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

```python
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
df=pd.read_csv(r'/content/drive/MyDrive/Ml_Experiment3/spambase_csv.csv')
data=df.copy()
df.columns
df.head()
df.select_dtypes(include='object').columns
df.select_dtypes(include='number').columns
df.isnull().sum()


numerical_cols = df.select_dtypes(include=np.number).columns


n_cols = 3
n_rows = (len(numerical_cols) + n_cols - 1) // n_cols


fig, axes = plt.subplots(n_rows, n_cols, figsize=(15, n_rows * 5))
axes = axes.flatten() # Flatten the 2D array of axes for easy iteration

for i, col in enumerate(numerical_cols):
    sns.boxplot(y=df[col], ax=axes[i])
    axes[i].set_title(col)
    axes[i].set_ylabel('')

for j in range(i + 1, len(axes)):
    fig.delaxes(axes[j])

plt.tight_layout()
plt.show()
```

```python
from sklearn.preprocessing import StandardScaler
scaler=StandardScaler()

d_columns=df.columns
d_columns=d_columns.drop('class')
print(d_columns)
df[d_columns]=scaler.fit_transform(df[d_columns])
len(d_columns)
df
from sklearn.model_selection import train_test_split
x=df.drop('class',axis=1)
y=df['class']
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2,random_state=42)


from sklearn.naive_bayes import GaussianNB, MultinomialNB, BernoulliNB
from sklearn.metrics import accuracy_score, classification_report

# Gaussian Naive Bayes
gaussian_nb = GaussianNB()
gaussian_nb.fit(x_train, y_train)
y_pred_gaussian = gaussian_nb.predict(x_test)

print("Gaussian Naive Bayes:")
print("Accuracy:", accuracy_score(y_test, y_pred_gaussian))
print(classification_report(y_test, y_pred_gaussian))
bernoulli_nb = BernoulliNB()
bernoulli_nb.fit(x_train, y_train)
y_pred_bernoulli = bernoulli_nb.predict(x_test)

print("\nBernoulli Naive Bayes:")
print("Accuracy:", accuracy_score(y_test, y_pred_bernoulli))
print(classification_report(y_test, y_pred_bernoulli))
data

from sklearn.preprocessing import MinMaxScaler

# Create a MinMaxScaler
scaler_minmax = MinMaxScaler()

# Apply MinMaxScaler to the features (excluding the 'class' column) of the 'data' variable
x_scaled_minmax = scaler_minmax.fit_transform(data.drop('class', axis=1))

# Convert the scaled data back to a DataFrame
x_scaled_minmax_df = pd.DataFrame(x_scaled_minmax, columns=data.drop('class', axis=1).columns)

```

```python
# Display the first few rows of the scaled data
display(x_scaled_minmax_df.head())
from sklearn.model_selection import train_test_split

# Assuming 'class' is your target variable in the original 'data'
    DataFrame
y = data['class']

# Split the scaled data into training and testing sets
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)

print("Shape of x_train_minmax:", x_train_minmax.shape)
print("Shape of x_test_minmax:", x_test_minmax.shape)
print("Shape of y_train_minmax:", y_train_minmax.shape)
print("Shape of y_test_minmax:", y_test_minmax.shape)
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score, classification_report

# Multinomial Naive Bayes with MinMax scaled data
multinomial_nb_minmax = MultinomialNB()
multinomial_nb_minmax.fit(x_train_minmax, y_train_minmax)
y_pred_multinomial_minmax = multinomial_nb_minmax.predict(
    x_test_minmax)

print("Multinomial Naive Bayes with MinMax Scaled Data:")
print("Accuracy:", accuracy_score(y_test_minmax,
    y_pred_multinomial_minmax))
print(classification_report(y_test_minmax,
    y_pred_multinomial_minmax))

from sklearn.metrics import confusion_matrix


# Calculate the confusion matrix
cm = confusion_matrix(y_test_minmax, y_pred_multinomial_minmax)

# Display the confusion matrix using seaborn heatmap
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=['
    Not Spam', 'Spam'], yticklabels=['Not Spam', 'Spam'])
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix for Multinomial')
plt.show()
cm = confusion_matrix(y_test, y_pred_gaussian)

# Display the confusion matrix using seaborn heatmap
plt.figure(figsize=(8, 6))
```

```
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=['Not Spam', 'Spam'], yticklabels=['Not Spam', 'Spam'])
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix for Gaussian')
plt.show()
cm = confusion_matrix(y_test, y_pred_bernoulli)

# Display the confusion matrix using seaborn heatmap
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=['Not Spam', 'Spam'], yticklabels=['Not Spam', 'Spam'])
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix for bernoulli')
plt.show()
from sklearn.metrics import roc_curve, auc

# Gaussian Naive Bayes
y_pred_prob_gaussian = gaussian_nb.predict_proba(x_test)[:, 1]
fpr_gaussian, tpr_gaussian, thresholds_gaussian = roc_curve(y_test, y_pred_prob_gaussian)
auc_gaussian = auc(fpr_gaussian, tpr_gaussian)

# Multinomial Naive Bayes with MinMax scaled data
y_pred_prob_multinomial_minmax = multinomial_nb_minmax.predict_proba(x_test_minmax)[:, 1]
fpr_multinomial, tpr_multinomial, thresholds_multinomial = roc_curve(y_test_minmax, y_pred_prob_multinomial_minmax)
auc_multinomial = auc(fpr_multinomial, tpr_multinomial)

# Bernoulli Naive Bayes
y_pred_prob_bernoulli = bernoulli_nb.predict_proba(x_test)[:, 1]
fpr_bernoulli, tpr_bernoulli, thresholds_bernoulli = roc_curve(y_test, y_pred_prob_bernoulli)
auc_bernoulli = auc(fpr_bernoulli, tpr_bernoulli)

print("Gaussian Naive Bayes AUC:", auc_gaussian)
print("Multinomial Naive Bayes AUC (MinMax Scaled):", auc_multinomial)
print("Bernoulli Naive Bayes AUC:", auc_bernoulli)
plt.figure(figsize=(10, 8))
plt.plot(fpr_gaussian, tpr_gaussian, label='Gaussian NB (AUC = %0.2f)' % auc_gaussian)
plt.plot(fpr_multinomial, tpr_multinomial, label='Multinomial NB (AUC = %0.2f)' % auc_multinomial)
plt.plot(fpr_bernoulli, tpr_bernoulli, label='Bernoulli NB (AUC = %0.2f)' % auc_bernoulli)
plt.plot([0, 1], [0, 1], 'k--')  # Diagonal random classifier line
plt.xlabel('False Positive Rate')
```

```
plt.ylabel('True Positive Rate')
plt.title('ROC Curve for Naive Bayes Models')
plt.legend(loc='lower right')
plt.grid(True)
plt.show()

from sklearn.model_selection import cross_val_score

cv_scores = cross_val_score(bernoulli_nb, x, y, cv=5)

i=1
for score in cv_scores:
  print(f"Fold {i} accuracy: {score}")
  i+=1

print("Mean cross-validation accuracy:", cv_scores.mean())
<h1>KNN CLASSIFICATION</h1>
from sklearn.neighbors import KNeighborsClassifier

k_values = [1, 3, 5, 7]

for k in k_values:
    print(f"\nK-Nearest Neighbors (k={k}):")
    knn=KNeighborsClassifier(n_neighbors=k)
    knn.fit(x_train,y_train)
    y_pred_knn=knn.predict(x_test)
    print("Accuracy:", accuracy_score(y_test, y_pred_knn))
    print("Classification Report:")
    print(classification_report(y_test, y_pred_knn))
    cn=confusion_matrix(y_test,y_pred_knn)
    sns.heatmap(cn,annot=True,fmt='d',cmap='Blues',xticklabels=['Not Spam','Spam'],yticklabels=['Not Spam','Spam'])
    plt.xlabel('Predicted')
    plt.ylabel('Actual')
    plt.title(f'Confusion Matrix for k={k}')
    plt.show()
    y_pred_probab=knn.predict_proba(x_test)[:,1]
    fpr,tpr,thresholds=roc_curve(y_test,y_pred_probab)
    auc_score=auc(fpr,tpr)
    print(f"AUC Score for k={k}: {auc_score}")
    plt.figure(figsize=(8,6))
    plt.plot(fpr,tpr,label=f'KNN (AUC={auc_score:.2f})')
    plt.show()
knn_kd = KNeighborsClassifier(n_neighbors=5, algorithm='kd_tree')
knn_kd.fit(x_train, y_train)
y_pred_kd = knn_kd.predict(x_test)
accuracy_kd = accuracy_score(y_test, y_pred_kd)
print(f"Accuracy with KD-Tree: {accuracy_kd:.4f}")
print(classification_report(y_test, y_pred_kd))
# Using Ball Tree
```

```
knn_ball = KNeighborsClassifier(n_neighbors=5, algorithm='ball_tree')
knn_ball.fit(x_train, y_train)
y_pred_ball = knn_ball.predict(x_test)
accuracy_ball = accuracy_score(y_test, y_pred_ball)
print(f"Accuracy with Ball Tree: {accuracy_ball:.4f}")
print(classification_report(y_test, y_pred_ball))

knn=KNeighborsClassifier(n_neighbors=5)
cv_scores = cross_val_score(knn, x, y, cv=5)

i=1
for score in cv_scores:
  print(f"Fold {i} accuracy: {score}")
  i+=1

print("Mean cross-validation accuracy:", cv_scores.mean())
<h1>SVM</h1>

from sklearn.svm import SVC
lsvc=SVC(kernel='linear',C=1.0,random_state=42)
lsvc.fit(x_train,y_train)
y_pred_lsvc=lsvc.predict(x_test)
print("Accuracy:", accuracy_score(y_test, y_pred_lsvc))
print("Classification Report:")
print(classification_report(y_test, y_pred_lsvc))
psvc=SVC(kernel='poly',C=1.0,random_state=42,gamma=0.1,degree=2)
psvc.fit(x_train,y_train)
y_pred_psvc=psvc.predict(x_test)
print("Accuracy:", accuracy_score(y_test, y_pred_psvc))
print("Classification Report:")
print(classification_report(y_test, y_pred_psvc))
p3svc=SVC(kernel='poly',C=1.0,random_state=42,gamma=0.1,degree=3)
p3svc.fit(x_train,y_train)
y_pred_p3svc=p3svc.predict(x_test)
print("Accuracy:", accuracy_score(y_test, y_pred_p3svc))
print("Classification Report:")
print(classification_report(y_test, y_pred_p3svc))
rbfsvc=SVC(kernel='rbf',C=1.0,random_state=42,gamma=0.01)
rbfsvc.fit(x_train,y_train)
y_pred_rbfsvc=rbfsvc.predict(x_test)
print("Accuracy:", accuracy_score(y_test, y_pred_rbfsvc))
print("Classification Report:")
print(classification_report(y_test, y_pred_rbfsvc))
sigmoidsvc=SVC(kernel='sigmoid',C=1.0,random_state=42,gamma=0.01)
sigmoidsvc.fit(x_train,y_train)
y_pred_sigmoidsvc=sigmoidsvc.predict(x_test)
print("Accuracy:", accuracy_score(y_test, y_pred_sigmoidsvc))
print("Classification Report:")
print(classification_report(y_test, y_pred_sigmoidsvc))
cv_score=cross_val_score(lsvc,x,y,cv=5)
```

```
print(cv_score)
print("Mean cross-validation accuracy:", cv_score.mean())
cv_score=cross_val_score(psvc,x,y,cv=5)
print(cv_score)
print("Mean cross-validation accuracy:", cv_score.mean())
cv_score=cross_val_score(p3svc,x,y,cv=5)
print(cv_score)
print("Mean cross-validation accuracy:", cv_score.mean())
cv_score=cross_val_score(rbfsvc,x,y,cv=5)
print(cv_score)
print("Mean cross-validation accuracy:", cv_score.mean())
cv_score=cross_val_score(sigmoidsvc,x,y,cv=5)
print(cv_score)
print("Mean cross-validation accuracy:", cv_score.mean())import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
df=pd.read_csv(r'/content/drive/MyDrive/Ml_Experiment3/spambase_csv.csv')
data=df.copy()
df.columns
df.head()
df.select_dtypes(include='object').columns
df.select_dtypes(include='number').columns
df.isnull().sum()


numerical_cols = df.select_dtypes(include=np.number).columns


n_cols = 3
n_rows = (len(numerical_cols) + n_cols - 1) // n_cols


fig, axes = plt.subplots(n_rows, n_cols, figsize=(15, n_rows * 5))
axes = axes.flatten() # Flatten the 2D array of axes for easy iteration

for i, col in enumerate(numerical_cols):
    sns.boxplot(y=df[col], ax=axes[i])
    axes[i].set_title(col)
    axes[i].set_ylabel('')

for j in range(i + 1, len(axes)):
    fig.delaxes(axes[j])

plt.tight_layout()
plt.show()
from sklearn.preprocessing import StandardScaler
```

```
scaler=StandardScaler()

d_columns=df.columns
d_columns=d_columns.drop('class')
print(d_columns)
df[d_columns]=scaler.fit_transform(df[d_columns])
len(d_columns)
df
from sklearn.model_selection import train_test_split
x=df.drop('class',axis=1)
y=df['class']
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2,
    random_state=42)


from sklearn.naive_bayes import GaussianNB, MultinomialNB,
    BernoulliNB
from sklearn.metrics import accuracy_score, classification_report

# Gaussian Naive Bayes
gaussian_nb = GaussianNB()
gaussian_nb.fit(x_train, y_train)
y_pred_gaussian = gaussian_nb.predict(x_test)

print("Gaussian␣Naive␣Bayes:")
print("Accuracy:", accuracy_score(y_test, y_pred_gaussian))
print(classification_report(y_test, y_pred_gaussian))
bernoulli_nb = BernoulliNB()
bernoulli_nb.fit(x_train, y_train)
y_pred_bernoulli = bernoulli_nb.predict(x_test)

print("\nBernoulli␣Naive␣Bayes:")
print("Accuracy:", accuracy_score(y_test, y_pred_bernoulli))
print(classification_report(y_test, y_pred_bernoulli))
data

from sklearn.preprocessing import MinMaxScaler

# Create a MinMaxScaler
scaler_minmax = MinMaxScaler()

# Apply MinMaxScaler to the features (excluding the 'class'
    column) of the 'data' variable
x_scaled_minmax = scaler_minmax.fit_transform(data.drop('class',
    axis=1))

# Convert the scaled data back to a DataFrame
x_scaled_minmax_df = pd.DataFrame(x_scaled_minmax, columns=data.
    drop('class', axis=1).columns)

# Display the first few rows of the scaled data
```

```python
display(x_scaled_minmax_df.head())
from sklearn.model_selection import train_test_split

# Assuming 'class' is your target variable in the original 'data' DataFrame
y = data['class']

# Split the scaled data into training and testing sets
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)

print("Shape of x_train_minmax:", x_train_minmax.shape)
print("Shape of x_test_minmax:", x_test_minmax.shape)
print("Shape of y_train_minmax:", y_train_minmax.shape)
print("Shape of y_test_minmax:", y_test_minmax.shape)
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score, classification_report

# Multinomial Naive Bayes with MinMax scaled data
multinomial_nb_minmax = MultinomialNB()
multinomial_nb_minmax.fit(x_train_minmax, y_train_minmax)
y_pred_multinomial_minmax = multinomial_nb_minmax.predict(x_test_minmax)

print("Multinomial Naive Bayes with MinMax Scaled Data:")
print("Accuracy:", accuracy_score(y_test_minmax, y_pred_multinomial_minmax))
print(classification_report(y_test_minmax, y_pred_multinomial_minmax))

from sklearn.metrics import confusion_matrix


# Calculate the confusion matrix
cm = confusion_matrix(y_test_minmax, y_pred_multinomial_minmax)

# Display the confusion matrix using seaborn heatmap
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=['Not Spam', 'Spam'], yticklabels=['Not Spam', 'Spam'])
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix for Multinomial')
plt.show()
cm = confusion_matrix(y_test, y_pred_gaussian)

# Display the confusion matrix using seaborn heatmap
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=['Not Spam', 'Spam'], yticklabels=['Not Spam', 'Spam'])
```

```
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix for Gaussian')
plt.show()
cm = confusion_matrix(y_test, y_pred_bernoulli)

# Display the confusion matrix using seaborn heatmap
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=['Not Spam', 'Spam'], yticklabels=['Not Spam', 'Spam'])
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix for bernoulli')
plt.show()
from sklearn.metrics import roc_curve, auc

# Gaussian Naive Bayes
y_pred_prob_gaussian = gaussian_nb.predict_proba(x_test)[:, 1]
fpr_gaussian, tpr_gaussian, thresholds_gaussian = roc_curve(y_test, y_pred_prob_gaussian)
auc_gaussian = auc(fpr_gaussian, tpr_gaussian)

# Multinomial Naive Bayes with MinMax scaled data
y_pred_prob_multinomial_minmax = multinomial_nb_minmax.predict_proba(x_test_minmax)[:, 1]
fpr_multinomial, tpr_multinomial, thresholds_multinomial = roc_curve(y_test_minmax, y_pred_prob_multinomial_minmax)
auc_multinomial = auc(fpr_multinomial, tpr_multinomial)

# Bernoulli Naive Bayes
y_pred_prob_bernoulli = bernoulli_nb.predict_proba(x_test)[:, 1]
fpr_bernoulli, tpr_bernoulli, thresholds_bernoulli = roc_curve(y_test, y_pred_prob_bernoulli)
auc_bernoulli = auc(fpr_bernoulli, tpr_bernoulli)

print("Gaussian Naive Bayes AUC:", auc_gaussian)
print("Multinomial Naive Bayes AUC (MinMax Scaled):", auc_multinomial)
print("Bernoulli Naive Bayes AUC:", auc_bernoulli)
plt.figure(figsize=(10, 8))
plt.plot(fpr_gaussian, tpr_gaussian, label='Gaussian NB (AUC = %0.2f)' % auc_gaussian)
plt.plot(fpr_multinomial, tpr_multinomial, label='Multinomial NB (AUC = %0.2f)' % auc_multinomial)
plt.plot(fpr_bernoulli, tpr_bernoulli, label='Bernoulli NB (AUC = %0.2f)' % auc_bernoulli)
plt.plot([0, 1], [0, 1], 'k--')  # Diagonal random classifier line
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve for Naive Bayes Models')
```

```
plt.legend(loc='lower right')
plt.grid(True)
plt.show()

from sklearn.model_selection import cross_val_score

cv_scores = cross_val_score(bernoulli_nb, x, y, cv=5)

i=1
for score in cv_scores:
  print(f"Fold {i} accuracy: {score}")
  i+=1

print("Mean cross-validation accuracy:", cv_scores.mean())
<h1>KNN CLASSIFICATION</h1>
from sklearn.neighbors import KNeighborsClassifier

k_values = [1, 3, 5, 7]

for k in k_values:
    print(f"\nK-Nearest Neighbors (k={k}):")
    knn=KNeighborsClassifier(n_neighbors=k)
    knn.fit(x_train,y_train)
    y_pred_knn=knn.predict(x_test)
    print("Accuracy:", accuracy_score(y_test, y_pred_knn))
    print("Classification Report:")
    print(classification_report(y_test, y_pred_knn))
    cn=confusion_matrix(y_test,y_pred_knn)
    sns.heatmap(cn,annot=True,fmt='d',cmap='Blues',xticklabels=['Not Spam','Spam'],yticklabels=['Not Spam','Spam'])
    plt.xlabel('Predicted')
    plt.ylabel('Actual')
    plt.title(f'Confusion Matrix for k={k}')
    plt.show()
    y_pred_probab=knn.predict_proba(x_test)[:,1]
    fpr,tpr,thresholds=roc_curve(y_test,y_pred_probab)
    auc_score=auc(fpr,tpr)
    print(f"AUC Score for k={k}: {auc_score}")
    plt.figure(figsize=(8,6))
    plt.plot(fpr,tpr,label=f'KNN (AUC={auc_score:.2f})')
    plt.show()
knn_kd = KNeighborsClassifier(n_neighbors=5, algorithm='kd_tree')
knn_kd.fit(x_train, y_train)
y_pred_kd = knn_kd.predict(x_test)
accuracy_kd = accuracy_score(y_test, y_pred_kd)
print(f"Accuracy with KD-Tree: {accuracy_kd:.4f}")
print(classification_report(y_test, y_pred_kd))
# Using Ball Tree
knn_ball = KNeighborsClassifier(n_neighbors=5, algorithm='ball_tree')
knn_ball.fit(x_train, y_train)
```

```python
y_pred_ball = knn_ball.predict(x_test)
accuracy_ball = accuracy_score(y_test, y_pred_ball)
print(f"Accuracy with Ball Tree: {accuracy_ball:.4f}")
print(classification_report(y_test, y_pred_ball))

knn=KNeighborsClassifier(n_neighbors=5)
cv_scores = cross_val_score(knn, x, y, cv=5)

i=1
for score in cv_scores:
  print(f"Fold {i} accuracy: {score}")
  i+=1

print("Mean cross-validation accuracy:", cv_scores.mean())
<h1>SVM</h1>

from sklearn.svm import SVC
lsvc=SVC(kernel='linear',C=1.0,random_state=42)
lsvc.fit(x_train,y_train)
y_pred_lsvc=lsvc.predict(x_test)
print("Accuracy:", accuracy_score(y_test, y_pred_lsvc))
print("Classification Report:")
print(classification_report(y_test, y_pred_lsvc))
psvc=SVC(kernel='poly',C=1.0,random_state=42,gamma=0.1,degree=2)
psvc.fit(x_train,y_train)
y_pred_psvc=psvc.predict(x_test)
print("Accuracy:", accuracy_score(y_test, y_pred_psvc))
print("Classification Report:")
print(classification_report(y_test, y_pred_psvc))
p3svc=SVC(kernel='poly',C=1.0,random_state=42,gamma=0.1,degree=3)
p3svc.fit(x_train,y_train)
y_pred_p3svc=p3svc.predict(x_test)
print("Accuracy:", accuracy_score(y_test, y_pred_p3svc))
print("Classification Report:")
print(classification_report(y_test, y_pred_p3svc))
rbfsvc=SVC(kernel='rbf',C=1.0,random_state=42,gamma=0.01)
rbfsvc.fit(x_train,y_train)
y_pred_rbfsvc=rbfsvc.predict(x_test)
print("Accuracy:", accuracy_score(y_test, y_pred_rbfsvc))
print("Classification Report:")
print(classification_report(y_test, y_pred_rbfsvc))
sigmoidsvc=SVC(kernel='sigmoid',C=1.0,random_state=42,gamma=0.01)
sigmoidsvc.fit(x_train,y_train)
y_pred_sigmoidsvc=sigmoidsvc.predict(x_test)
print("Accuracy:", accuracy_score(y_test, y_pred_sigmoidsvc))
print("Classification Report:")
print(classification_report(y_test, y_pred_sigmoidsvc))
cv_score=cross_val_score(lsvc,x,y,cv=5)
print(cv_score)
print("Mean cross-validation accuracy:", cv_score.mean())
cv_score=cross_val_score(psvc,x,y,cv=5)
```

```
print(cv_score)
print("Mean␣cross-validation␣accuracy:", cv_score.mean())
cv_score=cross_val_score(p3svc,x,y,cv=5)
print(cv_score)
print("Mean␣cross-validation␣accuracy:", cv_score.mean())
cv_score=cross_val_score(rbfsvc,x,y,cv=5)
print(cv_score)
print("Mean␣cross-validation␣accuracy:", cv_score.mean())
cv_score=cross_val_score(sigmoidsvc,x,y,cv=5)
print(cv_score)
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