# EXPERIMENT 6 – CLASSIFICATION

**AIM**: Introduce basic concepts and methods for classification, including decision tree induction, Bayes classification, and rule-based classification. Also discuss the model evaluation and selection methods for improving classification accuracy, including ensemble methods and how to handle imbalanced data.

**SOFTWARE REQUIRED:**

Spyder IDE 5.1.5

Anaconda3 2021.11 (Python 3.9.7 64-bit)

Anaconda Inc., 2021.11

**DATA SET:** Glaucoma Eye Net Data Set

**PYTHON CODE:**

```
# Importing Necessary Libraries:-
import matplotlib.pyplot as plt # Provides an implicit way of plotting
import numpy as np # Support for large, multi-dimensional arrays and matrices
from sklearn.metrics import accuracy_score # Accuracy classification score
from sklearn.metrics import classification_report # Build a text report
showing the main classification metrics
from sklearn.metrics import confusion_matrix # Compute confusion matrix to
evaluate the accuracy of a classification
from sklearn.metrics import roc_auc_score # Compute Area Under the Receiver
Operating Characteristic Curve (ROC AUC) from prediction scores
from sklearn.metrics import roc_curve # Compute Receiver operating
characteristic (ROC); This implementation is restricted to the binary
classification task
from sklearn.naive_bayes import GaussianNB # Gaussian Naive Bayes (GaussianNB)
from sklearn.neighbors import KNeighborsClassifier # Classifier implementing
the k-nearest neighbors vote
from sklearn.preprocessing import StandardScaler # Standardize features by
removing the mean and scaling to unit variance
from sklearn.tree import DecisionTreeClassifier # A decision tree classifier
from sklearn.svm import SVC # C-Support Vector Classification

import warnings
warnings.filterwarnings('ignore') # Never print matching warnings

def load_dataset_feature_scaling():
```

```
    X_train = np.load("eyenet_features256svm.npy")
    X_test = np.load("eyenet_test_features256svm.npy")
    y_train = np.load("Ytrain256.npy")
    y_test = np.load("Ytest256.npy")

    st_x = StandardScaler() # Standardize features by removing the mean and
scaling to unit variance
    X_train = st_x.fit_transform(X_train) # Fit to data, then transform it
    X_test = st_x.transform(X_test) # Perform standardization by centering and
scaling

    return X_train, X_test, y_train, y_test

def k_nearest_neighbor():

    X_train, X_test, y_train, y_test = load_dataset_feature_scaling()
    neighbors = np.arange(1, 11) # Return evenly spaced values within a given
interval

    # Return a new array of given shape, without initializing entries:-
    train_accuracy = np.empty(len(neighbors))
    test_accuracy = np.empty(len(neighbors))

    # Loop over K values:-
    for i, k in enumerate(neighbors):
        knn = KNeighborsClassifier(n_neighbors=k) # Classifier implementing
the k-nearest neighbors vote
        knn.fit(X_train, y_train) # Fit the k-nearest neighbors classifier
from the training dataset

        # Compute training and test data accuracy - Return the mean accuracy
on the given test data and labels:-
        train_accuracy[i] = knn.score(X_train, y_train)
        test_accuracy[i] = knn.score(X_test, y_test)

    # Generate plot:-
    plt.xlabel("n_neighbors"); plt.ylabel("Accuracy")
    plt.title("K-Nearest Neighbor (KNN) - Model Accuracy")
    plt.plot(neighbors, test_accuracy, label = 'Testing Dataset Accuracy')
    plt.plot(neighbors, train_accuracy, label = 'Training Dataset Accuracy')
    plt.legend(); plt.grid(True); plt.show()

# Function to make predictions:-
def prediction(X_test, clf_object):

    # Predicton on test with gini index:-
    y_pred = clf_object.predict(X_test)
    print("Predicted Values -"); print(y_pred)
```

```
    return y_pred

# Function to calculate accuracy:-
def cal_accuracy(y_test, y_pred):

    print("\nConfusion Matrix -\n", confusion_matrix(y_test, y_pred)) #
Compute confusion matrix to evaluate the accuracy of a classification

    print ("\nAccuracy - ", accuracy_score(y_test, y_pred)*100) # Accuracy
classification score

    print("Report -\n", classification_report(y_test, y_pred)) # Build a text
report showing the main classification metrics

def naive_bayes_classifier():

    X_train, X_test, y_train, y_test = load_dataset_feature_scaling()

    # Fitting Naive Bayes to the training set:-
    classifier = GaussianNB() # Gaussian Naive Bayes (GaussianNB)
    classifier.fit(X_train, y_train[:, 0]) # Fit Gaussian Naive Bayes
according to X, y
    y_pred= classifier.predict(X_test) # Predict the test set result
    cal_accuracy(y_test[:, 0], y_pred) # Function to calculate accuracy

def support_vector_machine():

    X_train, X_test, y_train, y_test = load_dataset_feature_scaling()

    # Fitting the SVM classifier to the training set:-
    classifier = SVC(kernel='linear', random_state=0) # C-Support Vector
Classification
    classifier.fit(X_train, y_train[:, 0]) # Fit the SVM model according to
the given training data
    y_pred = classifier.predict(X_test) # Predict the test set result
    cal_accuracy(y_test[:, 0], y_pred) # Function to calculate accuracy

def decision_tree():

    def train_using_gini(X_train, X_test, y_train):

        clf_gini = DecisionTreeClassifier(criterion = "gini", random_state =
100, max_depth = 3, min_samples_leaf = 5) # Create the classifier object
        clf_gini.fit(X_train, y_train[:, 0]) # Perform training
        return clf_gini

    def train_using_entropy(X_train, X_test, y_train):
```

```
        clf_entropy = DecisionTreeClassifier(criterion = "entropy",
random_state = 100,    max_depth = 3, min_samples_leaf = 5) # Decision tree
with entropy
        clf_entropy.fit(X_train, y_train[:, 0]) # Perform training
        return clf_entropy

    X_train, X_test, y_train, y_test = load_dataset_feature_scaling()
    clf_gini = train_using_gini(X_train, X_test, y_train)
    clf_entropy = train_using_entropy(X_train, X_test, y_train)

    print("\n\t\t\t\t\t\t\t\t(i) Results Using Gini Index:-\n")
    y_pred_gini = prediction(X_test, clf_gini) # Function to make predictions
    cal_accuracy(y_test[:, 0], y_pred_gini) # Function to calculate accuracy

    print("\n\t\t\t\t\t\t\t\t(ii) Results Using Entropy:-\n")
    y_pred_entropy = prediction(X_test, clf_entropy) # Function to make
predictions
    cal_accuracy(y_test[:, 0], y_pred_entropy) # Function to calculate
accuracy

def auc_roc_curve():

    model1 = KNeighborsClassifier(n_neighbors=3) # Classifier implementing the
k-nearest neighbors vote
    model2 = GaussianNB() # Gaussian Naive Bayes (GaussianNB)
    model3 = SVC(kernel='linear', random_state=0, probability=True) # C-
Support Vector Classification
    model4 = DecisionTreeClassifier(criterion = "gini", random_state = 100,
max_depth = 3, min_samples_leaf = 5) # A decision tree classifier
    model5 = DecisionTreeClassifier(criterion = "entropy", random_state =
100,    max_depth = 3, min_samples_leaf = 5) # A decision tree classifier

    X_train, X_test, y_train, y_test = load_dataset_feature_scaling()

    # Fit Model:-
    model1.fit(X_train, y_train[:,0])
    model2.fit(X_train, y_train[:,0])
    model3.fit(X_train, y_train[:,0])
    model4.fit(X_train, y_train[:,0])
    model5.fit(X_train, y_train[:,0])

    # Predict Probabilities:-
    pred_prob1 = model1.predict_proba(X_test)
    pred_prob2 = model2.predict_proba(X_test)
    pred_prob3 = model3.predict_proba(X_test)
    pred_prob4 = model4.predict_proba(X_test)
    pred_prob5 = model5.predict_proba(X_test)
```

```python
    # ROC Curve For Models:-
    fpr1, tpr1, thresh1 = roc_curve(y_test[:,0], pred_prob1[:,1], pos_label=1)
    fpr2, tpr2, thresh2 = roc_curve(y_test[:,0], pred_prob2[:,1], pos_label=1)
    fpr3, tpr3, thresh3 = roc_curve(y_test[:,0], pred_prob3[:,1], pos_label=1)
    fpr4, tpr4, thresh4 = roc_curve(y_test[:,0], pred_prob4[:,1], pos_label=1)
    fpr5, tpr5, thresh5 = roc_curve(y_test[:,0], pred_prob5[:,1], pos_label=1)

    # ROC Curve for tpr = fpr:-
    random_probs = [0 for i in range(len(y_test))]
    p_fpr, p_tpr, _ = roc_curve(y_test[:,0], random_probs, pos_label=1)

    # AUC Scores:-
    auc_score1 = roc_auc_score(y_test[:,0], pred_prob1[:,1])
    auc_score2 = roc_auc_score(y_test[:,0], pred_prob2[:,1])
    auc_score3 = roc_auc_score(y_test[:,0], pred_prob3[:,1])
    auc_score4 = roc_auc_score(y_test[:,0], pred_prob4[:,1])
    auc_score5 = roc_auc_score(y_test[:,0], pred_prob5[:,1])

    print("\nAUC Scores-")
    print("(i) K-Nearest Neighbor (KNN): ", auc_score1)
    print("(ii) Naive Bayes Classifier: ", auc_score2)
    print("(iii) Support Vector Machine (SVM): ", auc_score3)
    print("(iv) Decision Tree Using Gini Index: ", auc_score4)
    print("(v) Decision Tree Using Entropy: ", auc_score5)

    # Plot ROC Curves:-
    plt.style.use('seaborn')

    plt.plot(fpr1, tpr1, linestyle='--', color='red', label='K-Nearest 
Neighbor (KNN): ' + str(round(auc_score1*100, 2)) + '%')
    plt.plot(fpr2, tpr2, linestyle='--', color='blue', label='Naive Bayes 
Classifier: ' + str(round(auc_score2*100, 2)) + '%')
    plt.plot(fpr3, tpr3, linestyle='--', color='green', label='Support Vector 
Machine (SVM): ' + str(round(auc_score3*100, 2)) + '%')
    plt.plot(fpr4, tpr4, linestyle='--', color='magenta', label='Decision Tree 
Using Gini Index: ' + str(round(auc_score4*100, 2)) + '%')
    plt.plot(fpr5, tpr5, linestyle='--', color='brown', label='Decision Tree 
Using Entropy: ' + str(round(auc_score5*100, 2)) + '%')
    plt.plot(p_fpr, p_tpr, linestyle='--', color='yellow')

    plt.xlabel("False Positive Rate"); plt.ylabel("True Positive rate")
    plt.title("ROC Curve"); plt.legend(loc='best'); plt.show()

# Driver Code: main():-
def main():

    print("\n"); heading = "K-Nearest Neighbor (KNN)"
    print('{:s}'.format('\u0332'.join(heading.center(100))))
```

```python
    k_nearest_neighbor()

    print("\n"); heading = "Naive Bayes Classifier"
    print('{:s}'.format('\u0332'.join(heading.center(100))))
    naive_bayes_classifier()

    print("\n"); heading = "Decision Tree"
    print('{:s}'.format('\u0332'.join(heading.center(100))))
    decision_tree()

    print("\n"); heading = "Support Vector Machine (SVM)"
    print('{:s}'.format('\u0332'.join(heading.center(100))))
    support_vector_machine()

    print("\n"); heading = "AUC-ROC Curve"
    print('{:s}'.format('\u0332'.join(heading.center(100))))
    auc_roc_curve()

# Call main function ; Execution starts here.
if __name__=="__main__":
    main()
```

## CLASSIFIERS’ OUTCOMES SUMMARIZED:

| | Naive Bayes Classifier | | | | |
|---|---|---|---|---|---|
| Confusion Matrix | 26 | 14 | | | |
| | 15 | 36 | | | |
| Accuracy | 68.13186813 | | | | |
| Report | | Precision | Recall | F1-Score | Support |
| | 0 | 0.63 | 0.65 | 0.64 | 40 |
| | 1 | 0.72 | 0.71 | 0.71 | 51 |
| | Accuracy | | | 0.68 | 91 |
| | Macro Average | 0.68 | 0.68 | 0.68 | 91 |
| | Weighted Average | 0.68 | 0.68 | 0.68 | 91 |
| AUC Score | 0.677941176 | | | | |

| | Decision Tree Using Gini Index | | | | |
|---|---|---|---|---|---|
| Confusion Matrix | 27 | 13 | | | |
| | 12 | 39 | | | |
| Accuracy | 72.52747253 | | | | |
| Report | | Precision | Recall | F1-Score | Support |
| | 0 | 0.69 | 0.68 | 0.68 | 40 |
| | 1 | 0.75 | 0.76 | 0.76 | 51 |
| | Accuracy | | | 0.73 | 91 |
| | Macro Average | 0.72 | 0.72 | 0.72 | 91 |
| | Weighted Average | 0.72 | 0.73 | 0.72 | 91 |
| AUC Score | 0.758333333 | | | | |

| | Decision Tree Using Entropy | | | | |
|---|---|---|---|---|---|
| Confusion Matrix | 28 | 12 | | | |
| | 12 | 39 | | | |
| Accuracy | 73.62637363 | | | | |
| Report | | Precision | Recall | F1-Score | Support |
| | 0 | 0.7 | 0.7 | 0.7 | 40 |
| | 1 | 0.76 | 0.76 | 0.76 | 51 |
| | Accuracy | | | 0.74 | 91 |
| | Macro Average | 0.73 | 0.73 | 0.73 | 91 |
| | Weighted Average | 0.74 | 0.74 | 0.74 | 91 |
| AUC Score | 0.793872549 | | | | |

| | Support Vector Machine (SVM) | | | | |
|---|---|---|---|---|---|
| Confusion Matrix | 34 | 6 | | | |
| | 5 | 46 | | | |
| Accuracy | 87.91208791 | | | | |
| Report | | Precision | Recall | F1-Score | Support |
| | 0 | 0.87 | 0.85 | 0.86 | 40 |
| | 1 | 0.88 | 0.9 | 0.89 | 51 |
| | Accuracy | | | 0.88 | 91 |
| | Macro Average | 0.88 | 0.88 | 0.88 | 91 |
| | Weighted Average | 0.88 | 0.88 | 0.88 | 91 |
| AUC Score | 0.935784314 | | | | |

## PLOTS:

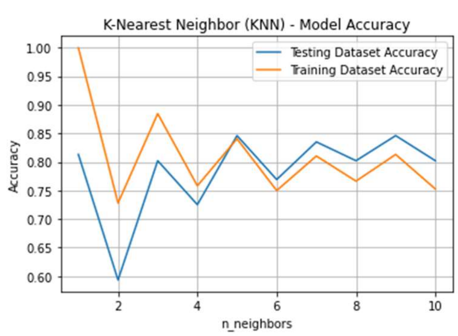


Figure 1. K-Nearest Neighbour (KNN) - Model Accuracy

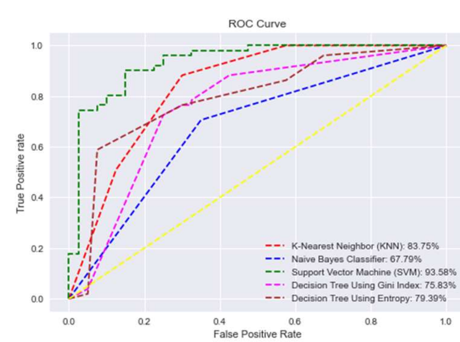


Figure 2. ROC Curve

**RESULT:**

Thus, described methods for data classification and prediction, including decision tree induction, Bayesian classification, support vector machine and k-nearest classifiers. Issues regarding accuracy and how to choose best classifier or predictor are discussed. All the simulation results were verified successfully.

```
Python 3.9.7 (default, Sep 16 2021, 16:59:28) [MSC v.1916 64 bit (AMD64)]
Type "copyright", "credits" or "license" for more information.

IPython 7.29.0 -- An enhanced Interactive Python.

Restarting kernel...
```

---

```
In [1]:         'E:/Plan B/Amrita Vishwa Vidyapeetham/Subject Materials/Semester IV/19CCE213 - Machine Learning and Artificial Intelligence/Lab/Experiment 6 - Classification/Expt_6_Code_Glaucoma.py'      ='E:/Plan B/Amrita Vishwa Vidyapeetham/Subject Materials/Semester IV/19CCE213 - Machine Learning and Artificial Intelligence/Lab/Experiment 6 - Classification'


 ________________________________________K-Nearest Neighbor (KNN)_


 _________________________________________Naive Bayes Classifier

Confusion Matrix -
 [[26 14]
 [15 36]]

Accuracy -  68.13186813186813
Report -
              precision    recall  f1-score   support

         0.0       0.63      0.65      0.64        40
         1.0       0.72      0.71      0.71        51

    accuracy                           0.68        91
   macro avg       0.68      0.68      0.68        91
weighted avg       0.68      0.68      0.68        91



 _____________________________________________Decision Tree

                           (i) Results Using Gini Index:-

Predicted Values -
[1. 1. 0. 1. 1. 0. 1. 1. 0. 0. 1. 1. 1. 1. 0. 1. 0. 1. 0. 1. 1. 1. 0. 1.
 0. 1. 1. 0. 0. 0. 0. 0. 0. 1. 0. 1. 0. 0. 1. 1. 1. 1. 0. 0. 1. 1. 1. 1.
 1. 0. 0. 1. 1. 1. 1. 0. 1. 0. 1. 1. 1. 0. 0. 0. 0. 1. 1. 0. 1. 1. 0. 1.
 1. 1. 1. 0. 0. 0. 1. 0. 1. 0. 0. 1. 0. 1. 1. 0. 1. 0. 1.]

Confusion Matrix -
 [[27 13]
 [12 39]]

Accuracy -  72.52747252747253
Report -
              precision    recall  f1-score   support
```

```
         0.0       0.69      0.68      0.68        40
         1.0       0.75      0.76      0.76        51

    accuracy                           0.73        91
   macro avg       0.72      0.72      0.72        91
weighted avg       0.72      0.73      0.72        91
```

(ii) Results Using Entropy:-

```
Predicted Values -
[1. 1. 0. 1. 1. 1. 1. 1. 0. 0. 1. 1. 1. 0. 0. 1. 0. 1. 0. 1. 1. 1. 0. 1.
 0. 1. 1. 0. 0. 0. 0. 1. 0. 0. 0. 1. 0. 0. 1. 1. 0. 1. 0. 0. 1. 1. 1. 1.
 1. 1. 0. 1. 1. 1. 1. 0. 1. 0. 1. 1. 1. 0. 0. 0. 0. 1. 1. 0. 1. 1. 0. 0.
 1. 1. 0. 0. 0. 0. 1. 1. 1. 0. 0. 1. 0. 1. 1. 0. 1. 0. 1.]

Confusion Matrix -
 [[28 12]
 [12 39]]

Accuracy -  73.62637362637363
Report -
              precision    recall  f1-score   support

         0.0       0.70      0.70      0.70        40
         1.0       0.76      0.76      0.76        51

    accuracy                           0.74        91
   macro avg       0.73      0.73      0.73        91
weighted avg       0.74      0.74      0.74        91
```

## Support Vector Machine (SVM)

```
Confusion Matrix -
 [[34  6]
 [ 5 46]]

Accuracy -  87.91208791208791
Report -
              precision    recall  f1-score   support

         0.0       0.87      0.85      0.86        40
         1.0       0.88      0.90      0.89        51

    accuracy                           0.88        91
   macro avg       0.88      0.88      0.88        91
weighted avg       0.88      0.88      0.88        91
```

## AUC-ROC Curve

```
AUC Scores-
(i) K-Nearest Neighbor (KNN):  0.8375
(ii) Naive Bayes Classifier:  0.6779411764705883
(iii) Support Vector Machine (SVM):  0.9357843137254902
(iv) Decision Tree Using Gini Index:  0.7583333333333333
(v) Decision Tree Using Entropy:  0.7938725490196079
```

In [2]: