

# Experiment 2: Email Spam or Ham Classification using Naïve Bayes, KNN, and SVM

Sri Sivasubramaniya Nadar College of Engineering, Chennai  
M. Tech (Integrated) Computer Science & Engineering - Semester V  
Subject Code & Name: ICS1512 – Machine Learning Algorithms Laboratory  
Academic Year: 2025-2026 (Odd Semester)  
Batch: 2023-2028

## Objective

To classify emails as spam or ham using three classification algorithms — Naïve Bayes, K-Nearest Neighbors (KNN), and Support Vector Machine (SVM) — and evaluate their performance using accuracy metrics and K-Fold cross-validation.

## Dataset

- Source: Spambase – Kaggle
- Description: This dataset includes extracted features from emails, labeled as spam or ham.

## Task Description

Develop models using Naïve Bayes, KNN, and SVM to classify email data. Evaluate their performance using a test split and K-Fold cross-validation, and interpret results with visualizations.

## Implementation Steps

1. Load and preprocess the dataset (handle missing values, normalization).
2. Perform Exploratory Data Analysis (EDA): class balance, feature distributions.
3. Split into training and testing sets.
4. Train models:
  - Naïve Bayes (Gaussian, Multinomial, Bernoulli)
  - K-Nearest Neighbors (vary k, KDTree, BallTree)
  - Support Vector Machine (Linear, Polynomial, RBF, Sigmoid kernels)

5. Evaluate using:
  - Accuracy, Precision, Recall, F1-Score
  - Confusion Matrix
  - ROC Curve
6. Perform K-Fold Cross Validation ( $K = 5$ ).
7. Compare results and record observations.

## Code Implementation

Listing 1: Spam/Ham Classification Code

```
1 import pandas as pd
2 import matplotlib.pyplot as plt
3 import seaborn as sns
4 import numpy as np
5 df=pd.read_csv(r'/content/drive/MyDrive/Ml_Experiment3/
   spambase_csv.csv')
6 data=df.copy()
7 df.columns
8 df.head()
9 df.select_dtypes(include='object').columns
10 df.select_dtypes(include='number').columns
11 df.isnull().sum()
12
13
14 numerical_cols = df.select_dtypes(include=np.number).columns
15
16
17 n_cols = 3
18 n_rows = (len(numerical_cols) + n_cols - 1) // n_cols
19
20
21 fig, axes = plt.subplots(n_rows, n_cols, figsize=(15, n_rows * 5)
   )
22 axes = axes.flatten() # Flatten the 2D array of axes for easy
   iteration
23
24 for i, col in enumerate(numerical_cols):
25     sns.boxplot(y=df[col], ax=axes[i])
26     axes[i].set_title(col)
27     axes[i].set_ylabel('')
28
29 for j in range(i + 1, len(axes)):
30     fig.delaxes(axes[j])
31
32 plt.tight_layout()
33 plt.show()
```

```

34 from sklearn.preprocessing import StandardScaler
35 scaler=StandardScaler()
36
37 d_columns=df.columns
38 d_columns=d_columns.drop('class')
39 print(d_columns)
40 df[d_columns]=scaler.fit_transform(df[d_columns])
41 len(d_columns)
42 df
43 from sklearn.model_selection import train_test_split
44 x=df.drop('class',axis=1)
45 y=df['class']
46 x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2,
47         random_state=42)
48
49 from sklearn.naive_bayes import GaussianNB, MultinomialNB,
50     BernoulliNB
51
52 from sklearn.metrics import accuracy_score, classification_report
53
54 # Gaussian Naive Bayes
55 gaussian_nb = GaussianNB()
56 gaussian_nb.fit(x_train, y_train)
57 y_pred_gaussian = gaussian_nb.predict(x_test)
58
59 print("Gaussian_Naive_Bayes:")
60 print("Accuracy:", accuracy_score(y_test, y_pred_gaussian))
61 print(classification_report(y_test, y_pred_gaussian))
62
63 bernoulli_nb = BernoulliNB()
64 bernoulli_nb.fit(x_train, y_train)
65 y_pred_bernoulli = bernoulli_nb.predict(x_test)
66
67 print("\nBernoulli_Naive_Bayes:")
68 print("Accuracy:", accuracy_score(y_test, y_pred_bernoulli))
69 print(classification_report(y_test, y_pred_bernoulli))
70 data
71
72 from sklearn.preprocessing import MinMaxScaler
73
74 # Create a MinMaxScaler
75 scaler_minmax = MinMaxScaler()
76
77 # Apply MinMaxScaler to the features (excluding the 'class'
78     column) of the 'data' variable
79 x_scaled_minmax = scaler_minmax.fit_transform(data.drop('class',
80     axis=1))
81
82 # Convert the scaled data back to a DataFrame
83 x_scaled_minmax_df = pd.DataFrame(x_scaled_minmax, columns=data.
84     drop('class', axis=1).columns)
85

```

```

80 # Display the first few rows of the scaled data
81 display(x_scaled_minmax_df.head())
82 from sklearn.model_selection import train_test_split
83
84 # Assuming 'class' is your target variable in the original 'data'
85   DataFrame
86
87 y = data['class']
88
89 # Split the scaled data into training and testing sets
90 x_train_minmax, x_test_minmax, y_train_minmax, y_test_minmax =
91   train_test_split(x_scaled_minmax_df, y, test_size=0.2,
92     random_state=42)
93
94 print("Shape of x_train_minmax:", x_train_minmax.shape)
95 print("Shape of x_test_minmax:", x_test_minmax.shape)
96 print("Shape of y_train_minmax:", y_train_minmax.shape)
97 print("Shape of y_test_minmax:", y_test_minmax.shape)
98 from sklearn.naive_bayes import MultinomialNB
99 from sklearn.metrics import accuracy_score, classification_report
100
101 # Multinomial Naive Bayes with MinMax scaled data
102 multinomial_nb_minmax = MultinomialNB()
103 multinomial_nb_minmax.fit(x_train_minmax, y_train_minmax)
104 y_pred_multinomial_minmax = multinomial_nb_minmax.predict(
105   x_test_minmax)
106
107 print("Multinomial Naive Bayes with MinMax Scaled Data:")
108 print("Accuracy:", accuracy_score(y_test_minmax,
109   y_pred_multinomial_minmax))
110 print(classification_report(y_test_minmax,
111   y_pred_multinomial_minmax))
112
113 from sklearn.metrics import confusion_matrix
114
115 # Calculate the confusion matrix
116 cm = confusion_matrix(y_test_minmax, y_pred_multinomial_minmax)
117
118 # Display the confusion matrix using seaborn heatmap
119 plt.figure(figsize=(8, 6))
120 sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=['
121   Not Spam', 'Spam'], yticklabels=['Not Spam', 'Spam'])
122 plt.xlabel('Predicted')
123 plt.ylabel('Actual')
124 plt.title('Confusion Matrix for Multinomial')
125 plt.show()
126 cm = confusion_matrix(y_test, y_pred_gaussian)
127
128 # Display the confusion matrix using seaborn heatmap
129 plt.figure(figsize=(8, 6))

```

```

123 sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=['
    Not_Spam', 'Spam'], yticklabels=['Not_Spam', 'Spam'])
124 plt.xlabel('Predicted')
125 plt.ylabel('Actual')
126 plt.title('Confusion_Matrix_for_Gaussian')
127 plt.show()
128 cm = confusion_matrix(y_test, y_pred_bernoulli)
129
130 # Display the confusion matrix using seaborn heatmap
131 plt.figure(figsize=(8, 6))
132 sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=['
    Not_Spam', 'Spam'], yticklabels=['Not_Spam', 'Spam'])
133 plt.xlabel('Predicted')
134 plt.ylabel('Actual')
135 plt.title('Confusion_Matrix_for_bernoulli')
136 plt.show()
137 from sklearn.metrics import roc_curve, auc
138
139 # Gaussian Naive Bayes
140 y_pred_prob_gaussian = gaussian_nb.predict_proba(x_test)[: , 1]
141 fpr_gaussian, tpr_gaussian, thresholds_gaussian = roc_curve(
    y_test, y_pred_prob_gaussian)
142 auc_gaussian = auc(fpr_gaussian, tpr_gaussian)
143
144 # Multinomial Naive Bayes with MinMax scaled data
145 y_pred_prob_multinomial_minmax = multinomial_nb_minmax.
    predict_proba(x_test_minmax)[: , 1]
146 fpr_multinomial, tpr_multinomial, thresholds_multinomial =
    roc_curve(y_test_minmax, y_pred_prob_multinomial_minmax)
147 auc_multinomial = auc(fpr_multinomial, tpr_multinomial)
148
149 # Bernoulli Naive Bayes
150 y_pred_prob_bernoulli = bernoulli_nb.predict_proba(x_test)[: , 1]
151 fpr_bernoulli, tpr_bernoulli, thresholds_bernoulli = roc_curve(
    y_test, y_pred_prob_bernoulli)
152 auc_bernoulli = auc(fpr_bernoulli, tpr_bernoulli)
153
154 print("Gaussian_Naive_Bayes_AUC:", auc_gaussian)
155 print("Multinomial_Naive_Bayes_AUC_(MinMax_Scaled):",
    auc_multinomial)
156 print("Bernoulli_Naive_Bayes_AUC:", auc_bernoulli)
157 plt.figure(figsize=(10, 8))
158 plt.plot(fpr_gaussian, tpr_gaussian, label='Gaussian_NB_(AUC=
    %0.2f)' % auc_gaussian)
159 plt.plot(fpr_multinomial, tpr_multinomial, label='Multinomial_NB_
    (AUC=%0.2f)' % auc_multinomial)
160 plt.plot(fpr_bernoulli, tpr_bernoulli, label='Bernoulli_NB_(AUC=
    %0.2f)' % auc_bernoulli)
161 plt.plot([0, 1], [0, 1], 'k--') # Diagonal random classifier
    line
162 plt.xlabel('False_Positive_Rate')

```

```

163 plt.ylabel('True_Positive_Rate')
164 plt.title('ROC_Curve_for_Naive_Bayes_Models')
165 plt.legend(loc='lower_right')
166 plt.grid(True)
167 plt.show()
168
169 from sklearn.model_selection import cross_val_score
170
171 cv_scores = cross_val_score(bernoulli_nb, x, y, cv=5)
172
173 i=1
174 for score in cv_scores:
175     print(f"Fold_{i}_accuracy:_{score}")
176     i+=1
177
178 print("Mean_cross-validation_accuracy:", cv_scores.mean())
179 <h1>KNN CLASSIFICATION</h1>
180 from sklearn.neighbors import KNeighborsClassifier
181
182 k_values = [1, 3, 5, 7]
183
184 for k in k_values:
185     print(f"\nK-Nearest_Neighbors_(k={k}):")
186     knn=KNeighborsClassifier(n_neighbors=k)
187     knn.fit(x_train,y_train)
188     y_pred_knn=knn.predict(x_test)
189     print("Accuracy:", accuracy_score(y_test, y_pred_knn))
190     print("Classification_Report:")
191     print(classification_report(y_test, y_pred_knn))
192     cn=confusion_matrix(y_test,y_pred_knn)
193     sns.heatmap(cn,annot=True,fmt='d',cmap='Blues',xticklabels=['
        Not_Spam', 'Spam'],yticklabels=['Not_Spam', 'Spam'])
194     plt.xlabel('Predicted')
195     plt.ylabel('Actual')
196     plt.title(f'Confusion_Matrix_for_k={k}')
197     plt.show()
198     y_pred_probab=knn.predict_proba(x_test)[: ,1]
199     fpr,tpr,thresholds=roc_curve(y_test,y_pred_probab)
200     auc_score=auc(fpr,tpr)
201     print(f"AUC_Score_for_k={k}:_{auc_score}")
202     plt.figure(figsize=(8,6))
203     plt.plot(fpr,tpr,label=f'KNN_(AUC={auc_score:.2f})')
204     plt.show()
205     knn_kd = KNeighborsClassifier(n_neighbors=5, algorithm='kd_tree')
206     knn_kd.fit(x_train, y_train)
207     y_pred_kd = knn_kd.predict(x_test)
208     accuracy_kd = accuracy_score(y_test, y_pred_kd)
209     print(f"Accuracy_with_KD-Tree:_{accuracy_kd:.4f}")
210     print(classification_report(y_test, y_pred_kd))
211     # Using Ball Tree

```

```

212 knn_ball = KNeighborsClassifier(n_neighbors=5, algorithm='
      ball_tree')
213 knn_ball.fit(x_train, y_train)
214 y_pred_ball = knn_ball.predict(x_test)
215 accuracy_ball = accuracy_score(y_test, y_pred_ball)
216 print(f"Accuracy with Ball Tree: {accuracy_ball:.4f}")
217 print(classification_report(y_test, y_pred_ball))
218
219 knn=KNeighborsClassifier(n_neighbors=5)
220 cv_scores = cross_val_score(knn, x, y, cv=5)
221
222 i=1
223 for score in cv_scores:
224     print(f"Fold {i} accuracy: {score}")
225     i+=1
226
227 print("Mean cross-validation accuracy:", cv_scores.mean())
228 <h1>SVM</h1>
229
230 from sklearn.svm import SVC
231 lsvc=SVC(kernel='linear',C=1.0,random_state=42)
232 lsvc.fit(x_train,y_train)
233 y_pred_lsvc=lsvc.predict(x_test)
234 print("Accuracy:", accuracy_score(y_test, y_pred_lsvc))
235 print("Classification Report:")
236 print(classification_report(y_test, y_pred_lsvc))
237 psvc=SVC(kernel='poly',C=1.0,random_state=42,gamma=0.1,degree=2)
238 psvc.fit(x_train,y_train)
239 y_pred_psvc=psvc.predict(x_test)
240 print("Accuracy:", accuracy_score(y_test, y_pred_psvc))
241 print("Classification Report:")
242 print(classification_report(y_test, y_pred_psvc))
243 p3svc=SVC(kernel='poly',C=1.0,random_state=42,gamma=0.1,degree=3)
244 p3svc.fit(x_train,y_train)
245 y_pred_p3svc=p3svc.predict(x_test)
246 print("Accuracy:", accuracy_score(y_test, y_pred_p3svc))
247 print("Classification Report:")
248 print(classification_report(y_test, y_pred_p3svc))
249 rbfsvc=SVC(kernel='rbf',C=1.0,random_state=42,gamma=0.01)
250 rbfsvc.fit(x_train,y_train)
251 y_pred_rbfsvc=rbfsvc.predict(x_test)
252 print("Accuracy:", accuracy_score(y_test, y_pred_rbfsvc))
253 print("Classification Report:")
254 print(classification_report(y_test, y_pred_rbfsvc))
255 sigmoidsvc=SVC(kernel='sigmoid',C=1.0,random_state=42,gamma=0.01)
256 sigmoidsvc.fit(x_train,y_train)
257 y_pred_sigmoidsvc=sigmoidsvc.predict(x_test)
258 print("Accuracy:", accuracy_score(y_test, y_pred_sigmoidsvc))
259 print("Classification Report:")
260 print(classification_report(y_test, y_pred_sigmoidsvc))
261 cv_score=cross_val_score(lsvc,x,y,cv=5)

```

```

262 print(cv_score)
263 print("Mean_cross-validation_accuracy:", cv_score.mean())
264 cv_score=cross_val_score(psvc,x,y,cv=5)
265 print(cv_score)
266 print("Mean_cross-validation_accuracy:", cv_score.mean())
267 cv_score=cross_val_score(p3svc,x,y,cv=5)
268 print(cv_score)
269 print("Mean_cross-validation_accuracy:", cv_score.mean())
270 cv_score=cross_val_score(rbfsvc,x,y,cv=5)
271 print(cv_score)
272 print("Mean_cross-validation_accuracy:", cv_score.mean())
273 cv_score=cross_val_score(sigmoidsvc,x,y,cv=5)
274 print(cv_score)
275 print("Mean_cross-validation_accuracy:", cv_score.mean()) import
    pandas as pd
276 import matplotlib.pyplot as plt
277 import seaborn as sns
278 import numpy as np
279 df=pd.read_csv(r'/content/drive/MyDrive/Ml_Experiment3/
    spambase_csv.csv')
280 data=df.copy()
281 df.columns
282 df.head()
283 df.select_dtypes(include='object').columns
284 df.select_dtypes(include='number').columns
285 df.isnull().sum()
286
287
288 numerical_cols = df.select_dtypes(include=np.number).columns
289
290
291 n_cols = 3
292 n_rows = (len(numerical_cols) + n_cols - 1) // n_cols
293
294
295 fig, axes = plt.subplots(n_rows, n_cols, figsize=(15, n_rows * 5)
    )
296 axes = axes.flatten() # Flatten the 2D array of axes for easy
    iteration
297
298 for i, col in enumerate(numerical_cols):
299     sns.boxplot(y=df[col], ax=axes[i])
300     axes[i].set_title(col)
301     axes[i].set_ylabel('')
302
303 for j in range(i + 1, len(axes)):
304     fig.delaxes(axes[j])
305
306 plt.tight_layout()
307 plt.show()
308 from sklearn.preprocessing import StandardScaler

```



```

309 scaler=StandardScaler()
310
311 d_columns=df.columns
312 d_columns=d_columns.drop('class')
313 print(d_columns)
314 df[d_columns]=scaler.fit_transform(df[d_columns])
315 len(d_columns)
316 df
317 from sklearn.model_selection import train_test_split
318 x=df.drop('class',axis=1)
319 y=df['class']
320 x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2,
321         random_state=42)
322
323 from sklearn.naive_bayes import GaussianNB, MultinomialNB,
324     BernoulliNB
325
326 # Gaussian Naive Bayes
327 gaussian_nb = GaussianNB()
328 gaussian_nb.fit(x_train, y_train)
329 y_pred_gaussian = gaussian_nb.predict(x_test)
330
331 print("Gaussian_Naive_Bayes:")
332 print("Accuracy:", accuracy_score(y_test, y_pred_gaussian))
333 print(classification_report(y_test, y_pred_gaussian))
334 bernoulli_nb = BernoulliNB()
335 bernoulli_nb.fit(x_train, y_train)
336 y_pred_bernoulli = bernoulli_nb.predict(x_test)
337
338 print("\nBernoulli_Naive_Bayes:")
339 print("Accuracy:", accuracy_score(y_test, y_pred_bernoulli))
340 print(classification_report(y_test, y_pred_bernoulli))
341 data
342
343 from sklearn.preprocessing import MinMaxScaler
344
345 # Create a MinMaxScaler
346 scaler_minmax = MinMaxScaler()
347
348 # Apply MinMaxScaler to the features (excluding the 'class'
349     column) of the 'data' variable
350 x_scaled_minmax = scaler_minmax.fit_transform(data.drop('class',
351     axis=1))
352
353 # Convert the scaled data back to a DataFrame
354 x_scaled_minmax_df = pd.DataFrame(x_scaled_minmax, columns=data.
355     drop('class', axis=1).columns)
356
357 # Display the first few rows of the scaled data

```

```

355 display(x_scaled_minmax_df.head())
356 from sklearn.model_selection import train_test_split
357
358 # Assuming 'class' is your target variable in the original 'data'
    DataFrame
359 y = data['class']
360
361 # Split the scaled data into training and testing sets
362 x_train_minmax, x_test_minmax, y_train_minmax, y_test_minmax =
    train_test_split(x_scaled_minmax_df, y, test_size=0.2,
        random_state=42)
363
364 print("Shape of x_train_minmax:", x_train_minmax.shape)
365 print("Shape of x_test_minmax:", x_test_minmax.shape)
366 print("Shape of y_train_minmax:", y_train_minmax.shape)
367 print("Shape of y_test_minmax:", y_test_minmax.shape)
368 from sklearn.naive_bayes import MultinomialNB
369 from sklearn.metrics import accuracy_score, classification_report
370
371 # Multinomial Naive Bayes with MinMax scaled data
372 multinomial_nb_minmax = MultinomialNB()
373 multinomial_nb_minmax.fit(x_train_minmax, y_train_minmax)
374 y_pred_multinomial_minmax = multinomial_nb_minmax.predict(
    x_test_minmax)
375
376 print("Multinomial Naive Bayes with MinMax Scaled Data:")
377 print("Accuracy:", accuracy_score(y_test_minmax,
    y_pred_multinomial_minmax))
378 print(classification_report(y_test_minmax,
    y_pred_multinomial_minmax))
379
380 from sklearn.metrics import confusion_matrix
381
382 # Calculate the confusion matrix
383 cm = confusion_matrix(y_test_minmax, y_pred_multinomial_minmax)
384
385 # Display the confusion matrix using seaborn heatmap
386 plt.figure(figsize=(8, 6))
387 sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=['
    Not Spam', 'Spam'], yticklabels=['Not Spam', 'Spam'])
388 plt.xlabel('Predicted')
389 plt.ylabel('Actual')
390 plt.title('Confusion Matrix for Multinomial')
391 plt.show()
392 cm = confusion_matrix(y_test, y_pred_gaussian)
393
394 # Display the confusion matrix using seaborn heatmap
395 plt.figure(figsize=(8, 6))
396 sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=['
    Not Spam', 'Spam'], yticklabels=['Not Spam', 'Spam'])

```

```

398 plt.xlabel('Predicted')
399 plt.ylabel('Actual')
400 plt.title('Confusion_Matrix_for_Gaussian')
401 plt.show()
402 cm = confusion_matrix(y_test, y_pred_bernoulli)
403
404 # Display the confusion matrix using seaborn heatmap
405 plt.figure(figsize=(8, 6))
406 sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=['
    Not_Spam', 'Spam'], yticklabels=['Not_Spam', 'Spam'])
407 plt.xlabel('Predicted')
408 plt.ylabel('Actual')
409 plt.title('Confusion_Matrix_for_bernoulli')
410 plt.show()
411 from sklearn.metrics import roc_curve, auc
412
413 # Gaussian Naive Bayes
414 y_pred_prob_gaussian = gaussian_nb.predict_proba(x_test)[: , 1]
415 fpr_gaussian, tpr_gaussian, thresholds_gaussian = roc_curve(
    y_test, y_pred_prob_gaussian)
416 auc_gaussian = auc(fpr_gaussian, tpr_gaussian)
417
418 # Multinomial Naive Bayes with MinMax scaled data
419 y_pred_prob_multinomial_minmax = multinomial_nb_minmax.
    predict_proba(x_test_minmax)[: , 1]
420 fpr_multinomial, tpr_multinomial, thresholds_multinomial =
    roc_curve(y_test_minmax, y_pred_prob_multinomial_minmax)
421 auc_multinomial = auc(fpr_multinomial, tpr_multinomial)
422
423 # Bernoulli Naive Bayes
424 y_pred_prob_bernoulli = bernoulli_nb.predict_proba(x_test)[: , 1]
425 fpr_bernoulli, tpr_bernoulli, thresholds_bernoulli = roc_curve(
    y_test, y_pred_prob_bernoulli)
426 auc_bernoulli = auc(fpr_bernoulli, tpr_bernoulli)
427
428 print("Gaussian_Naive_Bayes_AUC:", auc_gaussian)
429 print("Multinomial_Naive_Bayes_AUC_(MinMax_Scaled):",
    auc_multinomial)
430 print("Bernoulli_Naive_Bayes_AUC:", auc_bernoulli)
431 plt.figure(figsize=(10, 8))
432 plt.plot(fpr_gaussian, tpr_gaussian, label='Gaussian_NB_(AUC=
    %0.2f)' % auc_gaussian)
433 plt.plot(fpr_multinomial, tpr_multinomial, label='Multinomial_NB_
    (AUC=%0.2f)' % auc_multinomial)
434 plt.plot(fpr_bernoulli, tpr_bernoulli, label='Bernoulli_NB_(AUC=
    %0.2f)' % auc_bernoulli)
435 plt.plot([0, 1], [0, 1], 'k--') # Diagonal random classifier
    line
436 plt.xlabel('False_Positive_Rate')
437 plt.ylabel('True_Positive_Rate')
438 plt.title('ROC_Curve_for_Naive_Bayes_Models')

```

```

439 plt.legend(loc='lower_right')
440 plt.grid(True)
441 plt.show()
442
443 from sklearn.model_selection import cross_val_score
444
445 cv_scores = cross_val_score(bernoulli_nb, x, y, cv=5)
446
447 i=1
448 for score in cv_scores:
449     print(f"Fold_{i}_accuracy:_{score}")
450     i+=1
451
452 print("Mean_cross-validation_accuracy:", cv_scores.mean())
453 <h1>KNN CLASSIFICATION</h1>
454 from sklearn.neighbors import KNeighborsClassifier
455
456 k_values = [1, 3, 5, 7]
457
458 for k in k_values:
459     print(f"\nK-Nearest_Neighbors_(k={k}):")
460     knn=KNeighborsClassifier(n_neighbors=k)
461     knn.fit(x_train,y_train)
462     y_pred_knn=knn.predict(x_test)
463     print("Accuracy:", accuracy_score(y_test, y_pred_knn))
464     print("Classification_Report:")
465     print(classification_report(y_test, y_pred_knn))
466     cn=confusion_matrix(y_test,y_pred_knn)
467     sns.heatmap(cn,annot=True,fmt='d',cmap='Blues',xticklabels=['
         Not_Spam', 'Spam'],yticklabels=['Not_Spam', 'Spam'])
468     plt.xlabel('Predicted')
469     plt.ylabel('Actual')
470     plt.title(f'Confusion_Matrix_for_k={k}')
471     plt.show()
472     y_pred_probab=knn.predict_proba(x_test)[:,-1]
473     fpr,tpr,thresholds=roc_curve(y_test,y_pred_probab)
474     auc_score=auc(fpr,tpr)
475     print(f"AUC_Score_for_k={k}:_{auc_score}")
476     plt.figure(figsize=(8,6))
477     plt.plot(fpr,tpr,label=f'KNN_(AUC={auc_score:.2f})')
478     plt.show()
479 knn_kd = KNeighborsClassifier(n_neighbors=5, algorithm='kd_tree')
480 knn_kd.fit(x_train, y_train)
481 y_pred_kd = knn_kd.predict(x_test)
482 accuracy_kd = accuracy_score(y_test, y_pred_kd)
483 print(f"Accuracy_with_KD-Tree:_{accuracy_kd:.4f}")
484 print(classification_report(y_test, y_pred_kd))
485 # Using Ball Tree
486 knn_ball = KNeighborsClassifier(n_neighbors=5, algorithm='
         ball_tree')
487 knn_ball.fit(x_train, y_train)

```

```

488 y_pred_ball = knn_ball.predict(x_test)
489 accuracy_ball = accuracy_score(y_test, y_pred_ball)
490 print(f"Accuracy with Ball Tree: {accuracy_ball:.4f}")
491 print(classification_report(y_test, y_pred_ball))
492
493 knn=KNeighborsClassifier(n_neighbors=5)
494 cv_scores = cross_val_score(knn, x, y, cv=5)
495
496 i=1
497 for score in cv_scores:
498     print(f"Fold {i} accuracy: {score}")
499     i+=1
500
501 print("Mean cross-validation accuracy:", cv_scores.mean())
502 <h1>SVM</h1>
503
504 from sklearn.svm import SVC
505 lsvc=SVC(kernel='linear',C=1.0,random_state=42)
506 lsvc.fit(x_train,y_train)
507 y_pred_lsvc=lsvc.predict(x_test)
508 print("Accuracy:", accuracy_score(y_test, y_pred_lsvc))
509 print("Classification Report:")
510 print(classification_report(y_test, y_pred_lsvc))
511 psvc=SVC(kernel='poly',C=1.0,random_state=42,gamma=0.1,degree=2)
512 psvc.fit(x_train,y_train)
513 y_pred_psvc=psvc.predict(x_test)
514 print("Accuracy:", accuracy_score(y_test, y_pred_psvc))
515 print("Classification Report:")
516 print(classification_report(y_test, y_pred_psvc))
517 p3svc=SVC(kernel='poly',C=1.0,random_state=42,gamma=0.1,degree=3)
518 p3svc.fit(x_train,y_train)
519 y_pred_p3svc=p3svc.predict(x_test)
520 print("Accuracy:", accuracy_score(y_test, y_pred_p3svc))
521 print("Classification Report:")
522 print(classification_report(y_test, y_pred_p3svc))
523 rbfsvc=SVC(kernel='rbf',C=1.0,random_state=42,gamma=0.01)
524 rbfsvc.fit(x_train,y_train)
525 y_pred_rbfsvc=rbfsvc.predict(x_test)
526 print("Accuracy:", accuracy_score(y_test, y_pred_rbfsvc))
527 print("Classification Report:")
528 print(classification_report(y_test, y_pred_rbfsvc))
529 sigmoidsvc=SVC(kernel='sigmoid',C=1.0,random_state=42,gamma=0.01)
530 sigmoidsvc.fit(x_train,y_train)
531 y_pred_sigmoidsvc=sigmoidsvc.predict(x_test)
532 print("Accuracy:", accuracy_score(y_test, y_pred_sigmoidsvc))
533 print("Classification Report:")
534 print(classification_report(y_test, y_pred_sigmoidsvc))
535 cv_score=cross_val_score(lsvc,x,y,cv=5)
536 print(cv_score)
537 print("Mean cross-validation accuracy:", cv_score.mean())
538 cv_score=cross_val_score(psvc,x,y,cv=5)

```

```

539 print(cv_score)
540 print("Mean cross-validation accuracy:", cv_score.mean())
541 cv_score=cross_val_score(p3svc,x,y,cv=5)
542 print(cv_score)
543 print("Mean cross-validation accuracy:", cv_score.mean())
544 cv_score=cross_val_score(rbfsvc,x,y,cv=5)
545 print(cv_score)
546 print("Mean cross-validation accuracy:", cv_score.mean())
547 cv_score=cross_val_score(sigmoidsvc,x,y,cv=5)
548 print(cv_score)
549 print("Mean cross-validation accuracy:", cv_score.mean())

```

## Performance Comparison Tables

**Table 1: Performance Comparison of Naïve Bayes Variants**

Metric	Gaussian NB	Multinomial NB	Bernoulli NB
Accuracy	0.822	0.872	0.900
Precision	0.830	0.890	0.910
Recall	0.840	0.850	0.890
F1 Score	0.820	0.860	0.900

**Table 2: KNN Performance for Different k Values**

k	Accuracy	Precision	Recall	F1 Score
1	0.896	0.890	0.890	0.890
3	0.894	0.895	0.890	0.890
5	0.896	0.900	0.890	0.890
7	0.896	0.900	0.890	0.890

**Table 3: KNN Comparison - KDTree vs BallTree**

Metric	KDTree	BallTree
Accuracy	0.896	0.896
Precision	0.900	0.900
Recall	0.890	0.890
F1 Score	0.890	0.890

**Table 4: SVM Performance with Different Kernels and Parameters**

Kernel	Hyperparameters	Accuracy	F1 Score	Training Time (s)
--------	-----------------	----------	----------	-------------------

Linear	$C = 1.0$	0.925	0.920	0.0265
Polynomial	$C = 1.0$ , degree = 3, $\gamma = 0.01$	0.931	0.930	0.711
RBF	$C = 1.0$ , $\gamma = 0.01$	0.935	0.930	0.3874
Sigmoid	$C = 1.0$ , $\gamma = 0.01$	0.904	0.900	0.2830

**Table 5: K-Fold Cross-Validation Results (K = 5)**

Fold	Naïve Bayes Accuracy	KNN Accuracy	SVM Accuracy
Fold 1	0.913	0.889	0.899
Fold 2	0.912	0.902	0.915
Fold 3	0.915	0.921	0.911
Fold 4	0.930	0.918	0.911
Fold 5	0.818	0.788	0.817
Average	0.898	0.884	0.891

## References

- scikit-learn: Naïve Bayes
- scikit-learn: KNN
- scikit-learn: SVM
- Spambase Dataset – Kaggle