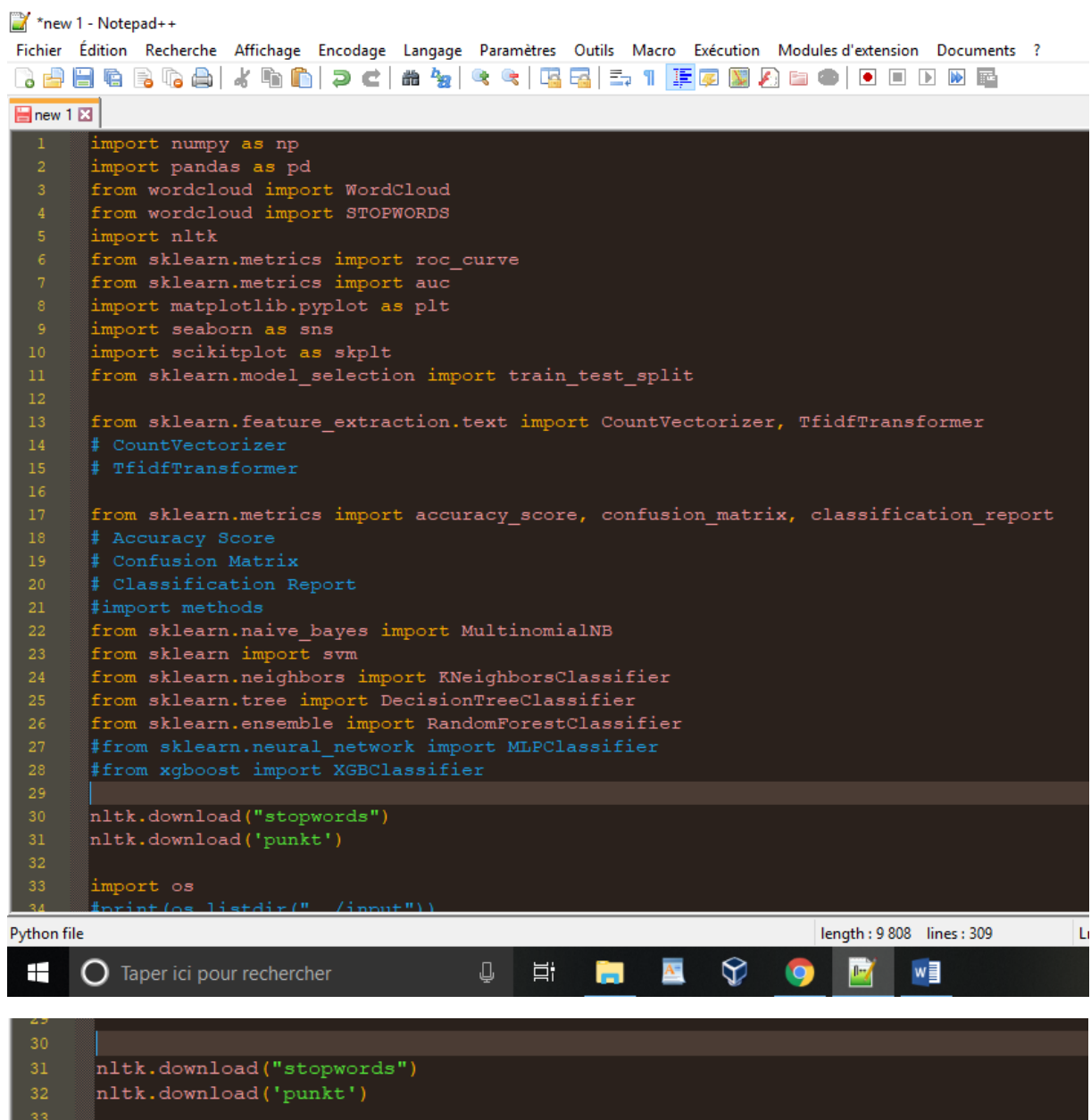


Spam Project of Machine Learning:

Members:

- **Edna Batana Diane**
- **Loukrimi Souad**
- **Zemmouri Kanza**



```
*new 1 - Notepad++
Fichier  Édition  Recherche  Affichage  Encodage  Langage  Paramètres  Outils  Macro  Exécution  Modules d'extension  Documents  ?

new 1 x
1  import numpy as np
2  import pandas as pd
3  from wordcloud import WordCloud
4  from wordcloud import STOPWORDS
5  import nltk
6  from sklearn.metrics import roc_curve
7  from sklearn.metrics import auc
8  import matplotlib.pyplot as plt
9  import seaborn as sns
10 import scikitplot as skplt
11 from sklearn.model_selection import train_test_split
12
13 from sklearn.feature_extraction.text import CountVectorizer, TfidfTransformer
14 # CountVectorizer
15 # TfidfTransformer
16
17 from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
18 # Accuracy Score
19 # Confusion Matrix
20 # Classification Report
21 #import methods
22 from sklearn.naive_bayes import MultinomialNB
23 from sklearn import svm
24 from sklearn.neighbors import KNeighborsClassifier
25 from sklearn.tree import DecisionTreeClassifier
26 from sklearn.ensemble import RandomForestClassifier
27 #from sklearn.neural_network import MLPClassifier
28 #from xgboost import XGBClassifier
29
30 nltk.download("stopwords")
31 nltk.download('punkt')
32
33 import os
34 #print(os.listdir("/input"))

Python file  length : 9 808  lines : 309  L
Taper ici pour rechercher
```

```

33
34 import os
35 #print(os.listdir("../input"))
36

```

Read and Show Data :

```

38
39
40 # ##### Read and show Data:
41
42 import urllib.request
43 import os
44
45 !gdown https://drive.google.com/uc?id=1XZcHqgnP_rvY7F6-B2ZuA_6w5ReFGV5H
46 df = pd.read_table('SMSSpamCollection',
47                   sep='\t',
48                   header=None,
49                   names=['Label', 'Message'])
50 df.head()
51
52

```

	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...

Splitting the labels and the data separately:

```

77
78 # ##### Splitting the labels and the data separately :
79 df_labels = df['Label']
80 df_labels.head(11)
81

```

```

0      ham
1      ham
2      spam
3      ham
4      ham
5      spam
6      ham
7      ham
8      spam
9      spam
10     ham
Name: Label, dtype: object

```

Data Visualization :

- To check the most used word in Ham sms and Spam SMS
- To visualize the percentage of Ham and Spam SMS

```

new 1 x
84 import nltk
85 nltk.download('stopwords')
86 nltk.download('punkt')
87 stopwords = STOPWORDS
88 stopwords = list(stopwords)
89 STOPWORDS = nltk.corpus.stopwords.words('english')
90 stopwords = stopwords + STOPWORDS
91 ham_dataset = df[df.Label == 'ham']
92 spam_dataset = df[df.Label == 'spam']
93 ham_words = ' '
94 spam_words = ' '
95
96 for words in ham_dataset.Message:
97     txt = words.lower()
98     tokens = nltk.word_tokenize(txt)
99     for word in tokens:
100         ham_words = ham_words + word + " "
101 for words in spam_dataset.Message:
102     txt = words.lower()
103     tokens = nltk.word_tokenize(txt)
104     for word in tokens:
105         spam_words = spam_words + word + " "
106
107 def gen_wordcloud(wordcloud):
108     plt.figure(figsize = (10,8))
109     plt.imshow(wordcloud)
110     plt.tight_layout(pad=0)
111     plt.axis('off')
112     plt.show()
113 print("\n")
114 print("\t\t\t\t\t HAM WORDS")
115 wordcloud = WordCloud(background_color = 'white', width = 500, height = 500, stopwords = stopwords,
116                       max_words = 500, max_font_size = 50, random_state = 42).generate(ham_words)
117 gen_wordcloud(wordcloud)
118 print("\t\t\t\t\t SPAM WORDS")
119 wordcloud = WordCloud(background_color = 'white', width = 500, height = 500, stopwords = stopwords,
120                       max_words = 500, max_font_size = 50, random_state = 42).generate(spam_words)
121 gen_wordcloud(wordcloud)

```

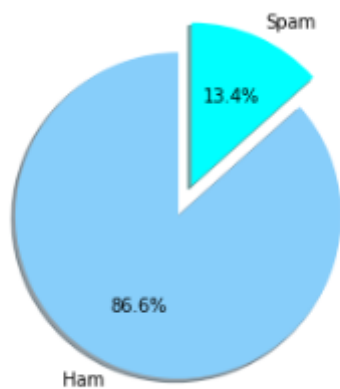
Python file | length : 8 784 | lines : 291 | Ln : 101

Taper ici pour rechercher

A word cloud of text messages, with 'free', 'mobile', 'call', 'text', 'message', 'reply', 'win', 'prize', 'claim', 'code', 'urgent', 'customer', 'service', 'call', 'line', 'chat', 'send', 'stop', 'go', 'free', 'text', 'guaranteed', 'call', 'xmas', 'points', 'chance', 'win', 'speciality', 'selected', 'auction'. The words are arranged in a dense, overlapping manner, with some words appearing larger than others. The colors are primarily green, yellow, and orange, with some blue and red accents. The background is black. The words are in various orientations, mostly horizontal but some rotated. The font is a sans-serif typeface. The overall impression is one of a busy, cluttered communication stream.

Plotting ham and spam data % in pie chart:

```
145 # ##### Plotting ham and spam data % in pie chart :
146
147
148 count_Class = pd.value_counts(df.Label, sort = True)
149
150 # Data to Plot
151 labels = 'Ham', 'Spam'
152 sizes = [count_Class[0], count_Class[1]]
153 colors = ['lightskyblue', 'aqua']
154 explode = (0.1, 0.1)
155
156 # Plot
157 plt.pie(sizes, explode = explode, labels = labels, colors = colors,
158         autopct = '%1.1f%%', shadow = True, startangle = 90)
159 plt.axis('equal')
160 plt.show()
```



Splitting the Test and Train Data:

```
165
166 # ##### Splitting the Test and Train Data:
167
168 train_set, test_set, train_label, test_label = train_test_split(df, df_labels, test_size = 0.33, random_state = 42)
169 print(train_set.shape)
170 print(test_set.shape)
171 print("\nThe Trainset consists of {} records and {} features".format(train_set.shape[0],train_set.shape[1]))
172 print("\nThe Testset consists of {} records and {} features".format(test_set.shape[0],train_set.shape[1]))
173
174
175
```

(3733, 2)
(1839, 2)

The Trainset consists of 3733 records and 2 features

The Testset consists of 1839 records and 2 features

Extracting N-grams from the Text Data:

```
176
177 # ##### Extracting N-grams from the Text Data:
178
179 countvect = CountVectorizer(ngram_range = (2,2), )
180 x_counts = countvect.fit(train_set.Message)
181
182 # preparing for training set
183 x_train_df = countvect.transform(train_set.Message)
184
185 # preparing for test set
186 x_test_df = countvect.transform(test_set.Message)
187
188
```

Data Model:

The Algorithms used below are:

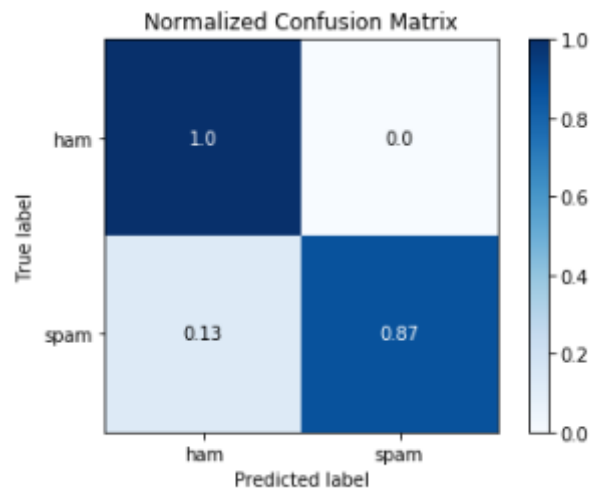
- Naive Bayes
- K-Nearest
- Decision Tree
- Support Vector Machine
- Random Forest

Naive Bayes classifier :

```
190
191 # ##### Naive Bayes classifier :
192
193
194 clf = MultinomialNB()
195 clf.fit(x_train_df, train_set.Label)
196 predicted_values_NB = clf.predict(x_test_df)
197 predictions = dict()
198 accuracy = accuracy_score(test_set.Label, predicted_values_NB)
199 predictions['Naive Bayes'] = accuracy * 100
200 confusionmatrix = confusion_matrix(test_set.Label, predicted_values_NB)
201 print("The accuracy of Naive Bayes classifier is {}".format(accuracy * 100))
202 print("\n", confusionmatrix)
203 skplt.metrics.plot_confusion_matrix(test_set.Label, predicted_values_NB, normalize = True)
204 plt.show()
205
```

The accuracy of Naive Bayes clasifier is 97.87928221859707%

```
[[1581    6]
 [33    219]]
```



K-Nearest Neighbors algorithm :

```

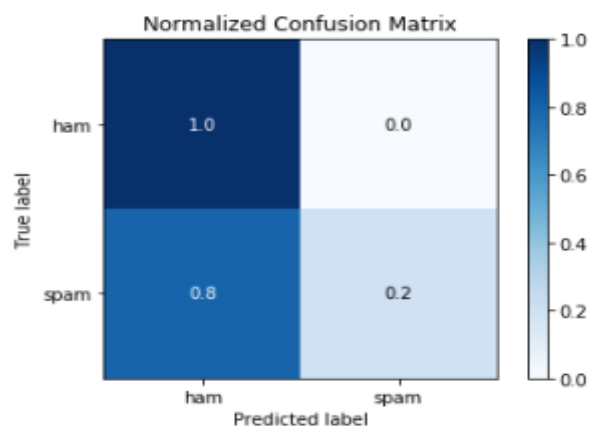
209 #KNN = KNeighborsClassifier(metric = 'euclidean')
210 KNN = KNeighborsClassifier()
211 KNN.fit(x_train_df, train_set.Label)
212 predicted_values_KNN = KNN.predict(x_test_df)
213 print(predicted_values_KNN)
214 accuracy_KNN = accuracy_score(test_set.Label, predicted_values_KNN)
215 predictions['K-Nearest Neighbors algorithm'] = accuracy_KNN * 100
216 print("\nThe accuracy of K-Nearest Neighbors algorithm is {}".format(accuracy_KNN * 100))
217 confusion_matrix_KNN = confusion_matrix(test_set.Label, predicted_values_KNN)
218 print("\n", confusion_matrix_KNN)
219 skplt.metrics.plot_confusion_matrix(test_set.Label, predicted_values_KNN, normalize = True)
220 plt.show()
221

```

```
['ham' 'ham' 'ham' ... 'ham' 'ham' 'ham']
```

The accuracy of K-Nearest Neighbors algorithm is 89.07014681892332%

```
[[1587    0]
 [201    51]]
```



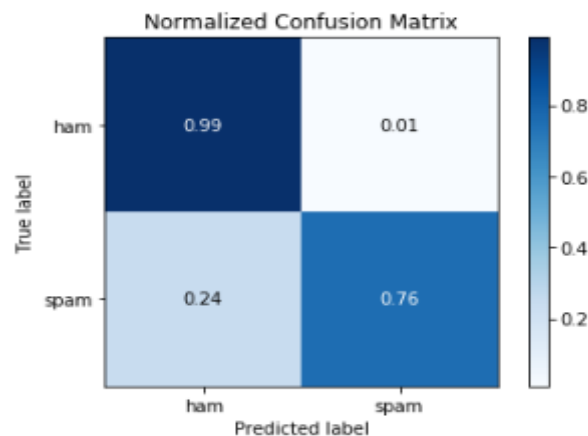
Decision Tree learning :

```
221
222 # #### Decision Tree learning :
223
224 DT = DecisionTreeClassifier()
225 DT.fit(x_train_df, train_set.Label)
226 predicted_values_DT = DT.predict(x_test_df)
227 print(predicted_values_DT)
228 accuracy_DT = accuracy_score(test_set.Label, predicted_values_DT)
229 predictions['Decision Tree learning'] = accuracy_DT * 100
230 print("\nThe accuracy of Decision Tree learning is {}".format(accuracy_DT * 100))
231 confusion_matrix_DT = confusion_matrix(test_set.Label, predicted_values_DT)
232 print("\n", confusion_matrix_DT)
233 skplt.metrics.plot_confusion_matrix(test_set.Label, predicted_values_DT, normalize = True)
234 plt.show()
235
236
```

['ham' 'ham' 'spam' ... 'ham' 'ham' 'spam'] The accuracy of Decision Tree learning is 96.08482871125612%

```
[[1576 11]
```

```
[ 61 191]]
```



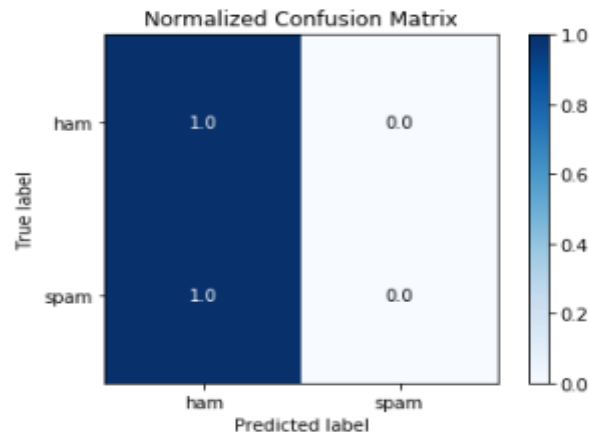
Support Vector Machine (SVM) :

```
238
239 # ##### Support Vector Machine (SVM) :
240
241 SVM = svm.SVC()
242 SVM.fit(x_train_df, train_set.Label)
243 predicted_values_SVM = SVM.predict(x_test_df)
244 print(predicted_values_SVM)
245 accuracy_SVM = accuracy_score(test_set.Label, predicted_values_SVM)
246 predictions['Support Vector Machine (SVM)'] = accuracy_SVM * 100
247 print("\nThe accuracy of Support Vector Machine (SVM) is {}".format(accuracy_SVM * 100))
248 confusion_matrix_SVM = confusion_matrix(test_set.Label, predicted_values_SVM)
249 print("\n", confusion_matrix_SVM)
250 skplt.metrics.plot_confusion_matrix(test_set.Label, predicted_values_SVM, normalize = True)
251 plt.show()
252
```

['ham' 'ham' 'ham' ... 'ham' 'ham' 'ham']

The accuracy of Support Vector Machine (SVM) is 86.2969004893964%

```
[[1587    0]
 [252     0]]
```

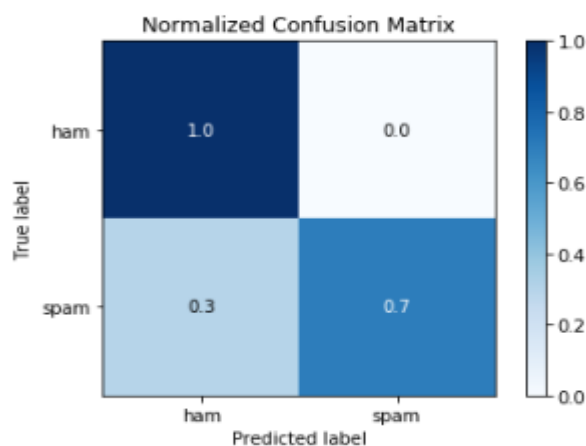


Random Forest:

```
254
255 # ##### Random Forest :
256
257 RF = RandomForestClassifier(n_estimators = 100, oob_score = True, random_state = 123456)
258 # n_estimators - количество деревьев в лесе
259 # oob_score - использовать ли образцы вне примеров для оценки точности обобщения
260 RF.fit(x_train_df, train_set.Label)
261 predicted_values_RF = RF.predict(x_test_df)
262 print(predicted_values_RF)
263 accuracy_RF = accuracy_score(test_set.Label, predicted_values_RF)
264 predictions['Random Forest'] = accuracy_RF * 100
265 print("\nThe accuracy of Random Forest is {}".format(accuracy_RF * 100))
266 confusion_matrix_RF = confusion_matrix(test_set.Label, predicted_values_RF)
267 print("\n", confusion_matrix_RF)
268 skplt.metrics.plot_confusion_matrix(test_set.Label, predicted_values_RF, normalize = True)
269 plt.show()
270
```

['ham' 'ham' 'ham' ... 'ham' 'ham' 'ham'] The accuracy of Random Forest is 95.86731919521479%

```
[[1587  0]
 [76 176]]
```



Logistic Rgression:

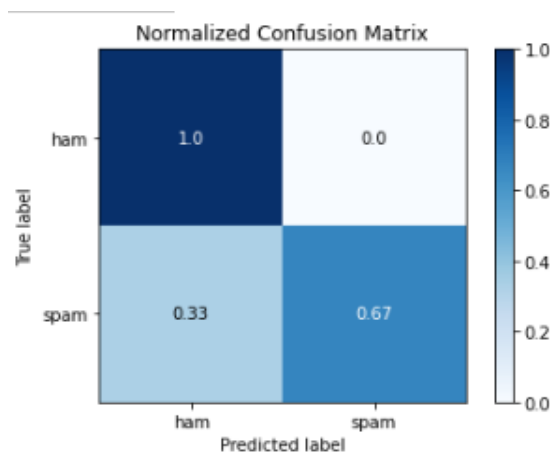
```
252
253
254 # ##### Logistic Regression :
255
256 from sklearn.linear_model import LogisticRegression
257 LR = DT = LogisticRegression()
258 LR.fit(x_train_df, train_set.Label)
259 predicted_values_LR = LR.predict(x_test_df)
260 print(predicted_values_LR)
261 accuracy_LR = accuracy_score(test_set.Label, predicted_values_LR)
262 predictions['Logistic Regression (LR)'] = accuracy_LR * 100
263 print("\nLogistic Regression (LR) is {}%".format(accuracy_LR * 100))
264
265 confusion_matrix_LR = confusion_matrix(test_set.Label, predicted_values_LR)
266 print("\n", confusion_matrix_LR)
267 skplt.metrics.plot_confusion_matrix(test_set.Label, predicted_values_LR, normalize = True)
268 plt.show()
269
270
```

['ham' 'ham' 'ham' ... 'ham' 'ham' 'ham']

Logistic Regression (LR) is 95.64980967917346%

```
[[1593    0]
```

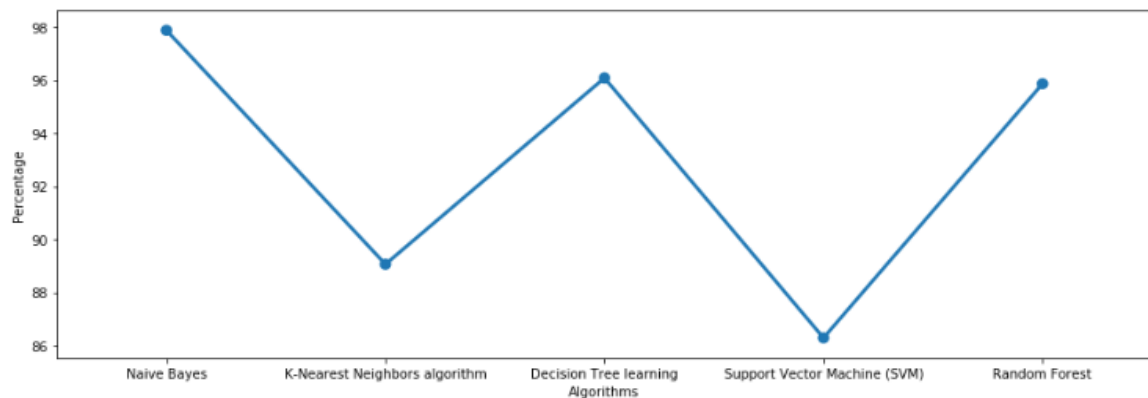
```
[80  166]]
```



Méthodes Comparison :

```
271
272 # ##### Method Comparison:
273
274 fig, (ax1) = plt.subplots(ncols = 1, sharey = True, figsize = (15,5))
275 df = pd.DataFrame(list(predictions.items()), columns = ['Algorithms', 'Percentage'])
276 display(df)
277 sns.pointplot(x = "Algorithms", y = "Percentage", data = df, ax = ax1);
278
279
```

	Algorithms	Percentage
0	Naive Bayes	97.879282
1	K-Nearest Neighbors algorithm	89.070147
2	Decision Tree learning	96.084829
3	Support Vector Machine (SVM)	86.296900
4	Random Forest	95.867319



ROC Accuracy :

```

282
283 #pr, tpr, thresholds = roc_curve(testset.v1,predicted_values_XGB, pos_label=2)
284 test_prediction = test_set.Label.tolist()
285 predicted_values = predicted_values_NB.tolist()
286 test_prediction = [1 if pred=="spam" else 0 for pred in test_prediction]
287 predicted_values = [1 if pred=="spam" else 0 for pred in predicted_values]
288 fpr, tpr, thresholds = roc_curve(test_prediction,predicted_values)
289 roc_auc = auc(fpr, tpr)
290 print("The ROC Accuracy is {}".format(roc_auc))
291

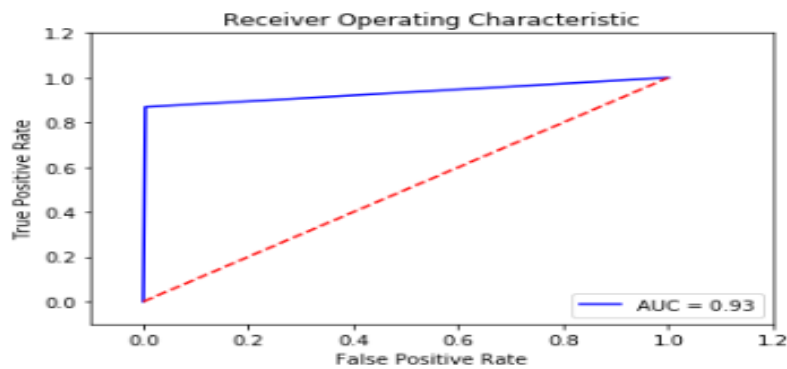
```

The ROC Accuracy is 0.9326334503555676

```

295
296 plt.title('Receiver Operating Characteristic')
297 plt.plot(fpr, tpr, 'b',
298 label='AUC = %0.2f'% roc_auc)
299 plt.legend(loc='lower right')
300 plt.plot([0,1],[0,1], 'r--')
301 plt.xlim([-0.1,1.2])
302 plt.ylim([-0.1,1.2])
303 plt.ylabel('True Positive Rate')
304 plt.xlabel('False Positive Rate')
305 plt.show()
306
307

```



Neural Network :

```

307
308
309 # ##### NN:
310 !gdown https://drive.google.com/uc?id=1XZcHqgnP_rvY7F6-B2ZuA_6w5ReFGV5H
311 df = pd.read_table('SMSSpamCollection',
312                   sep='\t',
313                   header=None,
314                   names=['Label', 'Message'])
315 df.head()
316 df["Label_tag"] = df.Label.map({'ham':0, 'spam':1})
317 df.head(5)
318

```

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

```

321
322 # get the size of our dataset
323 print(df.count())
324 df.Label_tag.value_counts()
325

```

```

Label      5572
Message    5572
Label_tag  5572
dtype: int64
0      4825
1       747
Name: Label_tag, dtype: int64

```

```

26
27 #Data Preparation
28 ##Training data
29 # first 4572/5572 emails
30 training_data = df[0:4572]
31 training_data_length = len(training_data.Label)
32 training_data.head()
33

```

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

```

334
335 #Testing data
336 test_data = df[-1000:]
337 test_data_length = len(test_data.Label)
338 test_data.head()
339

```

	Label	Message	Label_tag
4572	ham	CHA QUITEAMUZING THAT☐SCOOOL BABE,PROBPOP IN & ...	0
4573	ham	Omg how did u know what I ate?	0
4574	spam	URGENT! This is the 2nd attempt to contact U!U...	1
4575	ham	:(but your not here....	0
4576	ham	Not directly behind... Abt 4 rows behind ü...	0

```

339
340 #What is the shape of our input data
341 print(training_data.shape)
342 print(training_data.Label.shape)
343

```

(4572, 3)

(4572,)

```

343
344 #There are 3 features and 4572 samples in our trtraining set
345
346 #Test data
347 print(test_data.shape)
348 print(test_data.Label.shape)
349

```

(1000, 3)

(1000,)

```
350
351 #Develop a Predictive Theory
352 import random
353 print("labels \t : \t texts\n")
354 def pretty_print_Message_and_Label(i):
355     print(training_data.Label[i] + "\t:\t" + training_data.Message[i][:80] + "...")
356     # choose a random spam set to analyse
357     # random.randrange(start, stop, step)
358     pretty_print_Message_and_Label(random.randrange(0,4572))
359     pretty_print_Message_and_Label(random.randrange(0,4572,4))
360     pretty_print_Message_and_Label(random.randrange(0,4572,50))
361     pretty_print_Message_and_Label(random.randrange(0,4572,100))
362     pretty_print_Message_and_Label(random.randrange(0,4572,200))
363     pretty_print_Message_and_Label(random.randrange(0,4572,500))
364     pretty_print_Message_and_Label(random.randrange(0,4572,800))
365     pretty_print_Message_and_Label(random.randrange(0,4572,1000))
366
```

```
367
368 from collections import Counter
369 import numpy as np
370 import pprint
371 spam_counts = Counter()
372 ham_counts = Counter()
373 total_counts = Counter()
374 spam_ham_ratios = Counter()
375
376 pp = pprint.PrettyPrinter(indent=4)
377
378 for i in range(training_data_length):
379     if(training_data.Label[i] == 0):
380         for word in training_data.Message[i].split(" "):
381             ham_counts[word] += 1
382             total_counts[word] += 1
383     else:
384         for word in training_data.Message[i].split(" "):
385             spam_counts[word] += 1
386             total_counts[word] += 1
387
388 pp.pprint(spam_counts.most_common()[0:30])
389
```

```
391
392 for word,count in list(total_counts.most_common()):
393     if(count > 100):
394         spam_ham_ratio = spam_counts[word] / float(ham_counts[word]+1)
395         spam_ham_ratios[word] = spam_ham_ratio
396
397 for word,ratio in spam_ham_ratios.most_common():
398     if(ratio > 1):
399         spam_ham_ratios[word] = np.log(ratio)
400     else:
401         spam_ham_ratios[word] = -np.log((1 / (ratio+0.01)))
402
403
404 # words most frequently seen in a text with a "spam" label
405 pp.pprint(spam_ham_ratios.most_common()[0:30])
406
```



```

408
409 def update_input_layer(text):
410     pp.pprint(text)
411     global vocab_vector
412
413     # clear out previous state, reset the vector to be all 0s
414     vocab_vector *= 0
415     for word in text.split(" "):
416         vocab_vector[0][vocab_column_dict[word]] += 1
417
418     update_input_layer(training_data["text"][random.randrange(0,4572,4)])
419
420

```

```

423 import time
424 import sys
425 # Let's tweak our network from before to model these phenomena
426 class SpamClassificationNeuralNetwork(object):
427     def __init__(self, training_data, num_hidden_nodes = 10, num_epochs = 10, learning_rate = 0.1):
428         # set our random number generator
429         np.random.seed(1)
430         # pre-process data
431         self.pre_process_data(training_data)
432
433         self.num_features = len(self.vocab)
434         self.vocab_vector = np.zeros((1, len(self.vocab)))
435         self.num_input_nodes = self.num_features
436         self.num_hidden_nodes = num_hidden_nodes
437         self.num_epochs = num_epochs
438         self.num_output_nodes = 1
439         self.learning_rate = learning_rate
440
441         # Initialize weights
442         self.weights_i_h = np.random.randn(self.num_input_nodes, self.num_hidden_nodes)
443         self.weights_h_o = np.random.randn(self.num_hidden_nodes, self.num_output_nodes)
444

```

```

444
445 def forward_backward_propagate(self, Message, Label):
446     ### Forward pass ###
447     # Input Layer
448     self.update_input_layer(Message)
449     # Hidden layer
450     hidden_layer = self.vocab_vector.dot(self.weights_i_h)
451     # Output layer
452     output_layer = self.sigmoid(hidden_layer.dot(self.weights_h_o))
453
454     ### Backward pass ###
455     # Output error
456     output_layer_error = output_layer - Label
457     output_layer_delta = output_layer_error * self.sigmoid_derivative(output_layer)
458     # Backpropagated error - to the hidden layer
459     hidden_layer_error = output_layer_delta.dot(self.weights_h_o.T)
460     # hidden layer gradients - no nonlinearity so it's the same as the error
461     hidden_layer_delta = output_layer_error
462
463     # update the weights - with gradient descent
464     self.weights_h_o -= hidden_layer.T.dot(output_layer_delta) * self.learning_rate
465     self.weights_i_h -= self.vocab_vector.T.dot(hidden_layer_delta) * self.learning_rate
466
467     if(np.abs(output_layer_error) < 0.5):
468         self.correct_so_far += 1
469
470

```

```

471
472 def sigmoid(self,x):
473     return 1 / (1 + np.exp(-x))
474
475
476 def sigmoid_derivative(self,x):
477     return x * (1 - x)
478

```

```

479
480 def train(self):
481     for epoch in range(self.num_epochs):
482         self.correct_so_far = 0
483         start = time.time()
484         for i in range(len(training_data)):
485             # Forward and Back Propagation
486             self.forward_backward_propagate(training_data["Message"][i], training_data["Label_tag"][i])
487
488             samples_per_second = i / float(time.time() - start + 0.001)
489
490             sys.stdout.write("\rEpoch: " + str(epoch)
491                             + " Progress: " + str(100 * i/float(len(training_data)))[:4]
492                             + " % Speed(samples/sec): " + str(samples_per_second)[0:5]
493                             + " #Correct: " + str(self.correct_so_far)
494                             + " #Trained: " + str(i+1)
495                             + " Training Accuracy: " + str(self.correct_so_far * 100 / float(i+1))[:4] + "%")
496             print("")
497

```

```

497
498 def pre_process_data(self, training_data):
499     vocab = set()
500
501     for review in training_data["Messsage"]:
502         for word in review.split(" "):
503             vocab.add(word)
504
505     self.vocab = list(vocab)
506
507     self.word_to_column = {}
508     for i, word in enumerate(self.vocab):
509         self.word_to_column[word] = i
510

```

```

511
512 def update_input_layer(self, Message):
513     global vocab_vector
514     # clear out previous state, reset the vector to be all 0s
515     self.vocab_vector *= 0
516     for word in Message.split(" "):
517         self.vocab_vector[0][word_column_dict[word]] += 1
518
519 nn = SpamClassificationNeuralNetwork(training_data, num_epochs = 10, learning_rate=0.01)
520 nn.train()
521

```