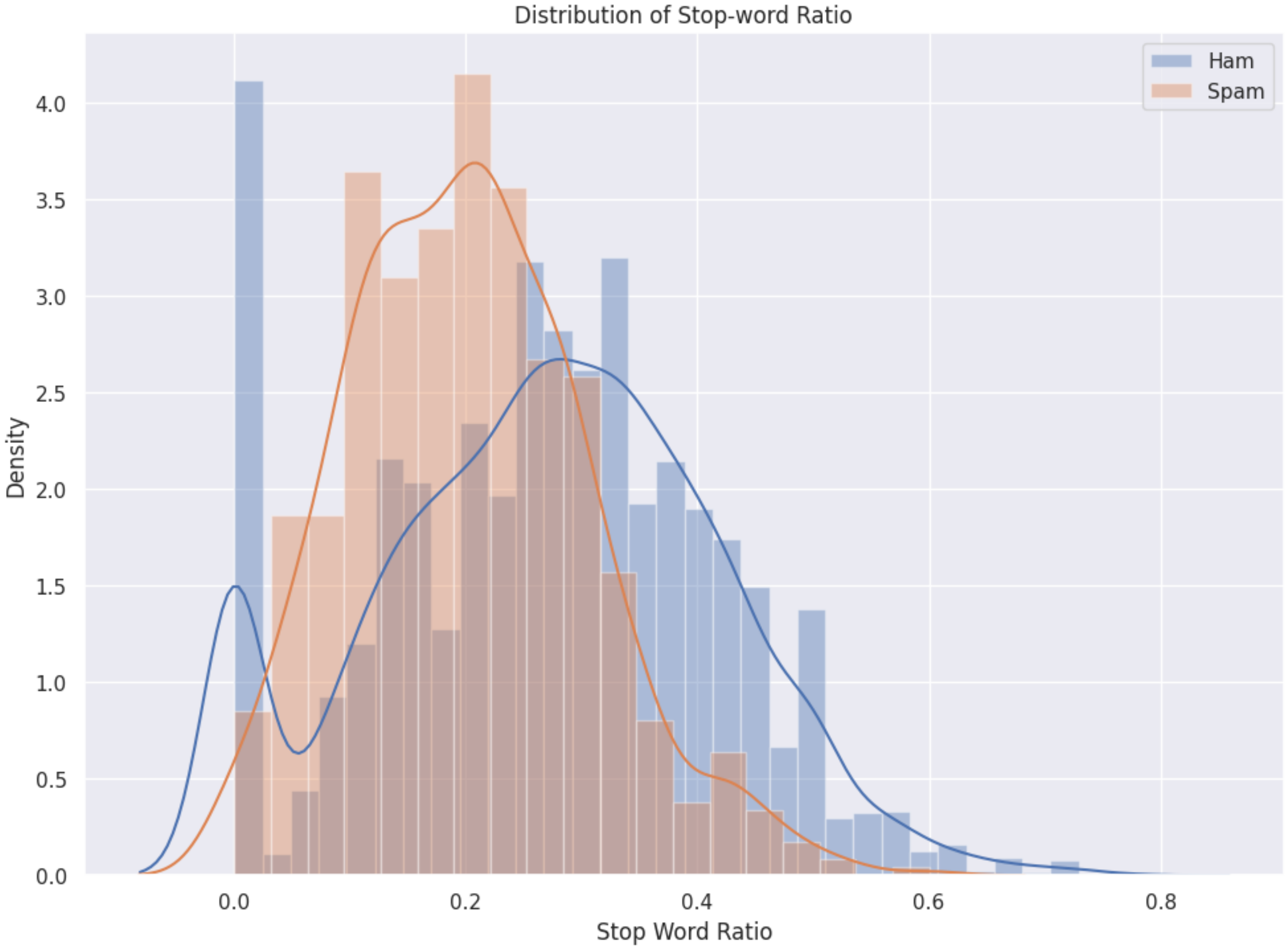


```
def mean_word_length(x):
    word_lengths = np.array([])
    for word in word_tokenize(x):
        word_lengths = np.append(word_lengths, len(word))
    return word_lengths.mean()

ham_meanword_length = data[data['Category']==0].Message.apply(mean_word_length)
spam_meanword_length = data[data['Category']==1].Message.apply(mean_word_length)


sns.distplot(ham_meanword_length, norm_hist = True, bins = 30, label = 'Ham')
sns.distplot(spam_meanword_length , norm_hist = True, bins = 30, label = 'Spam')
plt.title('Distribution of Mean Word Length')
plt.xlabel('Mean Word Length')
plt.legend()
plt.show()
```

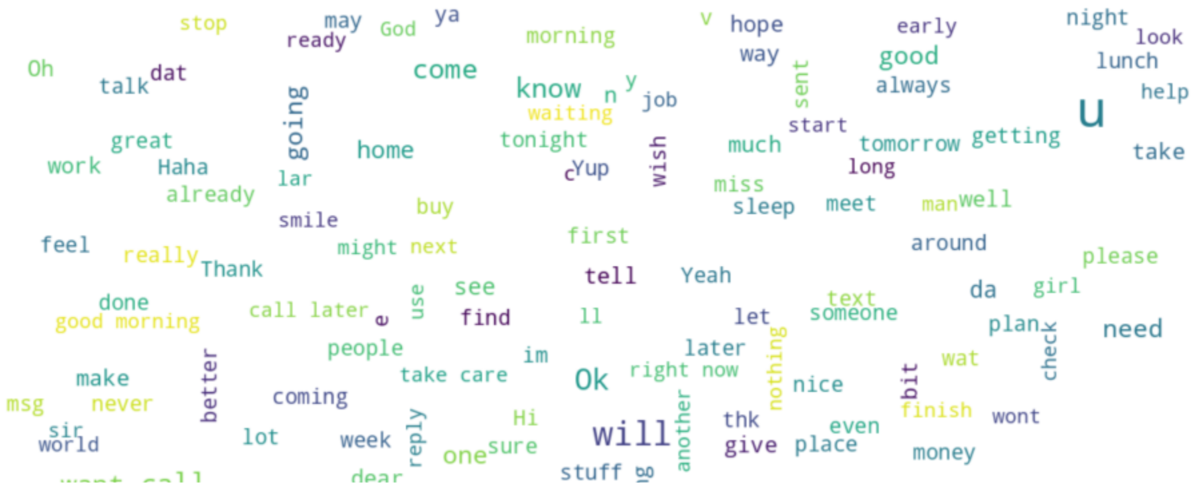


```
from nltk.corpus import stopwords
stop_words = set(stopwords.words('english'))


def stop_words_ratio(x):
    num_total_words = 0
    num_stop_words = 0
    for word in word_tokenize(x):
        if word in stop_words:
            num_stop_words += 1
        num_total_words += 1
    return num_stop_words/num_total_words

ham_stopwords = data[data['Category']==0].Message.apply(stop_words_ratio)
spam_stopwords = data[data['Category']==1].Message.apply(stop_words_ratio)


sns.distplot(ham_stopwords, norm_hist = True, label = 'Ham')
sns.distplot(spam_stopwords,  label = 'Spam')

print('Ham Mean: {:.3f}'.format(ham_stopwords.values.mean()))
print('Spam Mean: {:.3f}'.format(spam_stopwords.values.mean()))
plt.title('Distribution of Stop-word Ratio')
plt.xlabel('Stop Word Ratio')
plt.legend();
```

```
    Ham Mean: 0.268
    Spam Mean: 0.202
```



```
class data_read_write(object):
    def __init__(self):
        pass
    def __init__(self, file_link):
        self.data_frame =  pd.read_csv(file_link)
    def read_csv_file(self, file_link):
        return self.data_frame
    def write_to_csvfile(self, file_link):
        self.data_frame.to_csv(file_link, encoding='utf-8', index=False, header=True)
        return


class generate_word_cloud(data_read_write):
    def __init__(self):
        pass
    def variance_column(self, data):
        return variance(data)
    def word_cloud(self, data_frame_column, output_image_file):
        text = " ".join(review for review in data_frame_column)
        stopwords = set(STOPWORDS)
        stopwords.update(["subject"])
        wordcloud = WordCloud(width = 1200, height = 800, stopwords=stopwords, max_font_size = 50, margin=0, background_color = "white")
        plt.imshow(wordcloud, interpolation='bilinear')
        plt.axis("off")
        plt.show()
        wordcloud.to_file(output_image_file)
        return



ham = data[data['Category']==0]
spam = data[data['Category']==1]
word_cloud_obj = generate_word_cloud()
word_cloud_obj.word_cloud(ham["Message"], "ham_word_cloud.png")
word_cloud_obj.word_cloud(spam["Message"], "spam_word_cloud.png")
```



```
class data_cleaning(data_read_write):
    def __init__(self):
        pass
    def message_cleaning(self, message):
            Test_punc_removed = [char for char in message if char not in string.punctuation]
            Test_punc_removed_join = ''.join(Test_punc_removed)
            Test_punc_removed_join_clean = [word for word in Test_punc_removed_join.split() if word.lower() not in stopwords.words('engl
            final_join = ' '.join(Test_punc_removed_join_clean)
            return final_join


    def apply_to_column(self, data_column_text):
        data_processed = data_column_text.apply(self.message_cleaning)
        return data_processed
```

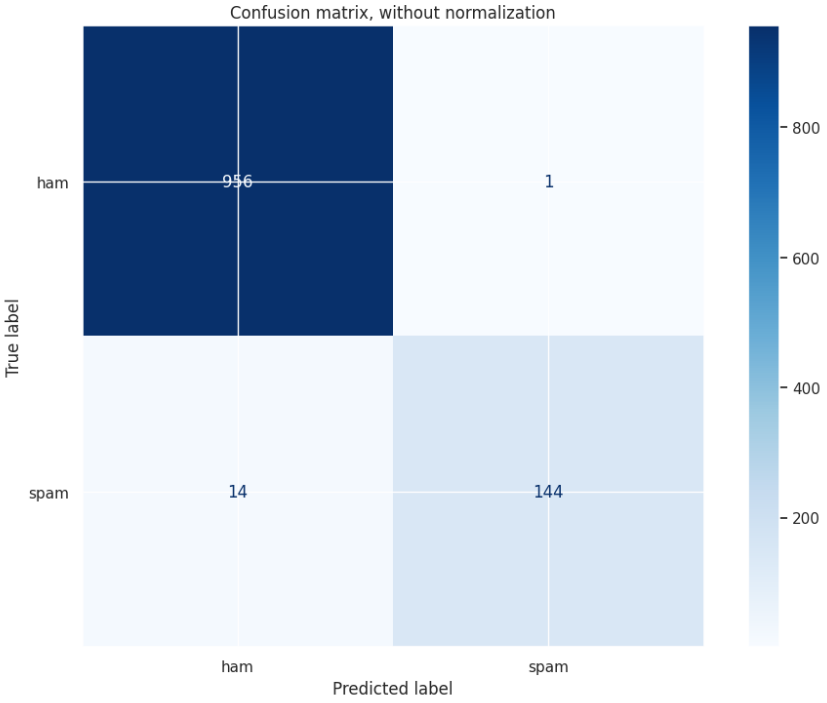
```
data_clean_obj = data_cleaning()
data['clean_text'] = data_clean_obj.apply_to_column(data['Message'])
```

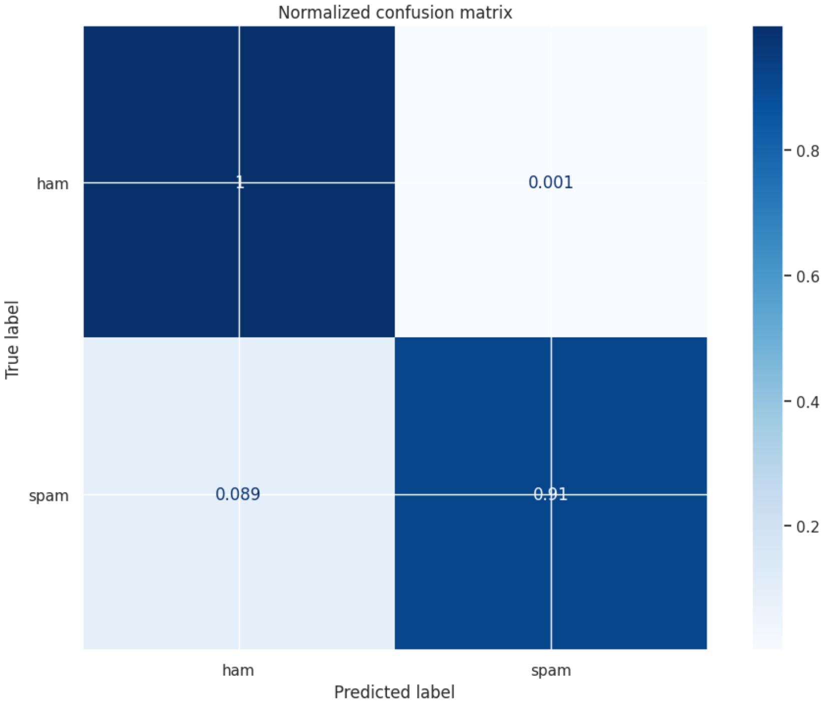
```
data.head()
```

	Category	Message	length	clean_text
0	0	Go until jurong point, crazy.. Available only ...	111	Go jurong point crazy Available bugis n great ...
1	0	Ok lar... Joking wif u oni...	29	Ok lar Joking wif u oni
2	1	Free entry in 2 a wkly comp to win FA Cup fina...	155	Free entry 2 wkly comp win FA Cup final tkts 2...
3	0	U dun say so early hor... U c already then say...	49	U dun say early hor U c already say
4	0	Nah I don't think he goes to usf, he lives aro...	61	Nah dont think goes usf lives around though

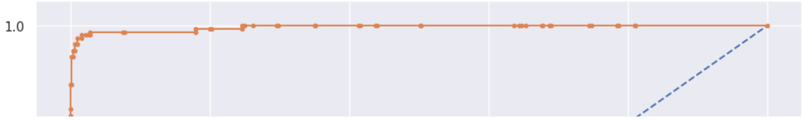
```
class apply_embeddding_and_model(data_read_write):
    def __init__(self):
        pass
    def apply_count_vector(self, v_data_column):
        vectorizer = CountVectorizer(min_df=2,analyzer = "word",tokenizer = None,preprocessor = None,stop_words = None)
        return vectorizer.fit_transform(v_data_column)

    def apply_naive_bayes(self, X, y):
        X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
        NB_classifier = MultinomialNB()
        NB_classifier.fit(X_train, y_train)
        y_predict_test = NB_classifier.predict(X_test)
        cm = confusion_matrix(y_test, y_predict_test)
        print(classification_report(y_test, y_predict_test))
        print("test set")

        print("\nAccuracy Score: " + str(metrics.accuracy_score(y_test, y_predict_test)))
        print("F1 Score: " + str(metrics.f1_score(y_test, y_predict_test)))
        print("Recall: " + str(metrics.recall_score(y_test, y_predict_test)))
        print("Precision: " + str(metrics.precision_score(y_test, y_predict_test)))

        class_names = ['ham', 'spam']
        titles_options = [("Confusion matrix, without normalization", None),
                  ("Normalized confusion matrix", 'true')]
        for title, normalize in titles_options:
            disp = ConfusionMatrixDisplay.from_estimator(NB_classifier, X_test, y_test,
                                 display_labels=class_names,
                                 cmap=plt.cm.Blues,
                                 normalize=normalize)
            disp.ax_.set_title(title)
            disp.ax_.set_title(title)
            print(title)
            print(disp.confusion_matrix)
        plt.show()
        ns_probs = [0 for _ in range(len(y_test))]
        lr_probs = NB_classifier.predict_proba(X_test)
        lr_probs = lr_probs[:, 1]
        ns_auc = roc_auc_score(y_test, ns_probs)
        lr_auc = roc_auc_score(y_test, lr_probs)
        print('No Skill: ROC AUC=%.3f' % (ns_auc))
        print('Naive Bayes: ROC AUC=%.3f' % (lr_auc))
        ns_fpr, ns_tpr, _ = roc_curve(y_test, ns_probs)
        lr_fpr, lr_tpr, _ = roc_curve(y_test, lr_probs)
        pyplot.plot(ns_fpr, ns_tpr, linestyle='--', label='No Skill')
        pyplot.plot(lr_fpr, lr_tpr, marker='.', label='Naive Bayes')
        pyplot.xlabel('False Positive Rate')
        pyplot.ylabel('True Positive Rate')
        pyplot.legend()
        pyplot.show()
        return
    def apply_svm(self, X, y):
        X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
        params = {'kernel': 'linear', 'C': 2, 'gamma': 1}
        svm_cv = svm.SVC(C=params['C'], kernel=params['kernel'], gamma=params['gamma'], probability=True)
        svm_cv.fit(X_train, y_train)
        y_predict_test = svm_cv.predict(X_test)
        cm = confusion_matrix(y_test, y_predict_test)
        print(classification_report(y_test, y_predict_test))
        print("test set")

        print("\nAccuracy Score: " + str(metrics.accuracy_score(y_test, y_predict_test)))
        print("F1 Score: " + str(metrics.f1_score(y_test, y_predict_test)))
        print("Recall: " + str(metrics.recall_score(y_test, y_predict_test)))
        print("Precision: " + str(metrics.precision_score(y_test, y_predict_test)))

        class_names = ['ham', 'spam']
        titles_options = [("Confusion matrix, without normalization", None),
                  ("Normalized confusion matrix", 'true')]
        for title, normalize in titles_options:
            disp = ConfusionMatrixDisplay.from_estimator(svm_cv, X_test, y_test,
                                 display_labels=class_names,
                                 cmap=plt.cm.Blues,
                                 normalize=normalize)
            disp.ax_.set_title(title)
            print(title)
            print(disp.confusion_matrix)
        plt.show()
        ns_probs = [0 for _ in range(len(y_test))]
        lr_probs = svm_cv.predict_proba(X_test)
        lr_probs = lr_probs[:, 1]
        ns_auc = roc_auc_score(y_test, ns_probs)
        lr_auc = roc_auc_score(y_test, lr_probs)
        print('No Skill: ROC AUC=%.3f' % (ns_auc))
        print('SVM: ROC AUC=%.3f' % (lr_auc))
        ns_fpr, ns_tpr, _ = roc_curve(y_test, ns_probs)
        lr_fpr, lr_tpr, _ = roc_curve(y_test, lr_probs)
        pyplot.plot(ns_fpr, ns_tpr, linestyle='--', label='No Skill')
        pyplot.plot(lr_fpr, lr_tpr, marker='.', label='SVM')
        pyplot.xlabel('False Positive Rate')
        pyplot.ylabel('True Positive Rate')
        pyplot.legend()
        pyplot.show()
        return
    def apply_decision_tree(self, X, y):
        X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
        dt_classifier = DecisionTreeClassifier()
        dt_classifier.fit(X_train, y_train)
        y_predict_test = dt_classifier.predict(X_test)
        cm = confusion_matrix(y_test, y_predict_test)
        print(classification_report(y_test, y_predict_test))
        print("test set")

        print("\nAccuracy Score: " + str(metrics.accuracy_score(y_test, y_predict_test)))
        print("F1 Score: " + str(metrics.f1_score(y_test, y_predict_test)))
        print("Recall: " + str(metrics.recall_score(y_test, y_predict_test)))
        print("Precision: " + str(metrics.precision_score(y_test, y_predict_test)))

        class_names = ['ham', 'spam']
        titles_options = [("Confusion matrix, without normalization", None),
                  ("Normalized confusion matrix", 'true')]
        for title, normalize in titles_options:
            disp = ConfusionMatrixDisplay.from_estimator(dt_classifier, X_test, y_test,
                                 display_labels=class_names,
                                 cmap=plt.cm.Blues,
                                 normalize=normalize)
            disp.ax_.set_title(title)
            print(title)
            print(disp.confusion_matrix)
        plt.show()
        ns_probs = [0 for _ in range(len(y_test))]
        lr_probs = dt_classifier.predict_proba(X_test)
        lr_probs = lr_probs[:, 1]
        ns_auc = roc_auc_score(y_test, ns_probs)
        lr_auc = roc_auc_score(y_test, lr_probs)
        print('No Skill: ROC AUC=%.3f' % (ns_auc))
        print('Decision Tree: ROC AUC=%.3f' % (lr_auc))
        ns_fpr, ns_tpr, _ = roc_curve(y_test, ns_probs)
        lr_fpr, lr_tpr, _ = roc_curve(y_test, lr_probs)
        pyplot.plot(ns_fpr, ns_tpr, linestyle='--', label='No Skill')
        pyplot.plot(lr_fpr, lr_tpr, marker='.', label='Decision Tree')
        pyplot.xlabel('False Positive Rate')
        pyplot.ylabel('True Positive Rate')
        pyplot.legend()
        pyplot.show()
        return
    def apply_logistic_regression(self, X, y):
        X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
        lr_classifier = LogisticRegression()
        lr_classifier.fit(X_train, y_train)
        y_predict_test = lr_classifier.predict(X_test)
        cm = confusion_matrix(y_test, y_predict_test)
        print(classification_report(y_test, y_predict_test))
        print("test set")

        print("\nAccuracy Score: " + str(metrics.accuracy_score(y_test, y_predict_test)))
        print("F1 Score: " + str(metrics.f1_score(y_test, y_predict_test)))
        print("Recall: " + str(metrics.recall_score(y_test, y_predict_test)))
        print("Precision: " + str(metrics.precision_score(y_test, y_predict_test)))

        class_names = ['ham', 'spam']
        titles_options = [("Confusion matrix, without normalization", None),
                  ("Normalized confusion matrix", 'true')]
        for title, normalize in titles_options:
            disp = ConfusionMatrixDisplay.from_estimator(lr_classifier, X_test, y_test,
                                 display_labels=class_names,
                                 cmap=plt.cm.Blues,
                                 normalize=normalize)
            disp.ax_.set_title(title)
            print(title)
            print(disp.confusion_matrix)
        plt.show()
        ns_probs = [0 for _ in range(len(y_test))]
        lr_probs = lr_classifier.predict_proba(X_test)
        lr_probs = lr_probs[:, 1]
        ns_auc = roc_auc_score(y_test, ns_probs)
        lr_auc = roc_auc_score(y_test, lr_probs)
        print('No Skill: ROC AUC=%.3f' % (ns_auc))
        print('Logistic Regression: ROC AUC=%.3f' % (lr_auc))
        ns_fpr, ns_tpr, _ = roc_curve(y_test, ns_probs)
        lr_fpr, lr_tpr, _ = roc_curve(y_test, lr_probs)
        pyplot.plot(ns_fpr, ns_tpr, linestyle='--', label='No Skill')
        pyplot.plot(lr_fpr, lr_tpr, marker='.', label='Logistic Regression')
        pyplot.xlabel('False Positive Rate')
        pyplot.ylabel('True Positive Rate')
        pyplot.legend()
        pyplot.show()
        return


cv_object = apply_embeddding_and_model()
spamham_countvectorizer = cv_object.apply_count_vector(data['clean_text'])


X = spamham_countvectorizer
label = data['Category'].values
y = label


cv_object.apply_logistic_regression(X,y)
```

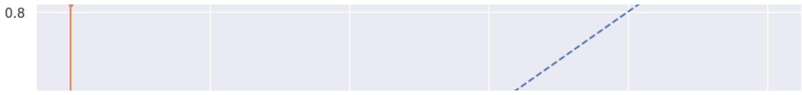
```
              precision    recall  f1-score   support

           0       0.99      1.00      0.99       957
           1       0.99      0.91      0.95       158

    accuracy                           0.99      1115
   macro avg       0.99      0.96      0.97      1115
weighted avg       0.99      0.99      0.99      1115

test set

Accuracy Score: 0.9865470852017937
F1 Score: 0.9504950495049505
Recall: 0.9113924050632911
Precision: 0.993103448275862
Confusion matrix, without normalization
[[956   1]
 [ 14 144]]
Normalized confusion matrix
[[0.99895507 0.00104493]
 [0.08860759 0.91139241]]
```





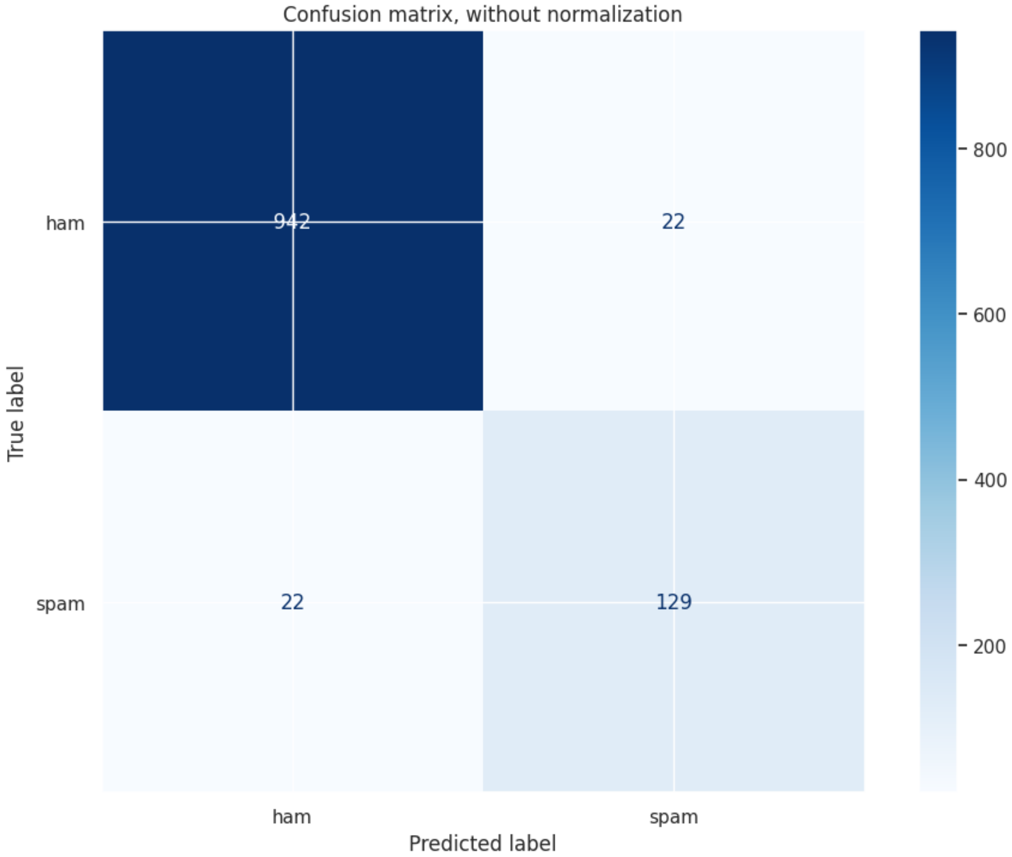
```
No Skill: ROC AUC=0.500
Logistic Regression: ROC AUC=0.997
```

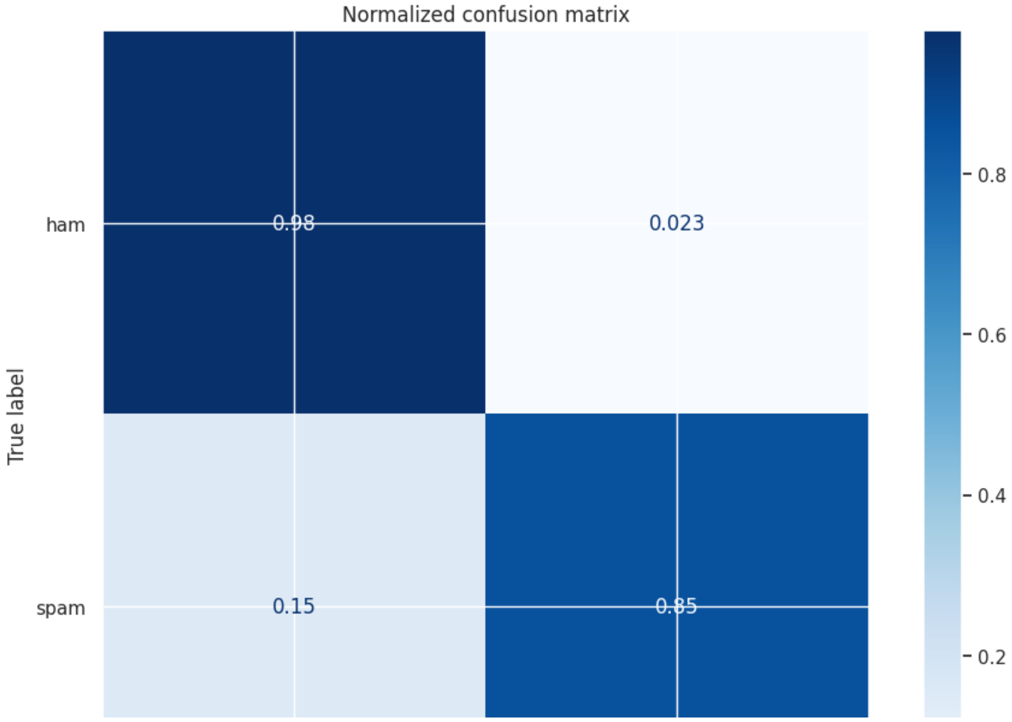




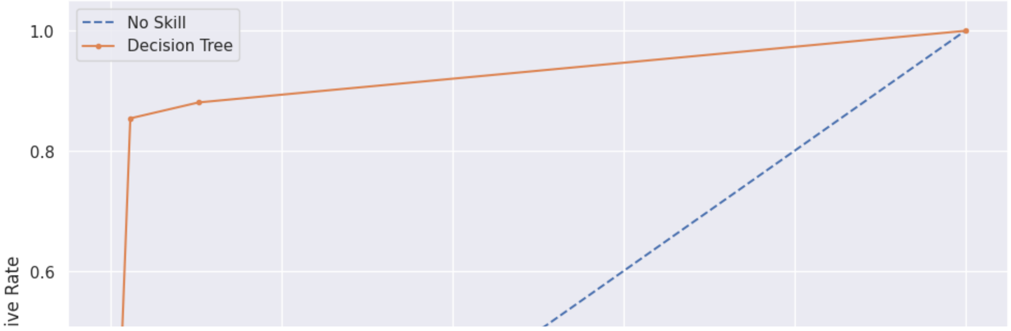
```
cv_object.apply_decision_tree(X,y)
```

```
              precision    recall  f1-score   support

           0       0.98      0.98      0.98       964
           1       0.85      0.85      0.85       151

    accuracy                           0.96      1115
   macro avg       0.92      0.92      0.92      1115
weighted avg       0.96      0.96      0.96      1115

test set

Accuracy Score: 0.9605381165919282
F1 Score: 0.8543046357615893
Recall: 0.8543046357615894
Precision: 0.8543046357615894
Confusion matrix, without normalization
[[942  22]
 [ 22 129]]
Normalized confusion matrix
[[0.97717842 0.02282158]
 [0.14569536 0.85430464]]
```



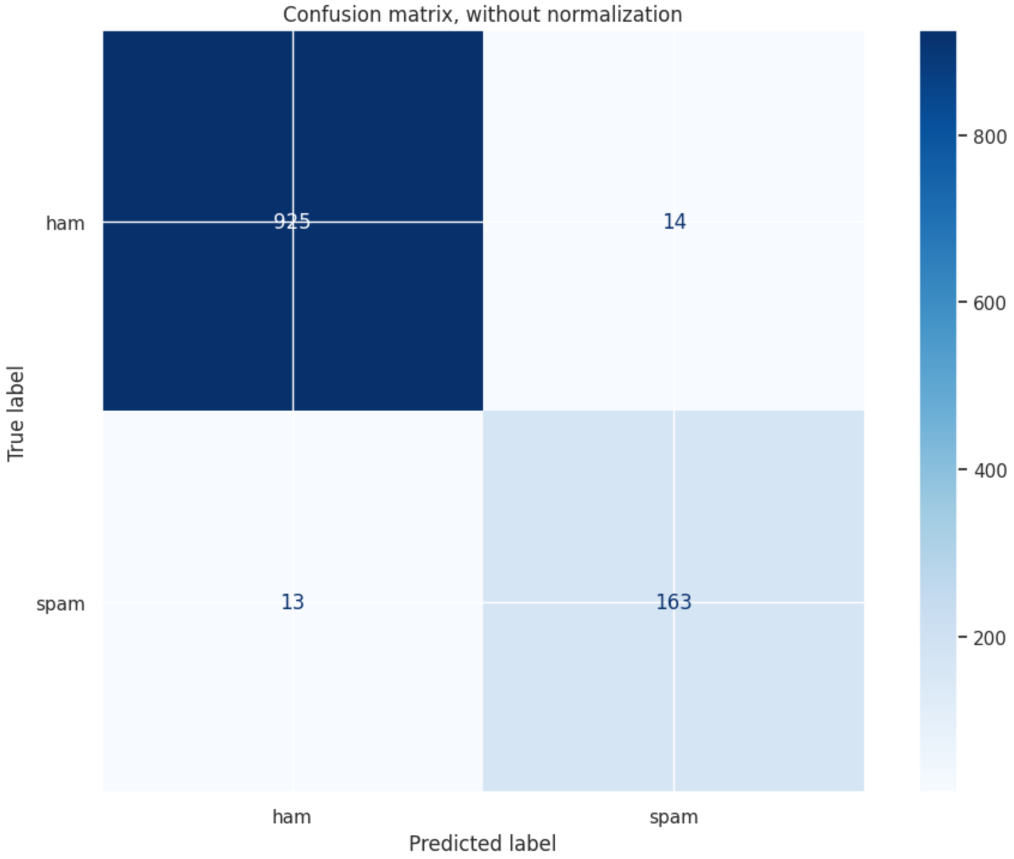


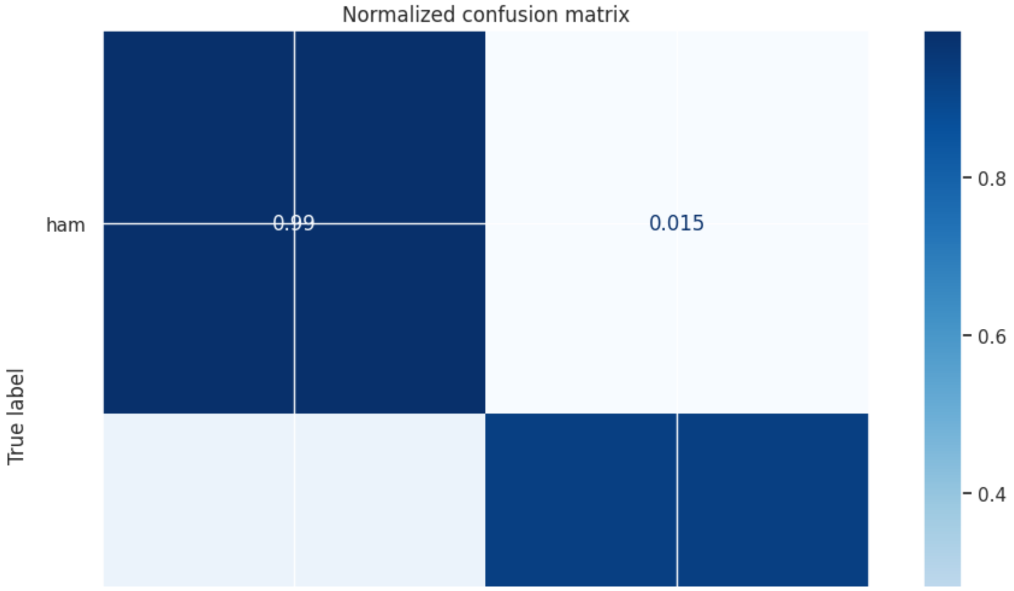
```
No Skill: ROC AUC=0.500
Decision Tree: ROC AUC=0.923
```



```
cv_object.apply_naive_bayes(X,y)
```

```
              precision    recall  f1-score   support

           0       0.99      0.99      0.99       939
           1       0.92      0.93      0.92       176

    accuracy                           0.98      1115
   macro avg       0.95      0.96      0.95      1115
weighted avg       0.98      0.98      0.98      1115

test set

Accuracy Score: 0.9757847533632287
F1 Score: 0.9235127478753541
Recall: 0.9261363636363636
Precision: 0.9209039548022598
Confusion matrix, without normalization
[[925  14]
 [ 13 163]]
Normalized confusion matrix
[[0.98509052 0.01490948]
 [0.07386364 0.92613636]]
```





```
cv_object.apply_svm(X,y)
```