DEPARTMENT OF COMPUTER APPLICATIONS COCHIN UNIVERSITY OF SCIENCE AND TECHNOLOGY COCHIN-22

MSC COMPUTER SCIENCE

SPECIALISATION IN DATA SCIENCE

MINI PROJECT ON

IMPORTANCE OF FEATURE SELECTION ALGORITHMS IN MACHINE LEARNING

APRIL- 23

DEPARTMENT OF COMPUTER APPLICATIONS

COCHIN UNIVERSITY OF

SCIENCE AND TECHNOLOGY COCHIN-22

CERTIFICATE

This is to certify that the mini project entitled “Importance of Feature selection in Machine Learning” is a bonafide record of the project work done by, HARIKRISHNAN S (Reg No: - 35922014), under our supervision and guidance in partial fulfilment of the award of Degree of Masters of Science in Computer Science during the year 2022-24 at Department of Computer Applications, Cochin University of Science and Technology, Cochin.

Internal Examiner

Department of Computer Applications, CUSAT

Head of the Department

Department of Computer Applications, CUSAT

ACKNOWLEDGEMENT

I would like to express my sincere gratitude to the various individuals who have provided their help throughout my project. Their contributions in terms of inspiration and technical assistance have been invaluable.

I would also like to extend my deepest appreciation to Almighty God for granting me the ability to undertake and successfully complete the main project.

I am pleased to acknowledge our indebtedness to **Dr. M.V. JUDY**, Head of the Department, Department of Computer Application, CUSAT, for her gracious encouragement. Additionally, I would like to express my heartfelt gratitude and thanks to our class coordinator and former HOD, **Dr. Sabu.M.K**, for their support and guidance. Special thanks are also due to **Mr. Manikandan**, our project coordinator. I would also like to extend my gratitude to **Mr. Keerthimon Prabhalachandran** for his insights and aid since the initiation of the project.

I am also thankful to the teaching staff for their helpfulness and cooperation during the project period. Their support has been instrumental in the successful completion of the project.

Lastly, I would like to extend my heartfelt thanks to my parents, friends, and well-wishers for their continuous support and timely assistance. Their encouragement has been a great source of motivation throughout this endeavor.

ABSTRACT

The aim of this project is to determine the best combination of feature selection and classifier algorithm for different types of datasets. The project will use three different datasets with different characteristics to explore the effectiveness of different feature selection techniques and classifier algorithms in improving model performance.

The project will start by collecting a diverse set of datasets, namely multivariate dataset, categorical dataset, and numerical dataset. Each dataset will be preprocessed to remove noise, handle missing values, and normalise the data. Then, a set of three feature selection techniques, including Linear regression coefficient,Random forest and Principal Component Analysis, will be applied to each dataset to select the most relevant features.

Next, several classifier algorithms, including support vector machine, decision trees, k-nearest neighbour, will be used to build classification models on the selected features. The performance of each model will be evaluated using metrics such as accuracy, precision and recall.The project will compare the performance of different feature selection techniques and classifier algorithms on each dataset to determine the best combination for that particular dataset.

The outcome of this project will be a set of guidelines on the best combination of feature selection and classifier algorithms for different types of datasets. The project can be further used to improve accuracy and efficiency of machine learning models.

TABLE OF CONTENTS

INTRODUCTION

In the realm of data science ,feature selection plays a crucial role in extracting meaningful insights and building robust predictive models.With the exponential growth of data and the increasing complexity of algorithms, identifying the most relevant features from a vast pool of variables becomes a significant challenge. Feature selection is a critical step in data preprocessing, where the aim is to reduce dimensionality, enhance model interpretability, and improve prediction accuracy.

The process of feature selection involves choosing a subset of relevant features that have the most significant impact on the target variable, while discarding irrelevant or redundant ones. By eliminating irrelevant features, we can mitigate the risk of overfitting, reduce computational costs, and enhance model performance. Moreover, feature selection helps us gain valuable insights into the underlying relationships between variables, enabling us to uncover meaningful patterns and uncover hidden trends within the data.

There are various feature selection techniques available. Some popular approaches include filter methods, wrapper methods, and embedded methods.Feature selection techniques can be applied to a wide range of data science projects across domains such as finance, healthcare, marketing, and more. In finance, for example, feature selection can help identify the key variables that drive stock prices or predict market trends. In healthcare, it can aid in selecting vital biomarkers for disease diagnosis or identifying risk factors. Similarly, in marketing, feature selection can assist in targeting the most influential customer attributes for personalised advertising campaigns.

In summary, feature selection is a fundamental aspect of data science projects, allowing us to uncover relevant patterns, enhance model performance, and

improve interpretability. By leveraging the right techniques and methodologies, data scientists can effectively identify the subset of features that have the most substantial impact on the problem at hand. Ultimately, this leads to more accurate predictions, better insights, and informed decision-making in a wide range of industries.

1. FEATURE SELECTION

Feature selection is the process of selecting a subset of relevant features or variables from a larger set of features that are available in a dataset. It is the process of identifying the most important and informative features that contribute to the predictive performance of a machine learning model.

The goal of feature selection is to improve the performance of a model by reducing the dimensionality of the dataset, while maintaining or improving the accuracy of the model.

1.1 DIFFERENT APPROACHES TO FEATURE SELECTION

1.Filter methods: This method involves selecting features based on statistical measures, such as correlation or mutual information, without taking into account the performance of the model.

2.Wrapper methods: On the other hand, use the model's performance as a criterion for feature selection by evaluating the model with different subsets of features.

3.Embedded methods: This method integrates feature selection into the model-building process, such as with regularisation techniques like Lasso or Ridge regression.

1.1 IMPORTANCE OF FEATURE SELECTION

1. It can help to reduce overfitting.
2. Improve model interpretability.
3. Reduce computational costs.
4. Reduces training time.
5. It is also useful for identifying the most relevant features for a particular problem, which can lead to a better understanding of the underlying data and potentially lead to new insights.

2. FEATURE SELECTION ALGORITHMS AND ITS METHODOLOGY

2.1 DECISION TREE

A decision tree is a machine learning algorithm that is commonly used for both classification and regression tasks. It is a supervised learning method that learns a hierarchical structure of decision rules based on the features of the input data.

The decision tree feature selection algorithm follows a recursive process to select the most important features for a given task. The basic idea is to construct a decision tree that maximally separates the data based on the features, and then use the tree structure to rank the importance of the features.

The algorithm starts with all features and constructs a decision tree using the entire dataset. It then evaluates the importance of each feature based on how much it contributes to the overall accuracy of the tree. The feature with the highest importance is selected and the tree is re-built using only that feature. This process is repeated until a predetermined number of features are selected, or until the desired level of accuracy is achieved.

2.2 RANDOM FOREST

Random forest is a machine learning algorithm that is commonly used for classification, regression, and feature selection tasks. It is an ensemble learning method that combines multiple decision trees to improve the accuracy and stability of the model.

In a random forest, each decision tree is built using a random subset of the training data and a random subset of the features. This helps to reduce the risk of overfitting, as each tree is built using a slightly different subset of the data and features. The final prediction of the random forest is then based on the combined

predictions of all the individual decision trees.

After constructing the decision trees, the algorithm ranks the importance of each feature based on how much the accuracy of the random forest model is reduced when that feature is removed. The feature with the lowest importance score is then eliminated, and the process is repeated until a predetermined number of features is selected.

2.3 RECURSIVE ELIMINATION

Recursive Feature Elimination is a feature selection method used in machine learning. It is a wrapper method that works by recursively removing features and building a model on the remaining features until a desired number of features is reached or the performance of the model stops improving.

In RFE, a machine learning algorithm is trained on the entire dataset with all the features. The feature weights are then calculated, and the least important feature(s) are removed from the dataset. The algorithm is retrained on the reduced dataset, and the feature weights are recalculated. This process is repeated until the desired number of features is reached.

In this method, a Ridge Regression model is first trained on the entire dataset, and the feature coefficients are used to rank the features based on their importance. The feature with the smallest coefficient is then eliminated, and a new Ridge Regression model is trained on the remaining features. The process of elimination and model training is repeated until the desired number of features is selected. The Ridge Regression model is used as the estimator because it shrinks the coefficients of less important features towards zero, making it well-suited for feature selection tasks.

COMPARISON OF ADVANTAGES

Decision Tree	Random Forest	Recursive Feature Elimination
1. Interpretability 2. Handling both categorical and numerical data. 3. Nonlinear relationships 4. Scalability 5. Flexibility 6. Little data preparation	1. Robustness 2. Handling missing values and outliers 3. Versatility 4. Scalability 5. Accuracy 6. Feature Importance 7. Resilience to noisy data	1. Robustness 2. Improved model performance 3. Versatility 4. Compatibility with various models 5. Flexibility 6. Feature ranking

COMPARISON OF DISADVANTAGES

Decision Tree	Random Forest	Recursive Feature Elimination
1. Overfitting 2. Instability 3. Bias 4. Limited expressiveness 5. Imbalanced classes 6. Greedy nature 7. Inconsistency with small changes	1. Overfitting 2. Complexity 3. Bias towards categorical variables 4. Computationally expensive 5. Memory usage 6. Lack of transparency 7. Poor performance on rare events	1. Computational complexity 2. Hyperparameter tuning 3. Performance metric dependency 4. Model dependence 5. Feature interactions 6. Large feature sets 7. Subjectivity

3.CLASSIFIER AND ITS METHODOLOGY

3.1 SUPPORT VECTOR MACHINE

SVM stands for Support Vector Machines, which is a supervised learning algorithm used for classification, regression, and outlier detection tasks in machine learning.

The main idea behind SVM is to find the hyperplane that best separates the data points into different classes. In a two-class classification problem, the hyperplane is the decision boundary that maximises the margin, which is the distance between the hyperplane and the closest data points from each class. SVM can handle both linearly separable and non-linearly separable data by using different types of kernels to transform the data into a higher dimensional space.

The SVM algorithm works by finding the optimal solution of a constrained

optimization problem, which involves minimising the classification error and maximising the margin. The optimization problem can be solved using different techniques such as quadratic programming, gradient descent, or interior point methods.

3.2 DECISION TREE

A decision tree classifier is a type of supervised machine learning algorithm that is commonly used for classification tasks. The algorithm works by recursively partitioning the data into subsets based on the values of different features, in order to create a tree-like model that can be used for classification.

At each node of the decision tree, the algorithm selects the feature that provides the most information gain, which is typically calculated using measures such as entropy or Gini impurity. The feature with the highest information gain is used to split the data into two or more subsets, which are then processed recursively until a

stopping criterion is met (e.g. a maximum depth of the tree is reached or a minimum number of instances in a leaf node is reached).

Once the decision tree has been constructed, it can be used to predict the class of new instances by traversing the tree from the root node to a leaf node, using the values of the features to determine which path to take at each node. The class assigned to the leaf node reached by the traversal is the predicted class of the new instance.

3.3 K-NEAREST NEIGHBORS

KNN (K-Nearest Neighbors) is a simple, non-parametric algorithm used for classification and regression tasks. The basic idea behind the KNN algorithm is to find the K nearest data points to a new data point in the feature space, and then use the majority class (in the case of classification) of those K neighbors to predict the class or value of the new data point.

In the KNN algorithm, the distance between two data points is typically measured using Euclidean distance or some other distance metric. The value of K is a hyperparameter that must be chosen before training the algorithm, and it determines how many neighbours to consider when making a prediction.

COMPARISON OF ADVANTAGES

Support Vector Machine	Decision Tree	K-Nearest Neighbors
1. Effective in high-dimensional spaces. 2. Good generalisation performance. 3. Robust to noise. 4. Versatility. 5. Effective with small datasets. 6. Control overfitting	1. Easy to interpret 2. Fast to train 3. Handle both categorical and numerical data 4. Robust to noise 5. Non-parametric 6. Feature selection 7. Can be used for both classification and regression	1. Simple to understand and implement 2. No training required 3. Robust to noisy data 4. Versatile 5. Non-parametric 6. Adaptability 7. High accuracy

COMPARISON OF DISADVANTAGES

Support Vector Machine	Decision Tree	K-Nearest Neighbors
1. Sensitivity to parameter settings. 2. Computationally intensive. 3. Limited interpretability. 4. Data scaling. 5. Binary classification. 6. Imbalanced datasets. 7. Memory-intensive.	1. Overfitting 2. Instability 3. Bias 4. Limited expressiveness 5. Imbalanced classes 6. Greedy nature 7. Inconsistency with small changes	1. Sensitivity to feature scaling 2. Computationally intensive 3. Choice of K 4. Not suitable for high-dimensional data 5. Imbalanced datasets 6. Requires labeled data

5.DATASETS USED

Iris

- Multivariate data set.
- Contains a set of 150 records .
- Attributes - Petal Length, Petal Width, Sepal Length, Sepal width; Class - Species.

Diabetes

- Numeric data set
- Contains 768 records
- Attributes - Pregnancies, Glucose, Blood Pressure, Skin Thickness, Insulin, BMI, Diabetes Pedigree Function, Age.
- Classes - 0 (tested negative), 1 (tested positive)

Wine Quality

- Categorical data
- Describes the amount of various chemicals present in wine and their effect on its quality.
- Attributes - fixed acidity, volatile acidity, citric acid, residual sugar, chlorides, free sulphur dioxide, total sulphur dioxide, density, pH, sulphates, alcohol.
- Classes - Quality (score between 0 and 10)

6. EXPERIMENT ANALYSIS AND OUTPUTS

6.1 IRIS DATASET

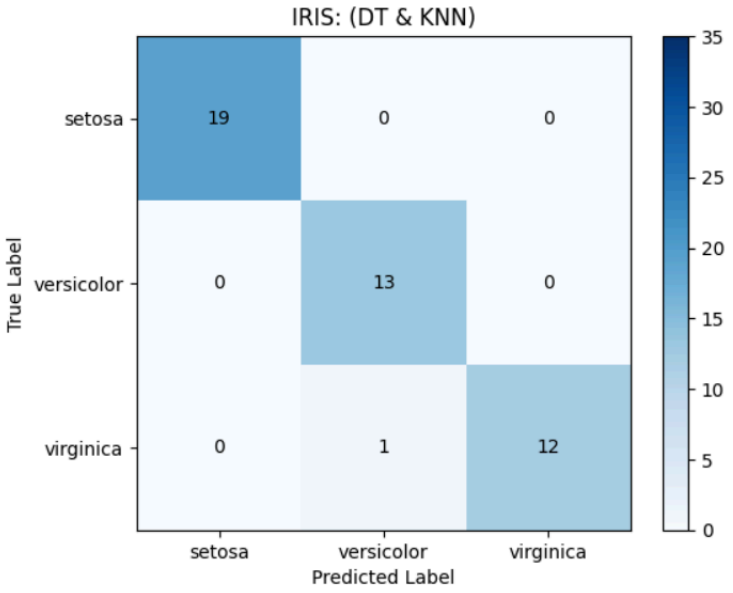
In [16]:
```
from sklearn.ensemble import RandomForestClassifier
from sklearn.feature_selection import SelectFromModel
import numpy as np
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
from sklearn import svm
import pandas as pd
from sklearn.metrics import confusion_matrix
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from sklearn.linear_model import Ridge
from sklearn.feature_selection import RFECV

# Load the iris dataset
iris = load_iris()
X, y = iris.data, iris.target
dtc = DecisionTreeClassifier(random_state=42)

# Use SelectFromModel to select the most important features
sfm = SelectFromModel(dtc, threshold='median')
X_selected = sfm.fit_transform(X, y)


# Print the selected features
selected_features = np.array(iris.feature_names)[sfm.get_support()]
print("Selected Features: ", selected_features)
X_new = pd.DataFrame(X_selected, columns=selected_features)
X_new["Species"]=iris["target_names"][y]
X_new.to_csv('iris_dt.csv', index=False)
```

```
Selected Features:  ['petal length (cm)' 'petal width (cm)']
```

In [21]:
```
#Knn
X_new = pd.read_csv("iris_dt.csv")
x = X_new.drop(X_new.columns[len(X_new.columns)-1],axis=1)
y = X_new[X_new.columns[len(X_new.columns)-1]]
#print(x,"\n",y)
# Split the dataset into training and testing sets
X_train_k, X_test_k, y_train_k, y_test_k = train_test_split(x, y, test_size=0.3, random_state=42)

# Create a KNN classifier with k=3
knn = KNeighborsClassifier(n_neighbors=2)

# Train the classifier on the training data
knn.fit(X_train_k, y_train_k)

# Predict the classes of the test set
y_pred_k = knn.predict(X_test_k)

# Calculate the accuracy of the classifier
accuracy_k = accuracy_score(y_test_k, y_pred_k)
print("Accuracy of KNN:", accuracy_k*100)
# Calculate the confusion matrix
cm_k = confusion_matrix(y_test_k, y_pred_k, labels=['setosa', 'versicolor', 'virginica'])

# Plot the confusion matrix
plt.imshow(cm_k, cmap=plt.cm.Blues,vmin=0,vmax=35)
for i in range(cm_k.shape[0]):
    for j in range(cm_k.shape[1]):
        plt.text(j, i, str(cm_k[i,j]), horizontalalignment='center', verticalalignment='center')

plt.title('IRIS: (DT & KNN)')
plt.colorbar()
plt.xticks([0,1,2], labels=['setosa', 'versicolor', 'virginica'])
plt.yticks([0,1,2], labels=['setosa', 'versicolor', 'virginica'])
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.savefig('IRIS_DT_KNN.png')
```

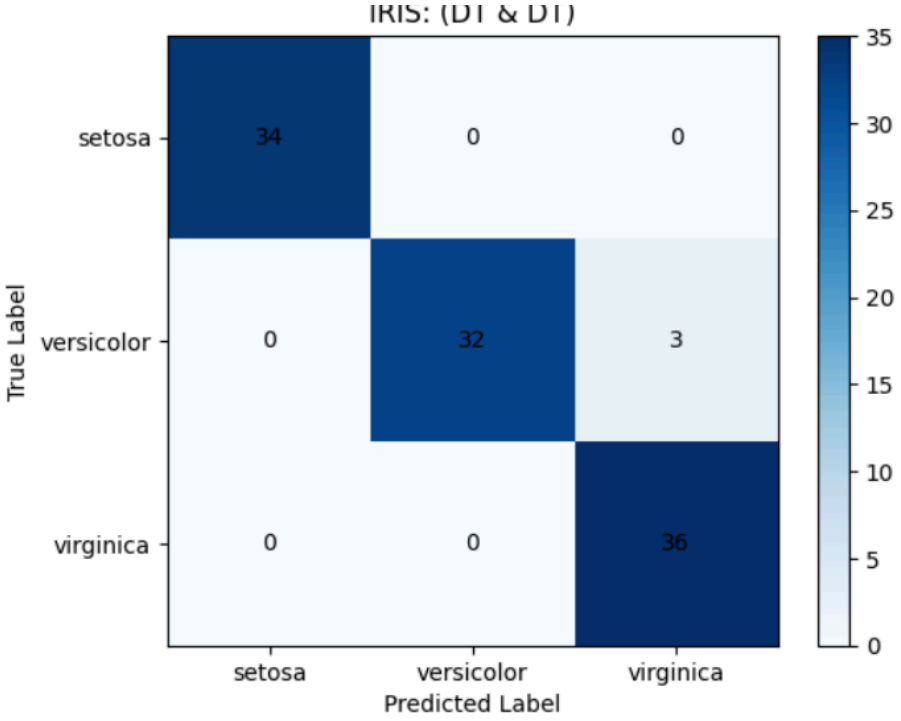
```
plt.savefig('IRIS_DT_KNN.png')
plt.show()
```

```
Accuracy of KNN: 97.77777777777777
```

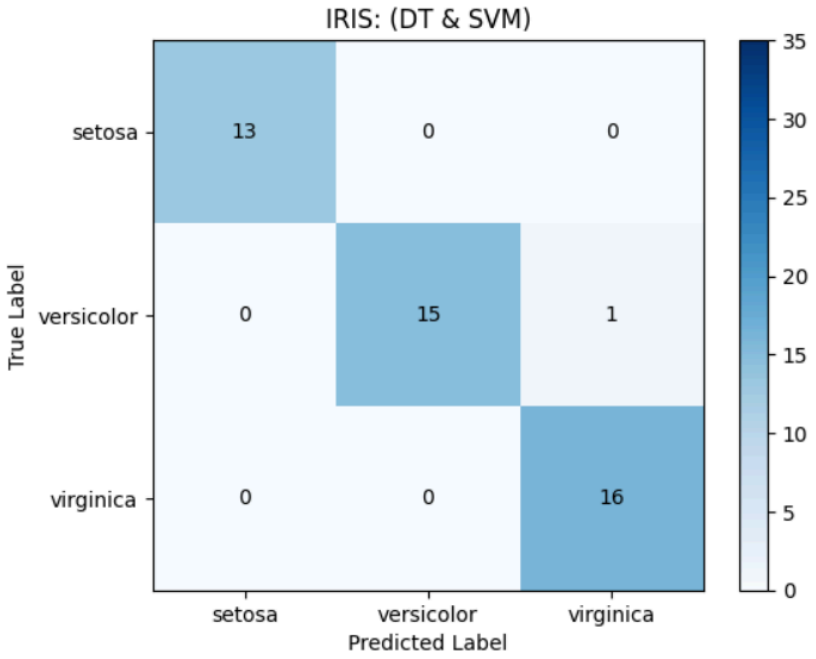


In [22]:
```
#Decision Tree
# Split the dataset into training and testing sets
X_train_dt, X_test_dt, y_train_dt, y_test_dt = train_test_split(x, y, test_size=0.7)

# Train a decision tree classifier
clf = DecisionTreeClassifier()
clf.fit(X_train_dt, y_train_dt)

# Predict the classes of the test set
y_pred_dt = clf.predict(X_test_dt)

# Evaluate the accuracy of the classifier
accuracy_dt = accuracy_score(y_test_dt, y_pred_dt)
print("Accuracy of Decision Tree Classifier:", accuracy_dt*100)
# Calculate the confusion matrix
cm_dt = confusion_matrix(y_test_dt, y_pred_dt, labels=['setosa', 'versicolor', 'virginica'])

# Plot the confusion matrix
plt.imshow(cm_dt, cmap=plt.cm.Blues,vmin=0,vmax=35)
for i in range(cm_dt.shape[0]):
    for j in range(cm_dt.shape[1]):
        plt.text(j, i, str(cm_dt[i,j]), horizontalalignment='center', verticalalignment='center')

plt.title('IRIS: (DT & DT)')
plt.colorbar()
plt.xticks([0,1,2], labels=['setosa', 'versicolor', 'virginica'])
plt.yticks([0,1,2], labels=['setosa', 'versicolor', 'virginica'])
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.savefig('IRIS_DT_DT.png')
plt.show()
```

```
Accuracy of Decision Tree Classifier: 97.14285714285714
```



In [23]:
```python
#SVM
X_train_svm, X_test_svm, y_train_svm, y_test_svm = train_test_split(x,y, test_size=0.3)
# create a support vector machine classifier with a linear kernel
clf = svm.SVC(kernel='linear')

# train the classifier on the training data
clf.fit(X_train_svm, y_train_svm)

# make predictions on the testing data
y_pred_svm = clf.predict(X_test_svm)

# print the accuracy score
print("Accuracy of SVM:", clf.score(X_test_svm, y_test_svm)*100)
# Calculate the confusion matrix
cm_svm = confusion_matrix(y_test_svm, y_pred_svm, labels=['setosa', 'versicolor', 'virginica'])

# Plot the confusion matrix
plt.imshow(cm_svm, cmap=plt.cm.Blues,vmin=0,vmax=35)
for i in range(cm_svm.shape[0]):
    for j in range(cm_svm.shape[1]):
        plt.text(j, i, str(cm_svm[i,j]), horizontalalignment='center', verticalalignment='center')

plt.title('IRIS: (DT & SVM)')
plt.colorbar()
plt.xticks([0,1,2], labels=['setosa', 'versicolor', 'virginica'])
plt.yticks([0,1,2], labels=['setosa', 'versicolor', 'virginica'])
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
```



In [20]:

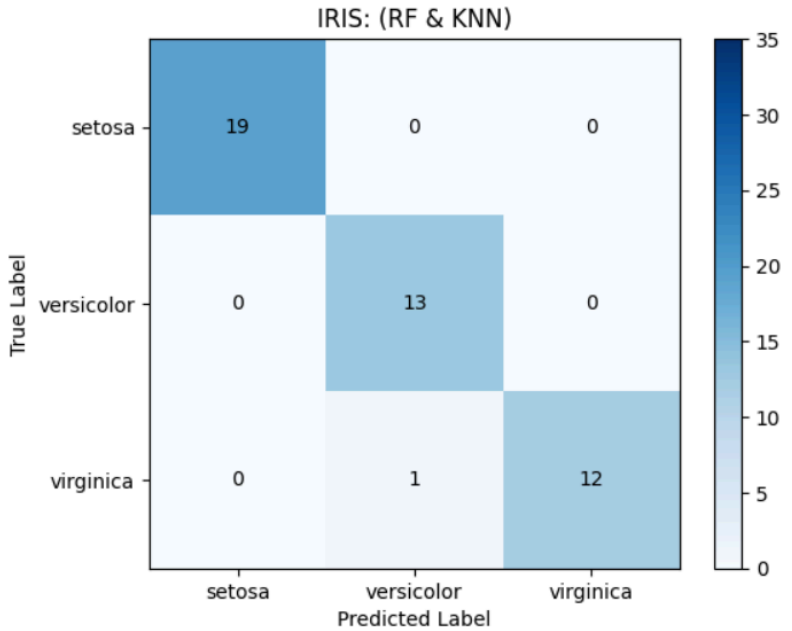
```
from sklearn.ensemble import RandomForestClassifier
from sklearn.feature_selection import SelectFromModel
import numpy as np
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
from sklearn import svm
import pandas as pd
from sklearn.metrics import confusion_matrix
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris


# Load the iris dataset
iris = load_iris()
X, y = iris.data, iris.target

# Create a Random Forest Classifier
rfc = RandomForestClassifier(n_estimators=100, random_state=42)

# Use SelectFromModel to select the most important features
sfm = SelectFromModel(rfc, threshold='median')
X_selected = sfm.fit_transform(X, y)

# Print the selected features
selected_features = np.array(iris.feature_names)[sfm.get_support()]
print("Selected Features: ", selected_features)
X_new = pd.DataFrame(X_selected, columns=selected_features)
X_new["Species"]=iris["target_names"][y]
X_new.to_csv('iris_rf.csv', index=False)
```

```
Selected Features:  ['petal length (cm)' 'petal width (cm)']
```

In [ ]:
```
#Knn
X_new = pd.read_csv("iris_rf.csv")
x = X_new.drop(X_new.columns[len(X_new.columns)-1],axis=1)
y = X_new[X_new.columns[len(X_new.columns)-1]]
#print(x,"\n",y)
# Split the dataset into training and testing sets
X_train_k, X_test_k, y_train_k, y_test_k = train_test_split(x, y, test_size=0.3, random_state=42)

# Create a KNN classifier with k=3
knn = KNeighborsClassifier(n_neighbors=2)

# Train the classifier on the training data
knn.fit(X_train_k, y_train_k)

# Predict the classes of the test set
y_pred_k = knn.predict(X_test_k)

# Calculate the accuracy of the classifier
accuracy_k = accuracy_score(y_test_k, y_pred_k)
print("Accuracy of KNN:", accuracy_k*100)
# Calculate the confusion matrix
cm_k = confusion_matrix(y_test_k, y_pred_k, labels=['setosa', 'versicolor', 'virginica'])

# Plot the confusion matrix
plt.imshow(cm_k, cmap=plt.cm.Blues,vmin=0,vmax=35)
for i in range(cm_k.shape[0]):
    for j in range(cm_k.shape[1]):
        plt.text(j, i, str(cm_k[i,j]), horizontalalignment='center', verticalalignment='center')

plt.title('IRIS: (RF & KNN)')
plt.colorbar()
plt.xticks([0,1,2], labels=['setosa', 'versicolor', 'virginica'])
plt.yticks([0,1,2], labels=['setosa', 'versicolor', 'virginica'])
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
```

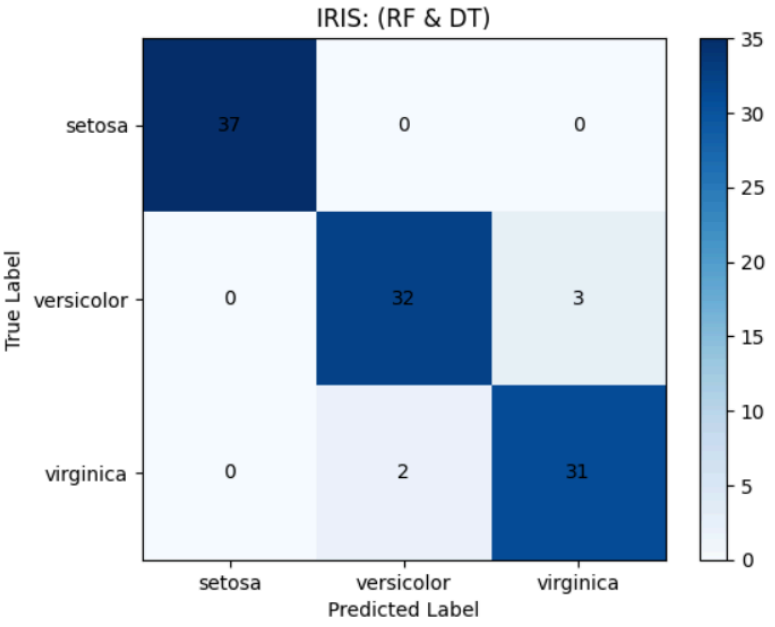
```
plt.savefig('IRIS_RF_KNN.png')
plt.show()
```

```
Accuracy of KNN: 97.77777777777777
```



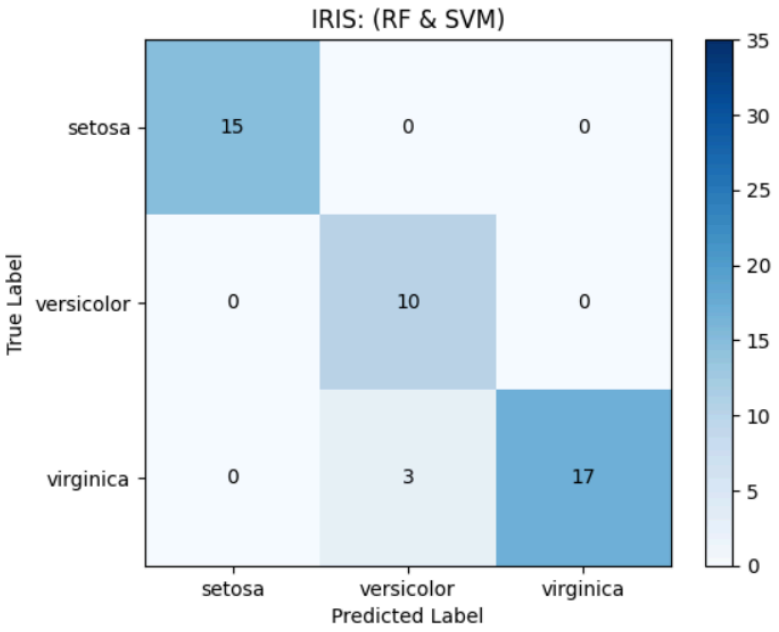
In [25]:
```
#Decision Tree
# Split the dataset into training and testing sets
X_train_dt, X_test_dt, y_train_dt, y_test_dt = train_test_split(x, y, test_size=0.7)

# Train a decision tree classifier
clf = DecisionTreeClassifier()
clf.fit(X_train_dt, y_train_dt)

# Predict the classes of the test set
y_pred_dt = clf.predict(X_test_dt)

# Evaluate the accuracy of the classifier
accuracy_dt = accuracy_score(y_test_dt, y_pred_dt)
print("Accuracy of Decision Tree Classifier:", accuracy_dt*100)
# Calculate the confusion matrix
cm_dt = confusion_matrix(y_test_dt, y_pred_dt, labels=['setosa', 'versicolor', 'virginica'])

# Plot the confusion matrix
plt.imshow(cm_dt, cmap=plt.cm.Blues,vmin=0,vmax=35)
for i in range(cm_dt.shape[0]):
    for j in range(cm_dt.shape[1]):
        plt.text(j, i, str(cm_dt[i,j]), horizontalalignment='center', verticalalignment='center')

plt.title('IRIS: (RF & DT)')
plt.colorbar()
plt.xticks([0,1,2], labels=['setosa', 'versicolor', 'virginica'])
plt.yticks([0,1,2], labels=['setosa', 'versicolor', 'virginica'])
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.savefig('IRIS_RF_DT.png')
plt.show()
```

```
Accuracy of Decision Tree Classifier: 95.23809523809523
```



In [23]:
```
#SVM
X_train_svm, X_test_svm, y_train_svm, y_test_svm = train_test_split(x,y, test_size=0.3)
# create a support vector machine classifier with a linear kernel
clf = svm.SVC(kernel='linear')

# train the classifier on the training data
clf.fit(X_train_svm, y_train_svm)

# make predictions on the testing data
y_pred_svm = clf.predict(X_test_svm)

# print the accuracy score
print("Accuracy of SVM:", clf.score(X_test_svm, y_test_svm)*100)
# Calculate the confusion matrix
cm_svm = confusion_matrix(y_test_svm, y_pred_svm, labels=['setosa', 'versicolor', 'virginica'])

# Plot the confusion matrix
plt.imshow(cm_svm, cmap=plt.cm.Blues,vmin=0,vmax=35)
for i in range(cm_svm.shape[0]):
    for j in range(cm_svm.shape[1]):
        plt.text(j, i, str(cm_svm[i,j]), horizontalalignment='center', verticalalignment='center')

plt.title('IRIS: (DT & SVM)')
plt.colorbar()
plt.xticks([0,1,2], labels=['setosa', 'versicolor', 'virginica'])
plt.yticks([0,1,2], labels=['setosa', 'versicolor', 'virginica'])
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.savefig('IRIS_DT_SVM.png')
plt.show()
```

```
Accuracy of SVM: 97.77777777777777
```



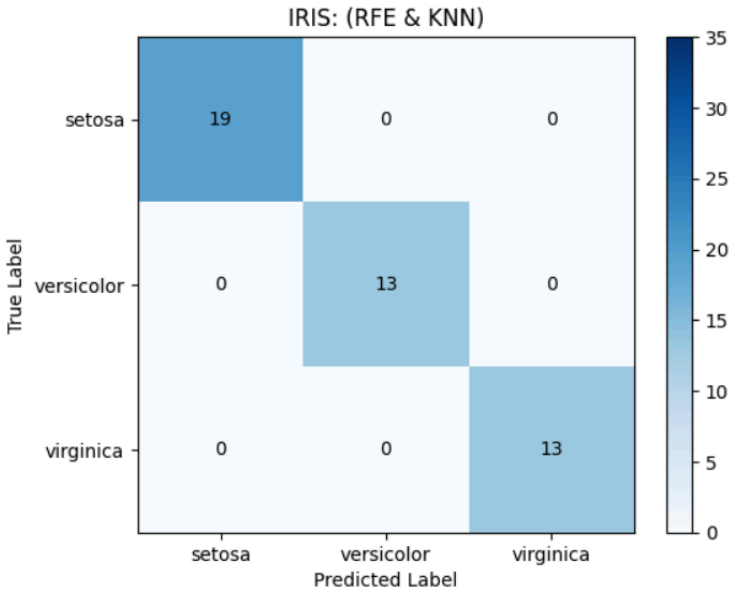
In [27]:
```
from sklearn.ensemble import RandomForestClassifier
from sklearn.feature_selection import SelectFromModel
import numpy as np
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
from sklearn import svm
import pandas as pd
from sklearn.metrics import confusion_matrix
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from sklearn.linear_model import Ridge
from sklearn.feature_selection import RFECV

# Load the iris dataset
iris = load_iris()
X, y = iris.data, iris.target

# Create a Ridge regression model
ridge = Ridge(alpha=1.0)

# Use RFECV to select the most important features
rfecv = RFECV(estimator=ridge, step=1, cv=5)
X_selected = rfecv.fit_transform(X, y)

# Print the selected features
selected_features = np.array(iris.feature_names)[rfecv.get_support()]
print("Selected Features: ", selected_features)
X_new = pd.DataFrame(X_selected, columns=selected_features)
X_new["Species"]=iris["target_names"][y]
X_new.to_csv('iris_rfe.csv', index=False)
```

```
Selected Features:  ['sepal length (cm)' 'petal length (cm)' 'petal width (cm)']
```

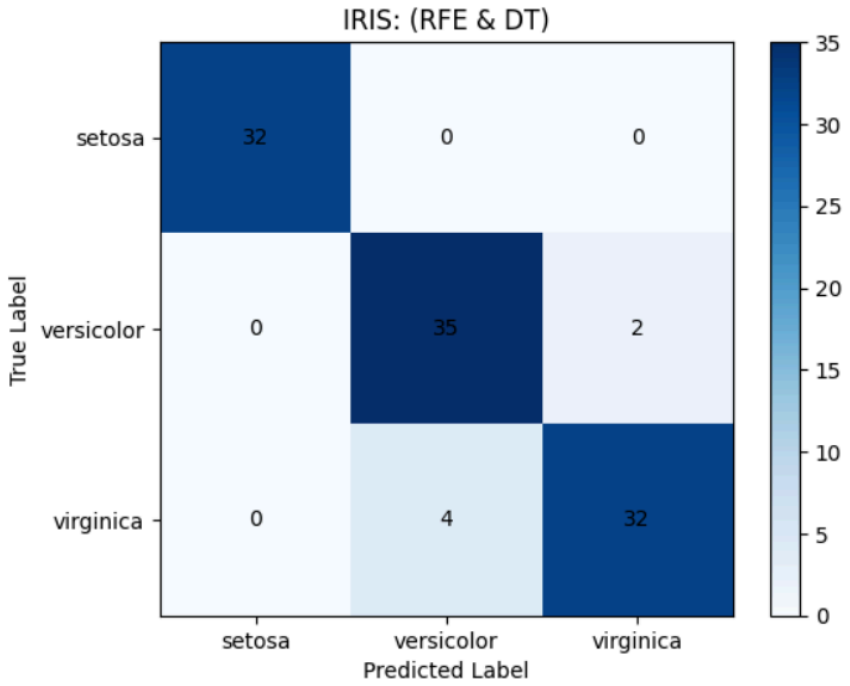
In [ ]:
```
#Knn
X_new = pd.read_csv("iris_rfe.csv")
x = X_new.drop(X_new.columns[len(X_new.columns)-1],axis=1)
y = X_new[X_new.columns[len(X_new.columns)-1]]
#print(x,"\n",y)
# Split the dataset into training and testing sets
X_train_k, X_test_k, y_train_k, y_test_k = train_test_split(x, y, test_size=0.3, random_state=42)

# Create a KNN classifier with k=3
knn = KNeighborsClassifier(n_neighbors=2)

# Train the classifier on the training data
knn.fit(X_train_k, y_train_k)

# Predict the classes of the test set
y_pred_k = knn.predict(X_test_k)

# Calculate the accuracy of the classifier
accuracy_k = accuracy_score(y_test_k, y_pred_k)
print("Accuracy of KNN:", accuracy_k*100)
# Calculate the confusion matrix
cm_k = confusion_matrix(y_test_k, y_pred_k, labels=['setosa', 'versicolor', 'virginica'])

# Plot the confusion matrix
plt.imshow(cm_k, cmap=plt.cm.Blues,vmin=0,vmax=35)
for i in range(cm_k.shape[0]):
    for j in range(cm_k.shape[1]):
        plt.text(j, i, str(cm_k[i,j]), horizontalalignment='center', verticalalignment='center')

plt.title('IRIS: (RFE & KNN)')
plt.colorbar()
plt.xticks([0,1,2], labels=['setosa', 'versicolor', 'virginica'])
plt.yticks([0,1,2], labels=['setosa', 'versicolor', 'virginica'])
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
```

```
plt.savefig('IRIS_RFE_KNN.png')
plt.show()
```

```
Accuracy of KNN: 100.0
```

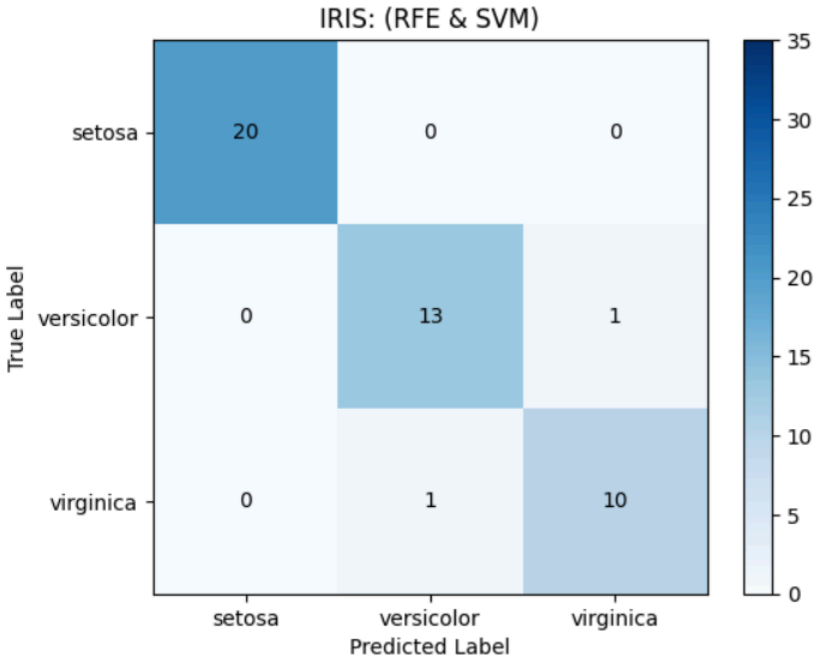


In [29]:
```
#Decision Tree
# Split the dataset into training and testing sets
X_train_dt, X_test_dt, y_train_dt, y_test_dt = train_test_split(x, y, test_size=0.7)

# Train a decision tree classifier
clf = DecisionTreeClassifier()
clf.fit(X_train_dt, y_train_dt)

# Predict the classes of the test set
y_pred_dt = clf.predict(X_test_dt)

# Evaluate the accuracy of the classifier
accuracy_dt = accuracy_score(y_test_dt, y_pred_dt)
print("Accuracy of Decision Tree Classifier:", accuracy_dt*100)
# Calculate the confusion matrix
cm_dt = confusion_matrix(y_test_dt, y_pred_dt, labels=['setosa', 'versicolor', 'virginica'])

# Plot the confusion matrix
plt.imshow(cm_dt, cmap=plt.cm.Blues,vmin=0,vmax=35)
for i in range(cm_dt.shape[0]):
    for j in range(cm_dt.shape[1]):
        plt.text(j, i, str(cm_dt[i,j]), horizontalalignment='center', verticalalignment='center')

plt.title('IRIS: (RFE & DT)')
plt.colorbar()
plt.xticks([0,1,2], labels=['setosa', 'versicolor', 'virginica'])
plt.yticks([0,1,2], labels=['setosa', 'versicolor', 'virginica'])
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.savefig('IRIS_RFE_DT.png')
plt.show()
```



In [30]:
```
#SVM
X_train_svm, X_test_svm, y_train_svm, y_test_svm = train_test_split(x,y, test_size=0.3)
# create a support vector machine classifier with a linear kernel
clf = svm.SVC(kernel='linear')

# train the classifier on the training data
clf.fit(X_train_svm, y_train_svm)

# make predictions on the testing data
y_pred_svm = clf.predict(X_test_svm)

# print the accuracy score
print("Accuracy of SVM:", clf.score(X_test_svm, y_test_svm)*100)
# Calculate the confusion matrix
cm_svm = confusion_matrix(y_test_svm, y_pred_svm, labels=['setosa', 'versicolor', 'virginica'])

# Plot the confusion matrix
plt.imshow(cm_svm, cmap=plt.cm.Blues,vmin=0,vmax=35)
for i in range(cm_svm.shape[0]):
    for j in range(cm_svm.shape[1]):
        plt.text(j, i, str(cm_svm[i,j]), horizontalalignment='center', verticalalignment='center')

plt.title('IRIS: (RFE & SVM)')
plt.colorbar()
plt.xticks([0,1,2], labels=['setosa', 'versicolor', 'virginica'])
plt.yticks([0,1,2], labels=['setosa', 'versicolor', 'virginica'])
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.savefig('IRIS_RFE_SVM.png')
plt.show()
```

```
Accuracy of SVM: 95.55555555555556
```



6.2 DIABETES DATASET

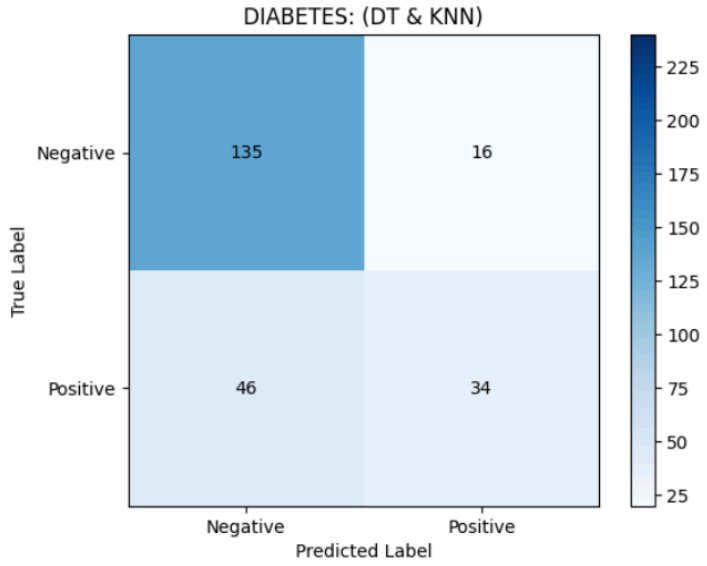
In [2]:

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.feature_selection import SelectFromModel
import numpy as np
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
from sklearn import svm
import pandas as pd
from sklearn.metrics import confusion_matrix
import matplotlib.pyplot as plt

df = pd.read_csv('diabetes.csv')

# Split the features and target variable
X = df.drop('Outcome', axis=1)
y = df['Outcome']

# Create a Decision Tree Classifier
dtc = DecisionTreeClassifier(random_state=42)

# Use SelectFromModel to select the most important features
sfm = SelectFromModel(dtc, threshold='median')
X_selected = sfm.fit_transform(X, y)

# Print the selected features
selected_features = np.array(X.columns)[sfm.get_support()]
print("Selected Features: ", selected_features)
X_new = pd.DataFrame(X_selected, columns=selected_features)
X_new["Outcome"] = y
X_new.to_csv('Diab_DT.csv', index=False)
#--------------------------------------------------------------------------------#
```

```
#Knn
X_new = pd.read_csv("Diab_DT.csv")
x = X_new.drop(X_new.columns[len(X_new.columns)-1],axis=1)
y = X_new[X_new.columns[len(X_new.columns)-1]]
#print(x,"\n",y)
# Split the dataset into training and testing sets
X_train_k, X_test_k, y_train_k, y_test_k = train_test_split(x, y, test_size=0.3, random_state=42)

# Create a KNN classifier with k=3
knn = KNeighborsClassifier(n_neighbors=2)

# Train the classifier on the training data
knn.fit(X_train_k, y_train_k)

# Predict the classes of the test set
y_pred_k = knn.predict(X_test_k)

# Calculate the accuracy of the classifier
accuracy_k = accuracy_score(y_test_k, y_pred_k)
print("Accuracy of KNN:", accuracy_k*100)
# Calculate the confusion matrix
cm_k = confusion_matrix(y_test_k, y_pred_k)

# Plot the confusion matrix
plt.imshow(cm_k, cmap=plt.cm.Blues,vmin=20,vmax=240)
for i in range(cm_k.shape[0]):
    for j in range(cm_k.shape[1]):
        plt.text(j, i, str(cm_k[i,j]), horizontalalignment='center', verticalalignment='center')
plt.title('DIABETES: (DT & KNN)')
plt.colorbar()
plt.xticks([0,1], labels=['Negative','Positive'])
plt.yticks([0,1], labels=['Negative','Positive'])
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.savefig('DIAB_DT_KNN.png')
plt.show()
```



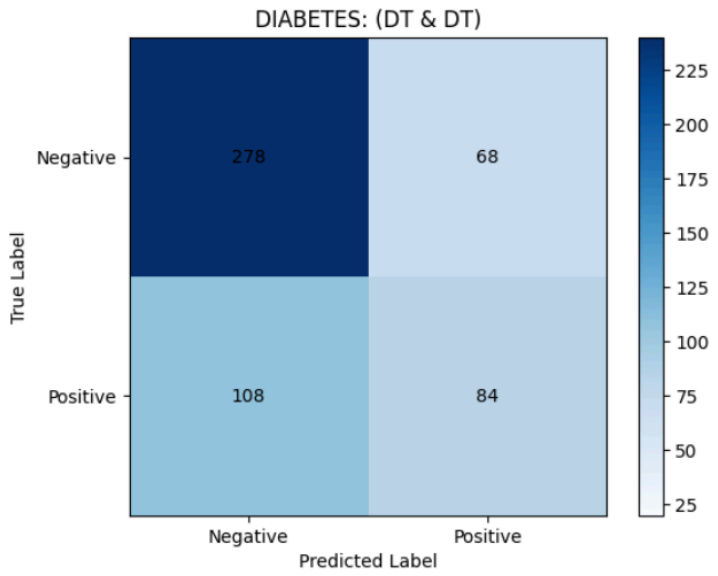
```
Selected Features:  ['Glucose' 'BMI' 'DiabetesPedigreeFunction' 'Age']
Accuracy of KNN: 73.16017316017316
```



```python
#Decision Tree
# Split the dataset into training and testing sets
X_train_dt, X_test_dt, y_train_dt, y_test_dt = train_test_split(x, y, test_size=0.7)

# Train a decision tree classifier
clf = DecisionTreeClassifier()
clf.fit(X_train_dt, y_train_dt)

# Predict the classes of the test set
y_pred_dt = clf.predict(X_test_dt)

# Evaluate the accuracy of the classifier
accuracy_dt = accuracy_score(y_test_dt, y_pred_dt)
print("Accuracy of Decision Tree Classifier:", accuracy_dt*100)
# Calculate the confusion matrix
cm_dt = confusion_matrix(y_test_dt, y_pred_dt)

# Plot the confusion matrix
plt.imshow(cm_dt, cmap=plt.cm.Blues,vmin=20,vmax=240)
for i in range(cm_dt.shape[0]):
    for j in range(cm_dt.shape[1]):
        plt.text(j, i, str(cm_dt[i,j]), horizontalalignment='center', verticalalignment='center')
plt.title('DIABETES: (DT & DT)')
plt.colorbar()
plt.xticks([0,1], labels=['Negative','Positive'])
plt.yticks([0,1], labels=['Negative','Positive'])
plt.xlabel('Predicted Label')
plt.savefig('DIAB_DT_DT.png')
plt.ylabel('True Label')
plt.show()
```

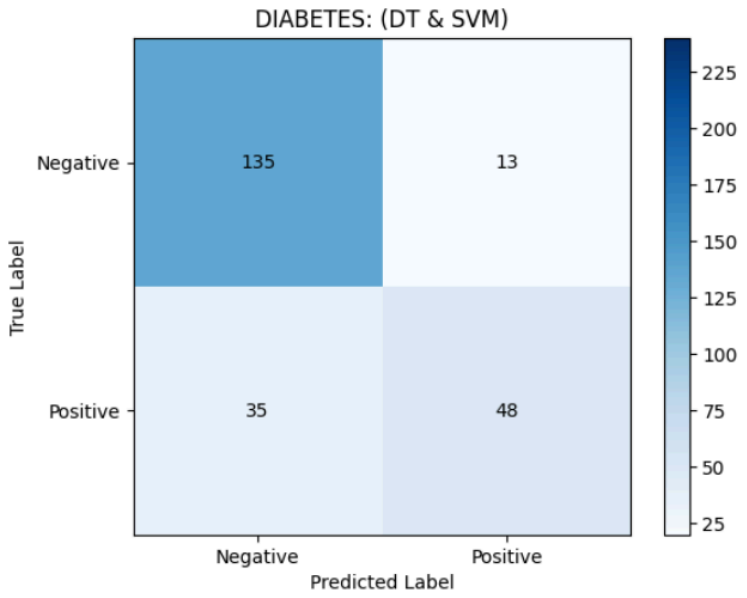


```
Accuracy of Decision Tree Classifier: 67.28624535315984
```



```python
#SVM
X_train_svm, X_test_svm, y_train_svm, y_test_svm = train_test_split(x,y, test_size=0.3)
# create a support vector machine classifier with a linear kernel
clf = svm.SVC(kernel='linear')

# train the classifier on the training data
clf.fit(X_train_svm, y_train_svm)

# make predictions on the testing data
y_pred_svm = clf.predict(X_test_svm)

# print the accuracy score
print("Accuracy of SVM:", clf.score(X_test_svm, y_test_svm)*100)
# Calculate the confusion matrix
cm_svm = confusion_matrix(y_test_svm, y_pred_svm)

# Plot the confusion matrix
plt.imshow(cm_svm, cmap=plt.cm.Blues,vmin=20,vmax=240)
for i in range(cm_svm.shape[0]):
    for j in range(cm_svm.shape[1]):
        plt.text(j, i, str(cm_svm[i,j]), horizontalalignment='center', verticalalignment='center')
plt.title('DIABETES: (DT & SVM)')
plt.colorbar()
plt.xticks([0,1], labels=['Negative','Positive'])
plt.yticks([0,1], labels=['Negative','Positive'])
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.savefig('DIAB_DT_SVM.png')
plt.show()
```

```
Accuracy of SVM: 79.22077922077922
```

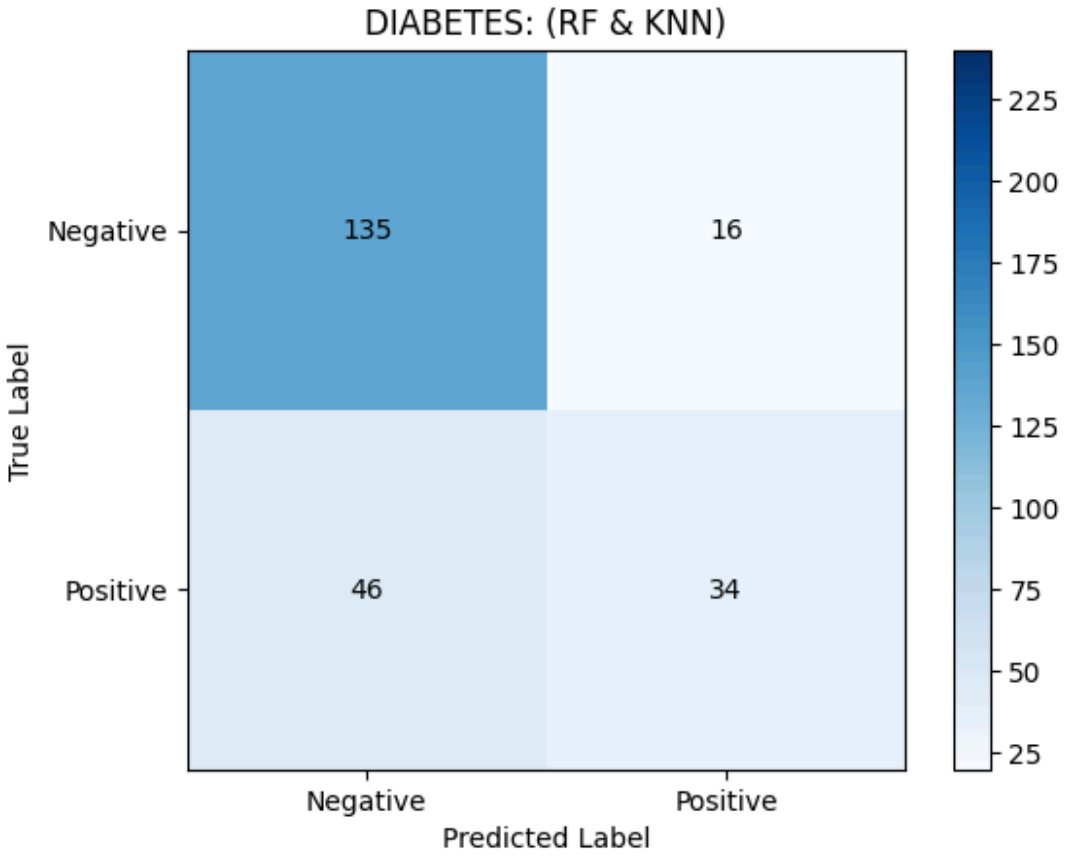


```
from sklearn.ensemble import RandomForestClassifier
from sklearn.feature_selection import SelectFromModel
import numpy as np
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
from sklearn import svm
import pandas as pd
from sklearn.metrics import confusion_matrix
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix
import matplotlib.pyplot as plt

df = pd.read_csv('diabetes.csv')

# Split the features and target variable
X = df.drop('Outcome', axis=1)
y = df['Outcome']

# Create a Random Forest Classifier
rfc = RandomForestClassifier(n_estimators=100, random_state=42)

# Use SelectFromModel to select the most important features
sfm = SelectFromModel(rfc, threshold='median')
X_selected = sfm.fit_transform(X, y)
print(type(sfm))
# Print the selected features
selected_features = np.array(X.columns)[sfm.get_support()]
print("Selected Features: ", selected_features)
X_new = pd.DataFrame(X_selected, columns=selected_features)
X_new["Outcome"] = y
X_new.to_csv('Diab_RandFor.csv', index=False)
```

```
<class 'sklearn.feature_selection._from_model.SelectFromModel'>
Selected Features:  ['Glucose' 'BMI' 'DiabetesPedigreeFunction' 'Age']
```

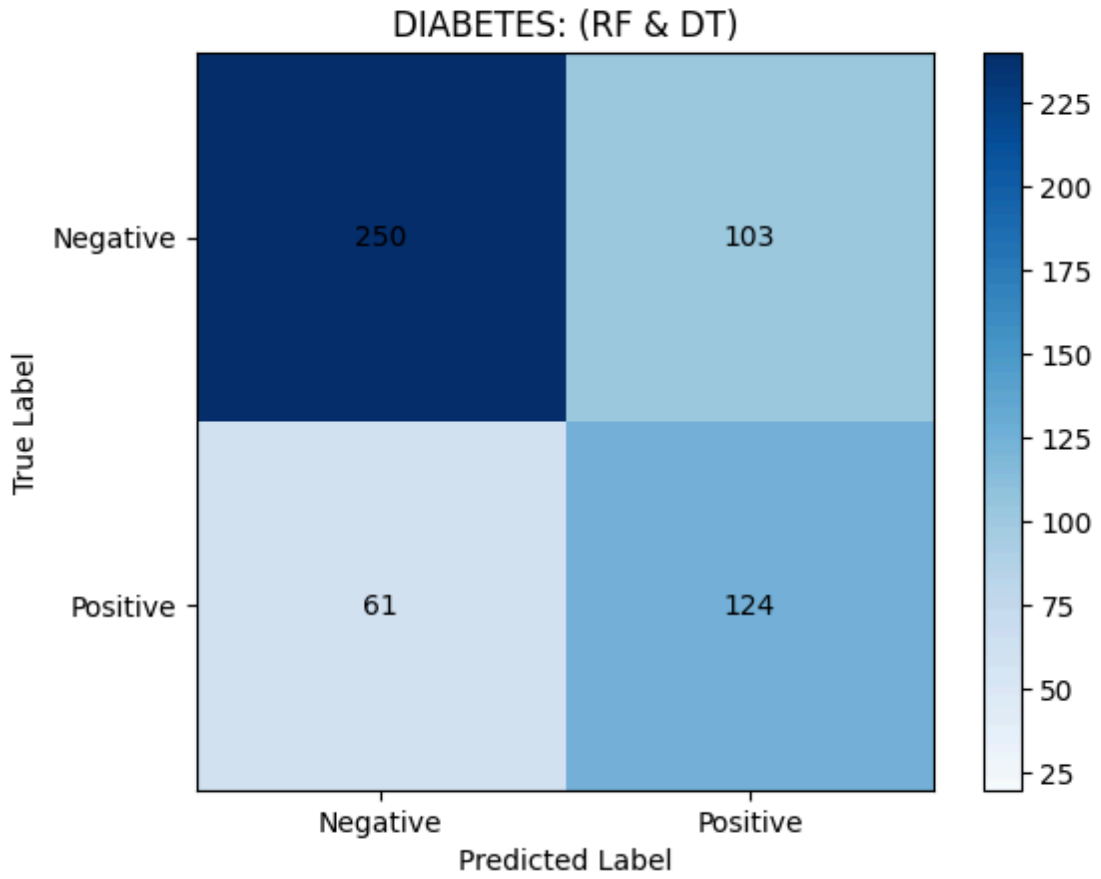
```
#Knn
X_new = pd.read_csv("Diab_RandFor.csv")
x = X_new.drop(X_new.columns[len(X_new.columns)-1],axis=1)
y = X_new[X_new.columns[len(X_new.columns)-1]]
#print(x,"\n",y)
# Split the dataset into training and testing sets
X_train_k, X_test_k, y_train_k, y_test_k = train_test_split(x, y, test_size=0.3, random_state=42)

# Create a KNN classifier with k=3
knn = KNeighborsClassifier(n_neighbors=2)

# Train the classifier on the training data
knn.fit(X_train_k, y_train_k)

# Predict the classes of the test set
y_pred_k = knn.predict(X_test_k)

# Calculate the accuracy of the classifier
accuracy_k = accuracy_score(y_test_k, y_pred_k)
print("Accuracy of KNN:", accuracy_k*100)
#Calculate the confusion matrix
cm_k = confusion_matrix(y_test_k, y_pred_k)

# Plot the confusion matrix
plt.imshow(cm_k, cmap=plt.cm.Blues,vmin=20,vmax=240)
for i in range(cm_k.shape[0]):
    for j in range(cm_k.shape[1]):
        plt.text(j, i, str(cm_k[i,j]), horizontalalignment='center', verticalalignment='center')

plt.title('DIABETES: (RF & KNN)')
plt.colorbar()
plt.xticks([0,1], labels=['Negative','Positive'])
plt.yticks([0,1], labels=['Negative','Positive'])
plt.xlabel('Predicted Label')
plt.savefig('DIAB_RF_KNN.png')
plt.ylabel('True Label')
plt.show()
```

```
Accuracy of KNN: 73.16017316017316
```



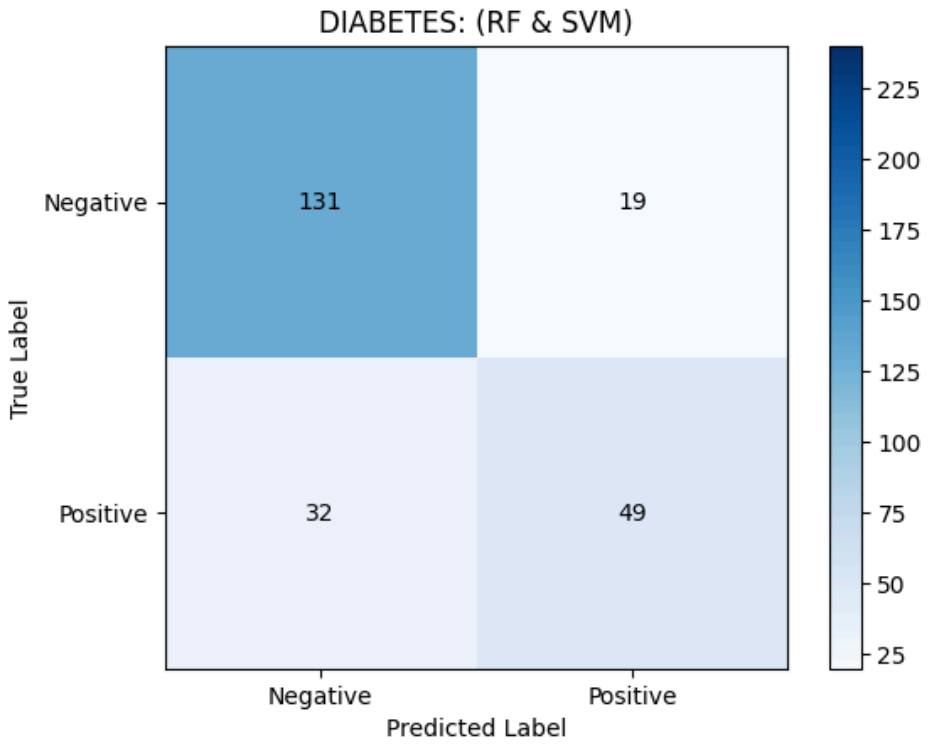

```
#Decision Tree
# Split the dataset into training and testing sets
X_train_dt, X_test_dt, y_train_dt, y_test_dt = train_test_split(x, y, test_size=0.7)

# Train a decision tree classifier
clf = DecisionTreeClassifier()
clf.fit(X_train_dt, y_train_dt)

# Predict the classes of the test set
y_pred_dt = clf.predict(X_test_dt)

# Evaluate the accuracy of the classifier
accuracy_dt = accuracy_score(y_test_dt, y_pred_dt)
print("Accuracy of Decision Tree Classifier:", accuracy_dt*100)
# Calculate the confusion matrix
cm_dt = confusion_matrix(y_test_dt, y_pred_dt)

# Plot the confusion matrix
plt.imshow(cm_dt, cmap=plt.cm.Blues,vmin=20,vmax=240)
for i in range(cm_dt.shape[0]):
    for j in range(cm_dt.shape[1]):
        plt.text(j, i, str(cm_dt[i,j]), horizontalalignment='center', verticalalignment='center')

plt.title('DIABETES: (RF & DT)')
plt.colorbar()
plt.xticks([0,1], labels=['Negative','Positive'])
plt.yticks([0,1], labels=['Negative','Positive'])
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.savefig('DIAB_RF_DT.png')
plt.show()
```

```
Accuracy of Decision Tree Classifier: 69.51672862453532
```




```python
#SVM
X_train_svm, X_test_svm, y_train_svm, y_test_svm = train_test_split(x,y, test_size=0.3)
# create a support vector machine classifier with a linear kernel
clf = svm.SVC(kernel='linear')

# train the classifier on the training data
clf.fit(X_train_svm, y_train_svm)

# make predictions on the testing data
y_pred_svm = clf.predict(X_test_svm)

# print the accuracy score
print("Accuracy of SVM:", clf.score(X_test_svm, y_test_svm)*100)
# Calculate the confusion matrix
cm_svm = confusion_matrix(y_test_svm, y_pred_svm)

# Plot the confusion matrix
plt.imshow(cm_svm, cmap=plt.cm.Blues,vmin=20,vmax=240)
for i in range(cm_svm.shape[0]):
    for j in range(cm_svm.shape[1]):
        plt.text(j, i, str(cm_svm[i,j]), horizontalalignment='center', verticalalignment='center')

plt.title('DIABETES: (RF & SVM)')
plt.colorbar()
plt.xticks([0,1], labels=['Negative','Positive'])
plt.yticks([0,1], labels=['Negative','Positive'])
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.savefig('DIAB_RF_SVM.png')
plt.show()
```

```
Accuracy of SVM: 77.92207792207793
```

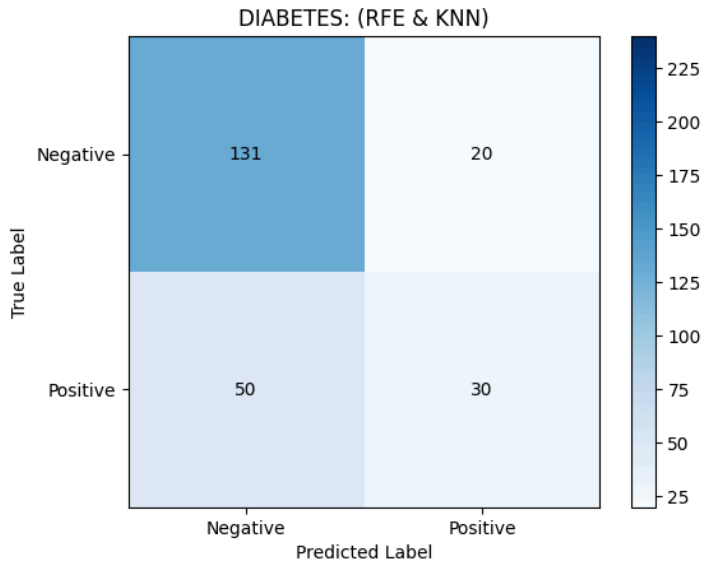



```python
from sklearn.linear_model import Ridge
from sklearn.feature_selection import RFECV
from sklearn.feature_selection import SelectFromModel
import numpy as np
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
from sklearn import svm
import pandas as pd
from sklearn.metrics import confusion_matrix
import matplotlib.pyplot as plt

df = pd.read_csv('diabetes.csv')

# Split the features and target variable
X = df.drop('Outcome', axis=1)
y = df['Outcome']

# Create a Ridge regression model
ridge = Ridge(alpha=1.0)

# Use RFECV to select the most important features
rfecv = RFECV(estimator=ridge, step=1, cv=5)
X_selected = rfecv.fit_transform(X, y)

# Print the selected features
selected_features = np.array(X.columns)[rfecv.support_]
print("Selected Features: ", selected_features)

# Create a new dataframe with the selected features
X_new = pd.DataFrame(X_selected, columns=selected_features)
X_new["Outcome"] = y

# Save the new dataframe to a CSV file
X_new.to_csv('Diab_Ridge_RFE.csv', index=False)
```

```
Selected Features:  ['Pregnancies' 'BloodPressure' 'Glucose' 'BMI' 'DiabetesPedigreeFunction'
 'Age']
```

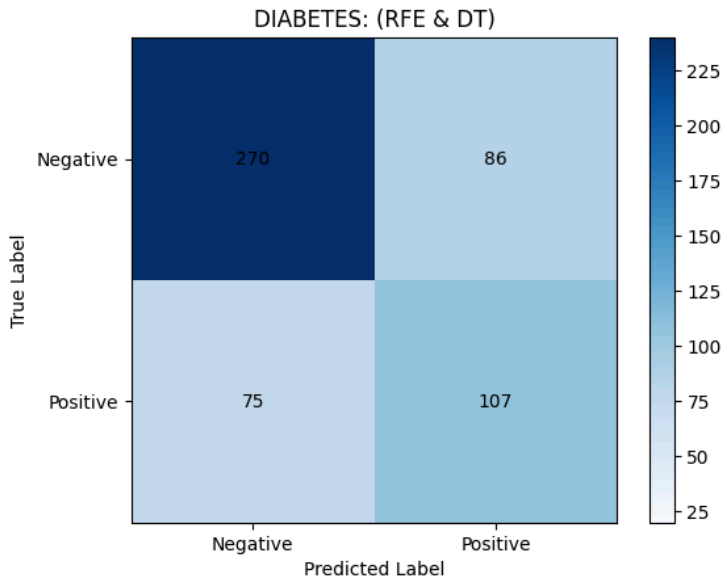
```
#Knn
X_new = pd.read_csv("Diab_Ridge_RFE.csv")
x = X_new.drop(X_new.columns[len(X_new.columns)-1],axis=1)
y = X_new[X_new.columns[len(X_new.columns)-1]]
#print(x,"\n",y)
# Split the dataset into training and testing sets
X_train_k, X_test_k, y_train_k, y_test_k = train_test_split(x, y, test_size=0.3, random_state=42)

# Create a KNN classifier with k=3
knn = KNeighborsClassifier(n_neighbors=2)

# Train the classifier on the training data
knn.fit(X_train_k, y_train_k)

# Predict the classes of the test set
y_pred_k = knn.predict(X_test_k)

# Calculate the accuracy of the classifier
accuracy_k = accuracy_score(y_test_k, y_pred_k)
print("Accuracy of KNN:", accuracy_k*100)
# Calculate the confusion matrix
cm_k = confusion_matrix(y_test_k, y_pred_k)

# Plot the confusion matrix
plt.imshow(cm_k, cmap=plt.cm.Blues,vmin=20,vmax=240)
for i in range(cm_k.shape[0]):
    for j in range(cm_k.shape[1]):
        plt.text(j, i, str(cm_k[i,j]), horizontalalignment='center', verticalalignment='center')
plt.title('DIABETES: (RFE & KNN)')
plt.colorbar()
plt.xticks([0,1], labels=['Negative','Positive'])
plt.yticks([0,1], labels=['Negative','Positive'])
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.savefig('DIAB_RFE_KNN.png')
plt.show()
```

```
Accuracy of KNN: 69.6969696969697
```



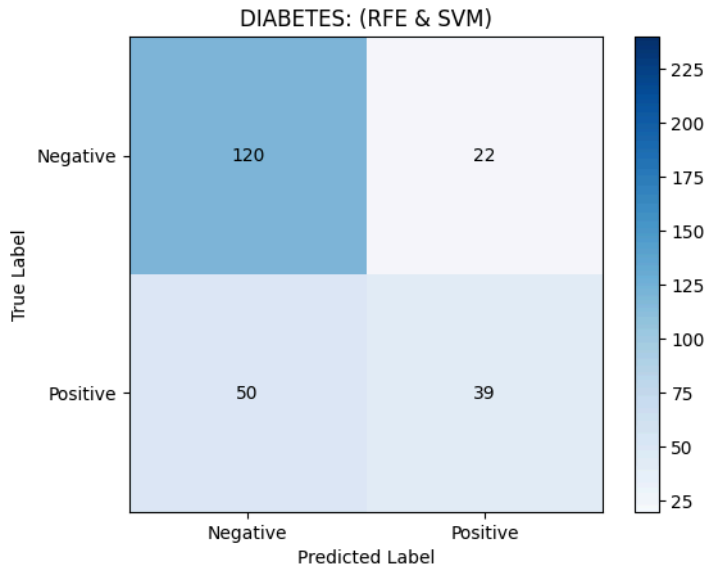
```
#Decision Tree
# Split the dataset into training and testing sets
X_train_dt, X_test_dt, y_train_dt, y_test_dt = train_test_split(x, y, test_size=0.7)

# Train a decision tree classifier
clf = DecisionTreeClassifier()
clf.fit(X_train_dt, y_train_dt)

# Predict the classes of the test set
y_pred_dt = clf.predict(X_test_dt)

# Evaluate the accuracy of the classifier
accuracy_dt = accuracy_score(y_test_dt, y_pred_dt)
print("Accuracy of Decision Tree Classifier:", accuracy_dt*100)
# Calculate the confusion matrix
cm_dt = confusion_matrix(y_test_dt, y_pred_dt)

# Plot the confusion matrix
plt.imshow(cm_dt, cmap=plt.cm.Blues,vmin=20,vmax=240)
for i in range(cm_dt.shape[0]):
    for j in range(cm_dt.shape[1]):
        plt.text(j, i, str(cm_dt[i,j]), horizontalalignment='center', verticalalignment='center')
plt.title('DIABETES: (RFE & DT)')
plt.colorbar()
plt.xticks([0,1], labels=['Negative','Positive'])
plt.yticks([0,1], labels=['Negative','Positive'])
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.savefig('DIAB_RFE_DT.png')
plt.show()
```

```
Accuracy of Decision Tree Classifier: 70.07434944237917
```



```
#SVM
X_train_svm, X_test_svm, y_train_svm, y_test_svm = train_test_split(x,y, test_size=0.3)
# create a support vector machine classifier with a linear kernel
clf = svm.SVC(kernel='linear')

# train the classifier on the training data
clf.fit(X_train_svm, y_train_svm)

# make predictions on the testing data
y_pred_svm = clf.predict(X_test_svm)

# print the accuracy score
print("Accuracy of SVM:", clf.score(X_test_svm, y_test_svm)*100)
# Calculate the confusion matrix
cm_svm = confusion_matrix(y_test_svm, y_pred_svm)

# Plot the confusion matrix
plt.imshow(cm_svm, cmap=plt.cm.Blues,vmin=20,vmax=240)
for i in range(cm_svm.shape[0]):
    for j in range(cm_svm.shape[1]):
        plt.text(j, i, str(cm_svm[i,j]), horizontalalignment='center', verticalalignment='center')
plt.title('DIABETES: (RFE & SVM)')
plt.colorbar()
plt.xticks([0,1], labels=['Negative','Positive'])
plt.yticks([0,1], labels=['Negative','Positive'])
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.savefig('DIAB_RFE_SVM.png')
plt.show()
```

```
Accuracy of SVM: 68.83116883116884
```



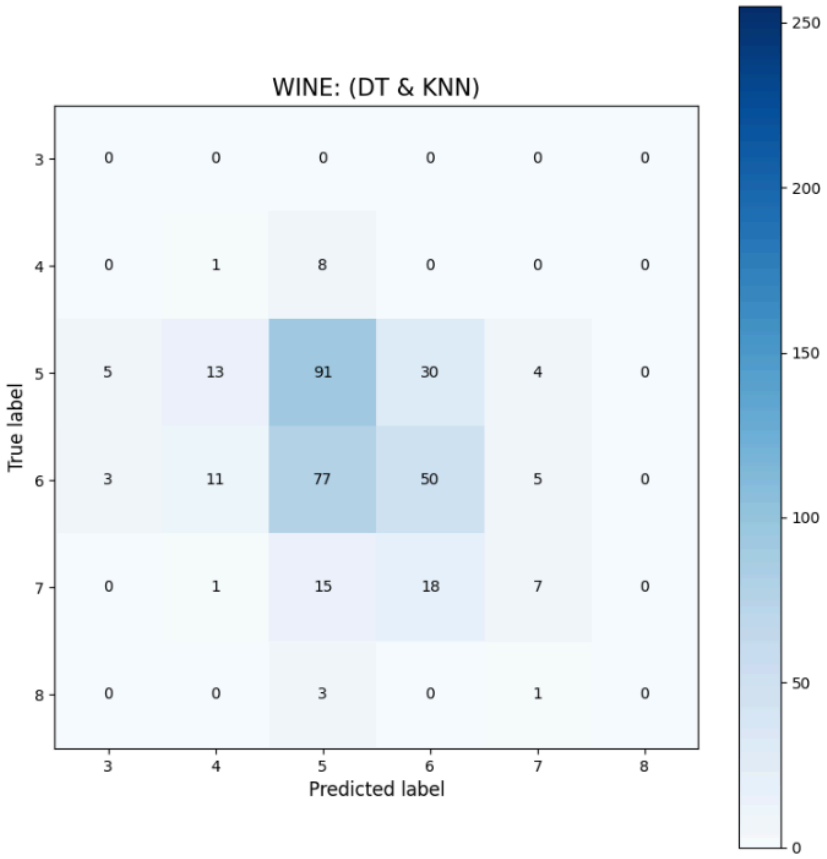
6.3 WINE QUALITY DATASET

```
from sklearn.feature_selection import SelectFromModel
import numpy as np
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
from sklearn import svm
import pandas as pd
from sklearn.metrics import confusion_matrix
import matplotlib.pyplot as plt

df = pd.read_csv('WineQT.csv')

# Split the features and target variable
X = df.drop('quality', axis=1)
y = df['quality']

# Create a Decision Tree Classifier
dtc = DecisionTreeClassifier(random_state=42)

# Use SelectFromModel to select the most important features
sfm = SelectFromModel(dtc, threshold='median')
X_selected = sfm.fit_transform(X,y)
# Print the selected features
selected_features = np.array(X.columns)[sfm.get_support()]
print("Selected Features: ", selected_features)
X_new = pd.DataFrame(X_selected, columns=selected_features)
X_new["quality"] = y
X_new.to_csv('Wine_DT.csv', index=False)
```

```
Selected Features:  ['citric acid' 'total sulfur dioxide' 'density' 'sulphates' 'alcohol' 'Id']
```

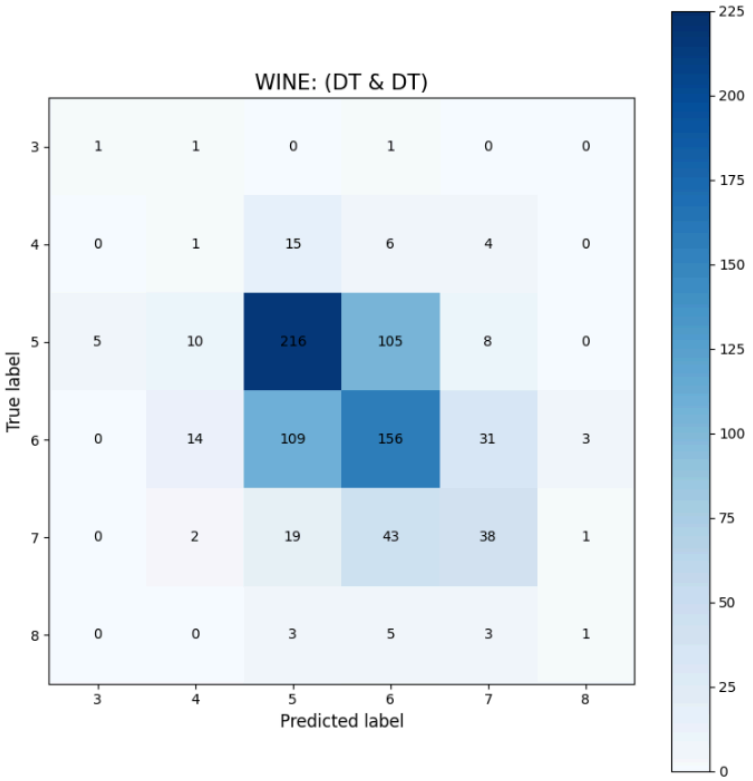
```
#Knn
X_new = pd.read_csv("Wine_DT.csv")
x = X_new.drop(X_new.columns[len(X_new.columns)-1],axis=1)
y = X_new[X_new.columns[len(X_new.columns)-1]]
#print(x,"\n",y)
# Split the dataset into training and testing sets
X_train_k, X_test_k, y_train_k, y_test_k = train_test_split(x, y, test_size=0.3, random_state=42)

# Create a KNN classifier with k=3
knn = KNeighborsClassifier(n_neighbors=2)

# Train the classifier on the training data
knn.fit(X_train_k, y_train_k)

# Predict the classes of the test set
y_pred_k = knn.predict(X_test_k)

# Calculate the accuracy of the classifier
accuracy_k = accuracy_score(y_test_k, y_pred_k)
print("Accuracy of KNN:", accuracy_k*100)
# Create the confusion matrix
cm_k = confusion_matrix(y_test_k, y_pred_k)

# Visualize the confusion matrix
plt.figure(figsize=(8,8))
plt.imshow(cm_k, interpolation='nearest', cmap='Blues',vmin=0,vmax=255)
for i in range(cm_k.shape[0]):
    for j in range(cm_k.shape[1]):
        plt.text(j, i, str(cm_k[i,j]), horizontalalignment='center', verticalalignment='center')
plt.title('WINE: (DT & KNN)', size=15)
plt.colorbar()
tick_marks = np.arange(len(set(y)))
plt.xticks(tick_marks, sorted(set(y)), size=10)
plt.yticks(tick_marks, sorted(set(y)), size=10)
plt.xlabel('Predicted label', size=12)
plt.ylabel('True label', size=12)
plt.tight_layout()
plt.savefig('WINE_DT_KNN.png')
plt.show()
```

```
Accuracy of KNN: 43.440233236151606
```



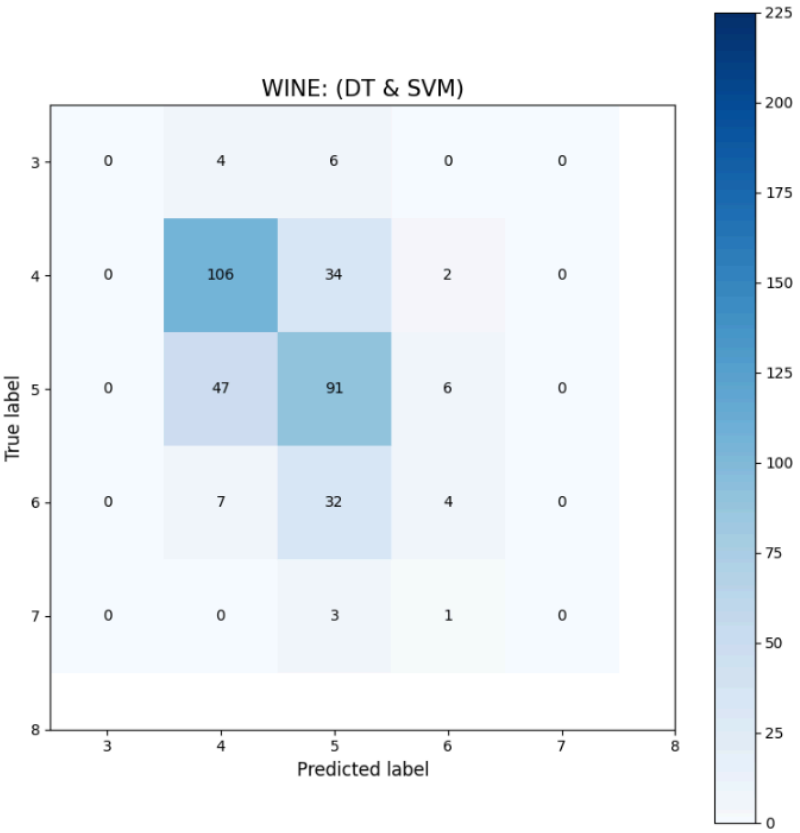

```
#Decision Tree
# Split the dataset into training and testing sets
X_train_dt, X_test_dt, y_train_dt, y_test_dt = train_test_split(x, y, test_size=0.7)

# Train a decision tree classifier
clf = DecisionTreeClassifier()
clf.fit(X_train_dt, y_train_dt)

# Predict the classes of the test set
y_pred_dt = clf.predict(X_test_dt)

# Evaluate the accuracy of the classifier
accuracy_dt = accuracy_score(y_test_dt, y_pred_dt)
print("Accuracy of Decision Tree Classifier:", accuracy_dt*100)
# Create the confusion matrix
cm_dt = confusion_matrix(y_test_dt, y_pred_dt)

# Visualize the confusion matrix
plt.figure(figsize=(8,8))
plt.imshow(cm_dt, interpolation='nearest', cmap='Blues',vmin=0,vmax=225)
for i in range(cm_dt.shape[0]):
    for j in range(cm_dt.shape[1]):
        plt.text(j, i, str(cm_dt[i,j]), horizontalalignment='center', verticalalignment='center')
plt.title('WINE: (DT & DT)', size=15)
plt.colorbar()
tick_marks = np.arange(len(set(y)))
plt.xticks(tick_marks, sorted(set(y)), size=10)
plt.yticks(tick_marks, sorted(set(y)), size=10)
plt.xlabel('Predicted label', size=12)
plt.ylabel('True label', size=12)
plt.tight_layout()
plt.savefig('WINE_DT_DT.png')
plt.show()
```

```
Accuracy of Decision Tree Classifier: 51.68539325842697
```




```
#SVM
X_train_svm, X_test_svm, y_train_svm, y_test_svm = train_test_split(x,y, test_size=0.3)
# create a support vector machine classifier with a linear kernel
clf = svm.SVC(kernel='linear')

# train the classifier on the training data
clf.fit(X_train_svm, y_train_svm)

# make predictions on the testing data
y_pred_svm = clf.predict(X_test_svm)

# print the accuracy score
print("Accuracy of SVM:", clf.score(X_test_svm, y_test_svm)*100)
# Create the confusion matrix
cm_svm = confusion_matrix(y_test_svm, y_pred_svm)

# Visualize the confusion matrix
plt.figure(figsize=(8,8))
plt.imshow(cm_svm, interpolation='nearest', cmap='Blues',vmin=0,vmax=225)
for i in range(cm_svm.shape[0]):
    for j in range(cm_svm.shape[1]):
        plt.text(j, i, str(cm_svm[i,j]), horizontalalignment='center', verticalalignment='center')
plt.title('WINE: (DT & SVM)', size=15)
plt.colorbar()
tick_marks = np.arange(len(set(y)))
plt.xticks(tick_marks, sorted(set(y)), size=10)
plt.yticks(tick_marks, sorted(set(y)), size=10)
plt.xlabel('Predicted label', size=12)
plt.ylabel('True label', size=12)
plt.tight_layout()
plt.savefig('WINE_DT_SVM.png')
plt.show()
```

```
Accuracy of SVM: 56.268221574344025
```

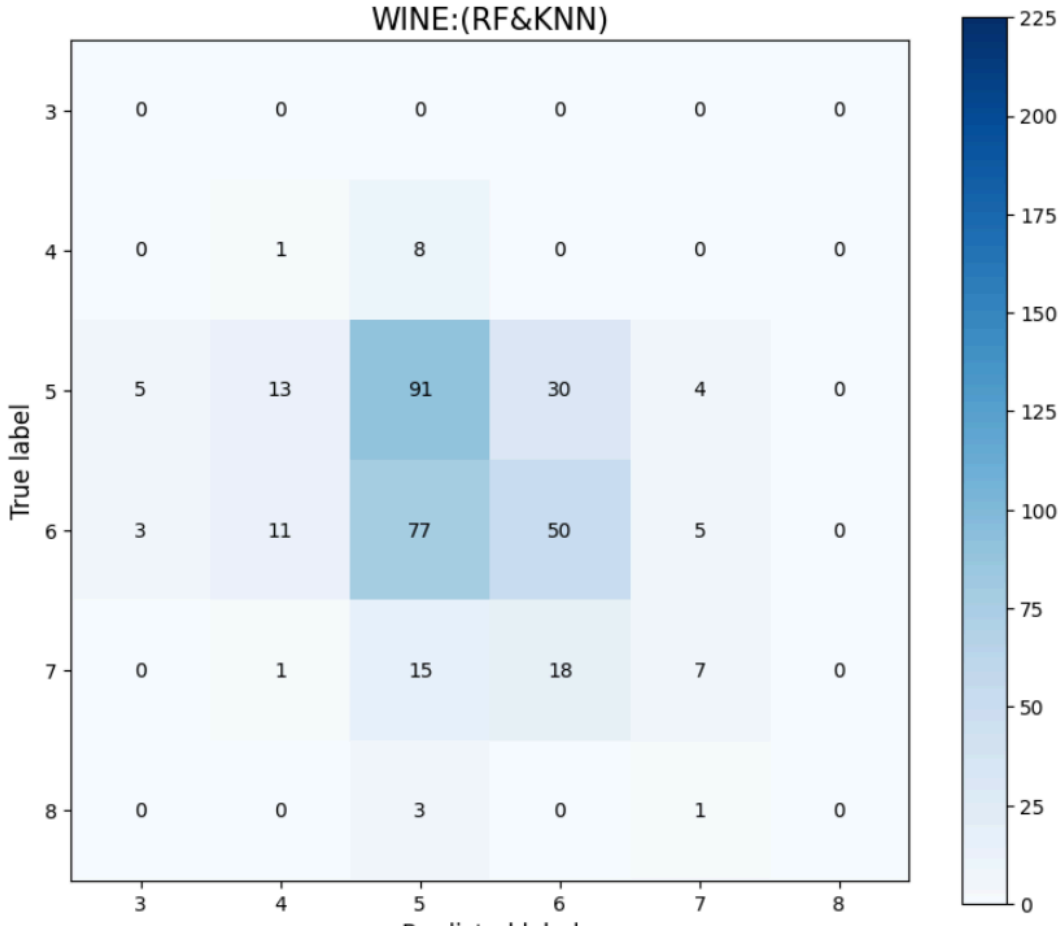



```
from sklearn.ensemble import RandomForestClassifier
from sklearn.feature_selection import SelectFromModel
import numpy as np
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
from sklearn import svm
import pandas as pd
from sklearn.metrics import confusion_matrix
import matplotlib.pyplot as plt


df = pd.read_csv('WineQT.csv')

# Split the features and target variable
X = df.drop('quality', axis=1)
y = df['quality']

# Create a Random Forest Classifier
rfc = RandomForestClassifier(n_estimators=100, random_state=42)

# Use SelectFromModel to select the most important features
sfm = SelectFromModel(rfc, threshold='median')
X_selected = sfm.fit_transform(X, y)
print(type(sfm))
# Print the selected features
selected_features = np.array(X.columns)[sfm.get_support()]
print("Selected Features: ", selected_features)
X_new = pd.DataFrame(X_selected, columns=selected_features)
X_new["quality"] = y
X_new.to_csv('Wine_RandFor.csv', index=False)
```

```
<class 'sklearn.feature_selection._from_model.SelectFromModel'>
Selected Features:  ['volatile acidity' 'total sulfur dioxide' 'density' 'sulphates' 'alcohol'
```

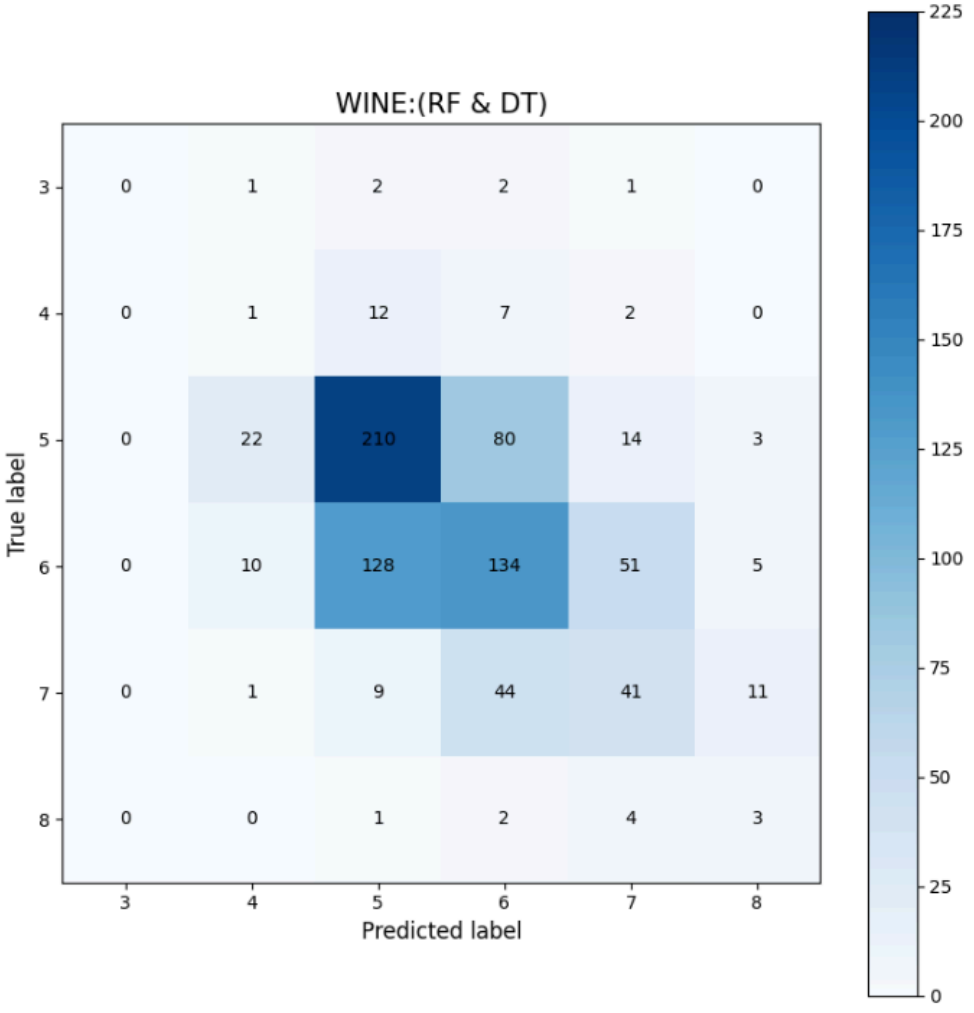
```
#Knn
X_new = pd.read_csv("Wine_RandFor.csv")
x = X_new.drop(X_new.columns[len(X_new.columns)-1],axis=1)
y = X_new[X_new.columns[len(X_new.columns)-1]]
#print(x,"\n",y)
# Split the dataset into training and testing sets
X_train_k, X_test_k, y_train_k, y_test_k = train_test_split(x, y, test_size=0.3, random_state=42)

# Create a KNN classifier with k=3
knn = KNeighborsClassifier(n_neighbors=2)

# Train the classifier on the training data
knn.fit(X_train_k, y_train_k)

# Predict the classes of the test set
y_pred_k = knn.predict(X_test_k)

# Calculate the accuracy of the classifier
accuracy_k = accuracy_score(y_test_k, y_pred_k)
print("Accuracy of KNN:", accuracy_k*100)
#Create the confusion matrix
cm_k = confusion_matrix(y_test_k, y_pred_k)

# Visualize the confusion matrix
plt.figure(figsize=(8,8))
plt.imshow(cm_k, interpolation='nearest', cmap='Blues',vmin=0,vmax=225)
for i in range(cm_k.shape[0]):
    for j in range(cm_k.shape[1]):
        plt.text(j, i, str(cm_k[i,j]), horizontalalignment='center', verticalalignment='center')
plt.title('WINE:(RF&KNN)', size=15)
plt.colorbar()
tick_marks = np.arange(len(set(y)))
plt.xticks(tick_marks, sorted(set(y)), size=10)
plt.yticks(tick_marks, sorted(set(y)), size=10)
plt.xlabel('Predicted label', size=12)
plt.ylabel('True label', size=12)
plt.tight_layout()
plt.show()
```

```
Accuracy of KNN: 43.440233236151606
```



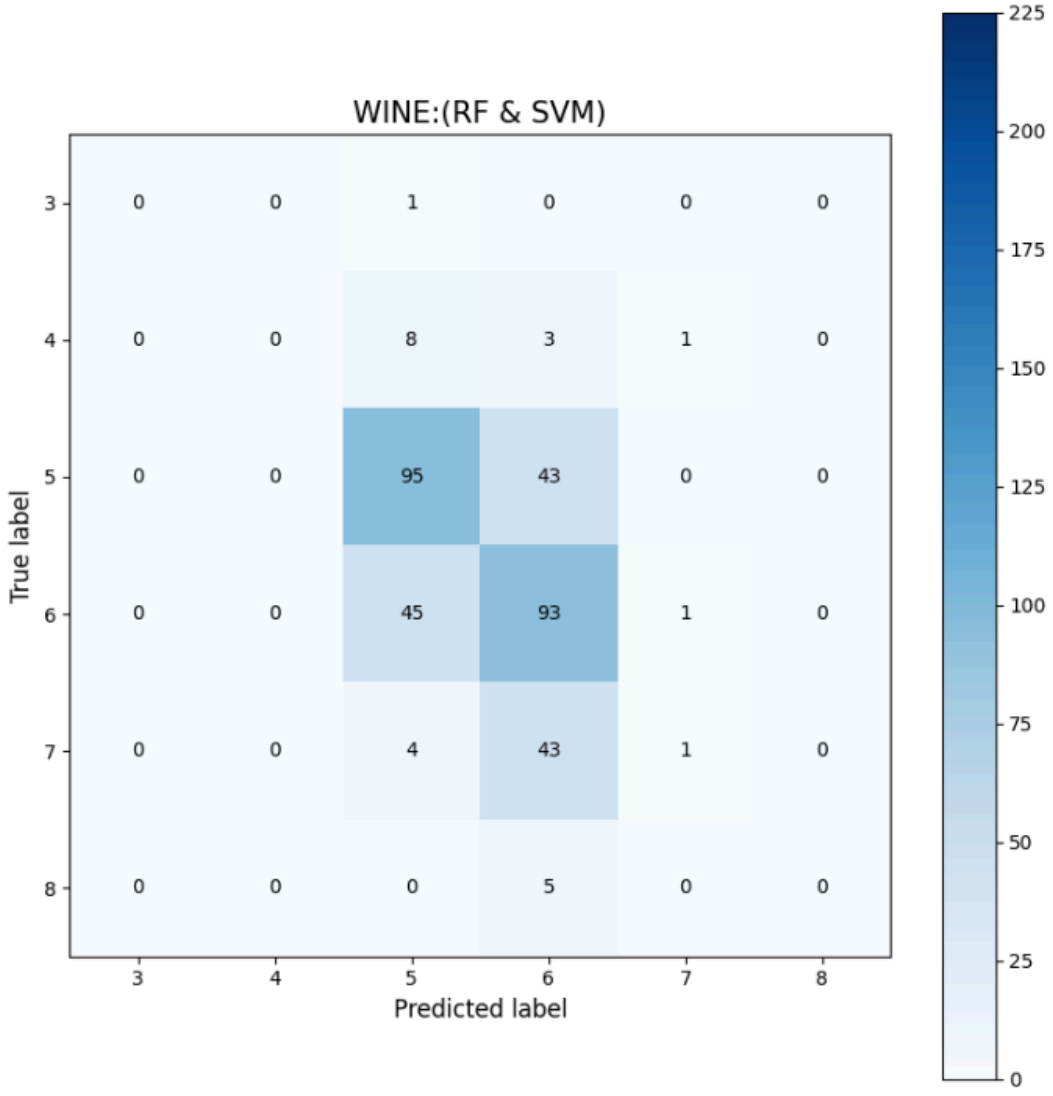
```
#Decision Tree
# Split the dataset into training and testing sets
X_train_dt, X_test_dt, y_train_dt, y_test_dt = train_test_split(x, y, test_size=0.7)

# Train a decision tree classifier
clf = DecisionTreeClassifier()
clf.fit(X_train_dt, y_train_dt)

# Predict the classes of the test set
y_pred_dt = clf.predict(X_test_dt)

# Evaluate the accuracy of the classifier
accuracy_dt = accuracy_score(y_test_dt, y_pred_dt)
print("Accuracy of Decision Tree Classifier:", accuracy_dt*100)

cm_dt = confusion_matrix(y_test_dt, y_pred_dt)
# Visualize the confusion matrix
plt.figure(figsize=(8,8))
plt.imshow(cm_dt, interpolation='nearest', cmap='Blues',vmin=0,vmax=225)
for i in range(cm_dt.shape[0]):
    for j in range(cm_dt.shape[1]):
        plt.text(j, i, str(cm_dt[i,j]), horizontalalignment='center', verticalalignment='center')
plt.title('WINE:(RF & DT)', size=15)
plt.colorbar()
tick_marks = np.arange(len(set(y)))
plt.xticks(tick_marks, sorted(set(y)),size=10)
plt.yticks(tick_marks, sorted(set(y)), size=10)
plt.xlabel('Predicted label', size=12)
plt.ylabel('True label', size=12)
plt.tight_layout()

plt.show()
```

```
Accuracy of Decision Tree Classifier: 50.06242197253433
```



```
#SVM
X_train_svm, X_test_svm, y_train_svm, y_test_svm = train_test_split(x,y, test_size=0.3)
# create a support vector machine classifier with a linear kernel
clf = svm.SVC(kernel='linear')

# train the classifier on the training data
clf.fit(X_train_svm, y_train_svm)

# make predictions on the testing data
y_pred_svm = clf.predict(X_test_svm)

# print the accuracy score
print("Accuracy of SVM:", clf.score(X_test_svm, y_test_svm)*100)
# Create the confusion matrix
cm_svm = confusion_matrix(y_test_svm, y_pred_svm)

# Visualize the confusion matrix
plt.figure(figsize=(8,8))
plt.imshow(cm_svm, interpolation='nearest', cmap='Blues',vmin=0,vmax=225)
for i in range(cm_svm.shape[0]):
    for j in range(cm_svm.shape[1]):
        plt.text(j, i, str(cm_svm[i,j]), horizontalalignment='center', verticalalignment='center')

plt.title('WINE:(RF & SVM)', size=15)
plt.colorbar()
tick_marks = np.arange(len(set(y)))
plt.xticks(tick_marks, sorted(set(y)),  size=10)
plt.yticks(tick_marks, sorted(set(y)), size=10)
plt.xlabel('Predicted label', size=12)
plt.ylabel('True label', size=12)
plt.tight_layout()
plt.show()
```

```
Accuracy of SVM: 55.10204081632652
```

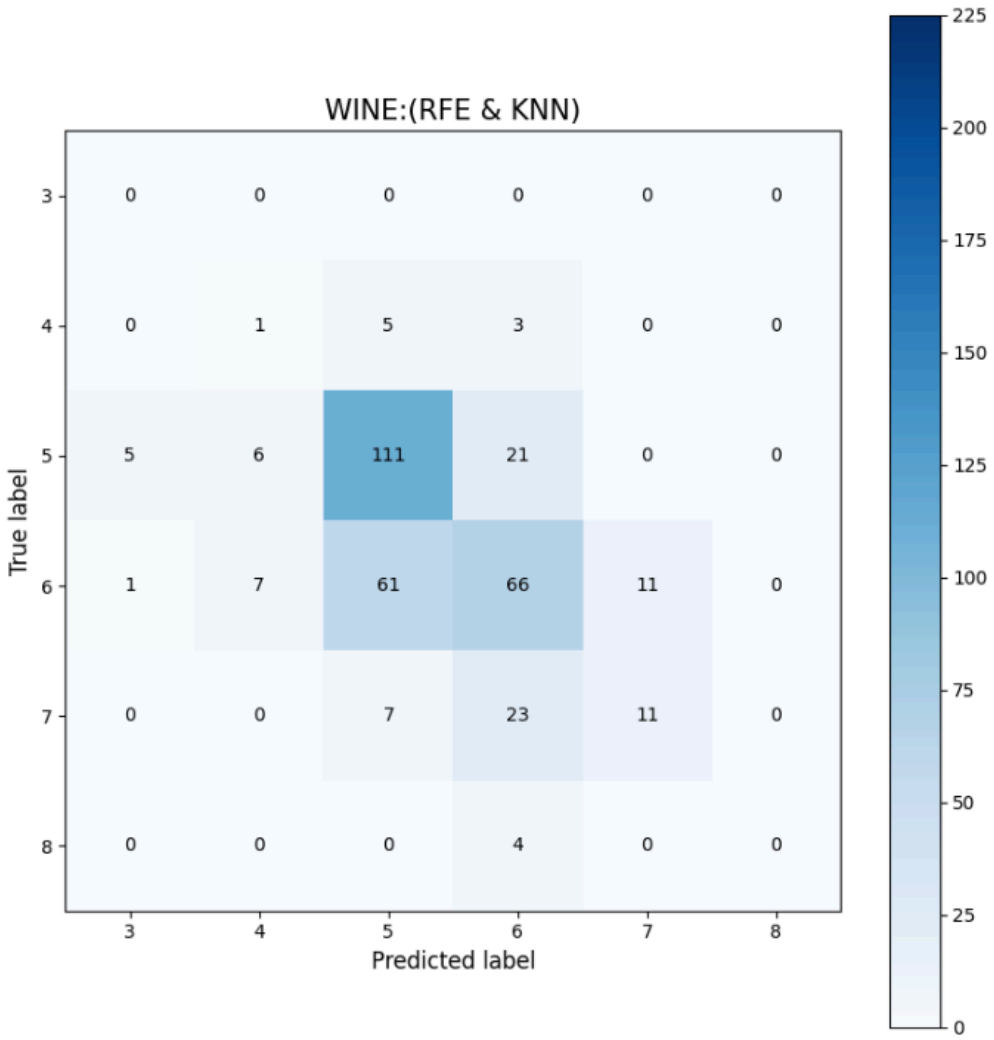


```
from sklearn.feature_selection import SelectFromModel
import numpy as np
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
from sklearn import svm
import pandas as pd
from sklearn.metrics import confusion_matrix
import matplotlib.pyplot as plt
from sklearn.linear_model import Ridge
from sklearn.feature_selection import RFECV

df = pd.read_csv('WineQT.csv')

# Split the features and target variable
X = df.drop('quality', axis=1)
y = df['quality']

# Create a Ridge regression model
ridge = Ridge(alpha=1.0)

# Use RFECV to select the most important features
rfecv = RFECV(estimator=ridge, step=1, cv=5)
X_selected = rfecv.fit_transform(X, y)

# Print the selected features
selected_features = np.array(X.columns)[rfecv.support_]
print("Selected Features: ", selected_features)
X_new = pd.DataFrame(X_selected, columns=selected_features)
X_new["quality"] = y
X_new.to_csv('Wine_RFE.csv', index=False)
```

```
Selected Features:  ['volatile acidity' 'chlorides' 'pH' 'sulphates' 'alcohol']
```

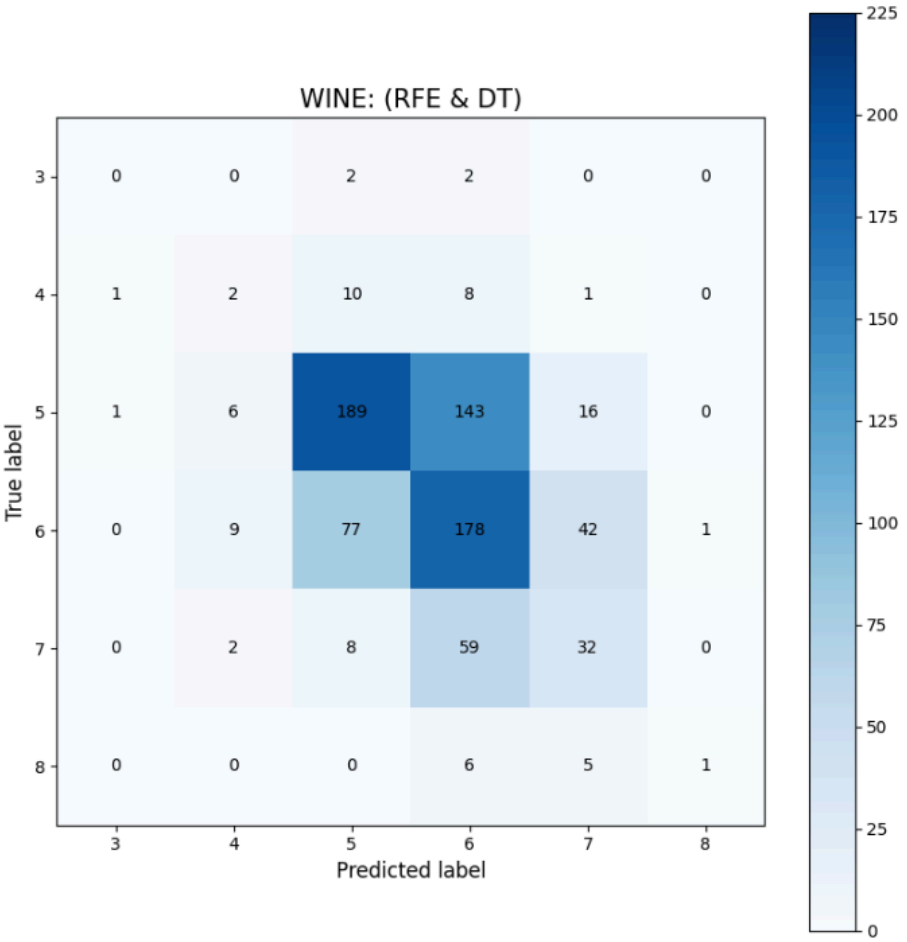
```
#Knn
X_new = pd.read_csv("Wine_RFE.csv")
x = X_new.drop(X_new.columns[len(X_new.columns)-1],axis=1)
y = X_new[X_new.columns[len(X_new.columns)-1]]
#print(x,"\n",y)
# Split the dataset into training and testing sets
X_train_k, X_test_k, y_train_k, y_test_k = train_test_split(x, y, test_size=0.3, random_state=42)

# Create a KNN classifier with k=3
knn = KNeighborsClassifier(n_neighbors=2)

# Train the classifier on the training data
knn.fit(X_train_k, y_train_k)

# Predict the classes of the test set
y_pred_k = knn.predict(X_test_k)

# Calculate the accuracy of the classifier
accuracy_k = accuracy_score(y_test_k, y_pred_k)
print("Accuracy of KNN:", accuracy_k*100)
# Create the confusion matrix
cm_k = confusion_matrix(y_test_k, y_pred_k)

# Visualize the confusion matrix
plt.figure(figsize=(8,8))
plt.imshow(cm_k, interpolation='nearest', cmap='Blues',vmin=0,vmax=225)
for i in range(cm_k.shape[0]):
    for j in range(cm_k.shape[1]):
        plt.text(j, i, str(cm_k[i,j]), horizontalalignment='center', verticalalignment='center')
plt.title('WINE:(RFE & KNN)', size=15)
plt.colorbar()
tick_marks = np.arange(len(set(y)))
plt.xticks(tick_marks, sorted(set(y)), size=10)
plt.yticks(tick_marks, sorted(set(y)), size=10)
plt.xlabel('Predicted label', size=12)
plt.ylabel('True label', size=12)
plt.tight_layout()
plt.savefig('WINE_RFE_KNN.png')
plt.show()
```

```
Accuracy of KNN: 55.10204081632652
```



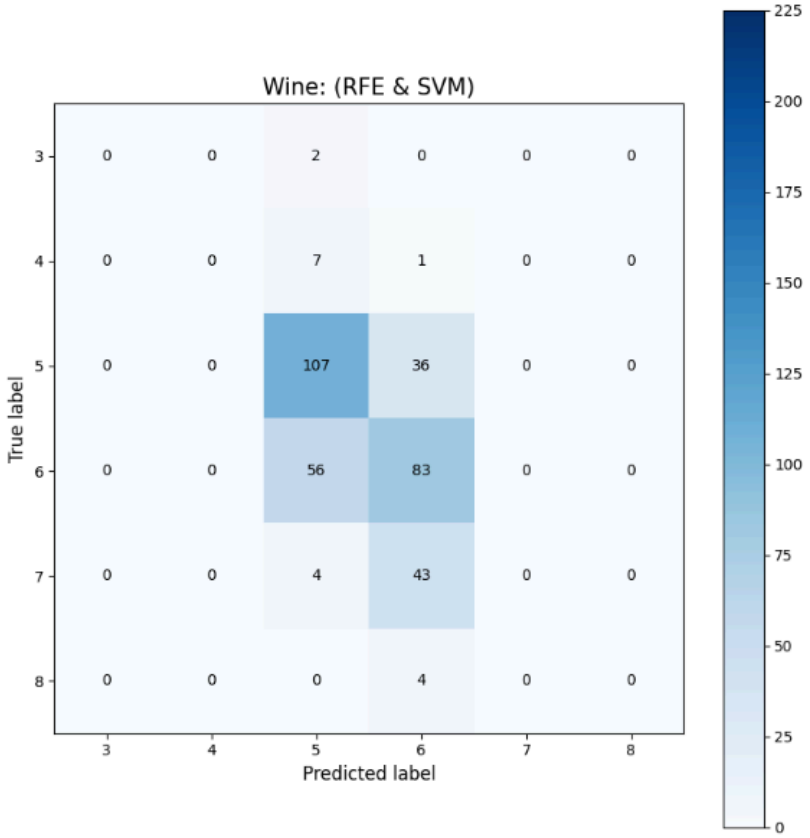

```
#Decision Tree
# Split the dataset into training and testing sets
X_train_dt, X_test_dt, y_train_dt, y_test_dt = train_test_split(x, y, test_size=0.7)

# Train a decision tree classifier
clf = DecisionTreeClassifier()
clf.fit(X_train_dt, y_train_dt)

# Predict the classes of the test set
y_pred_dt = clf.predict(X_test_dt)

# Evaluate the accuracy of the classifier
accuracy_dt = accuracy_score(y_test_dt, y_pred_dt)
print("Accuracy of Decision Tree Classifier:", accuracy_dt*100)
# Create the confusion matrix
cm_dt = confusion_matrix(y_test_dt, y_pred_dt)

# Visualize the confusion matrix
plt.figure(figsize=(8,8))
plt.imshow(cm_dt, interpolation='nearest', cmap='Blues',vmin=0,vmax=225)
for i in range(cm_dt.shape[0]):
    for j in range(cm_dt.shape[1]):
        plt.text(j, i, str(cm_dt[i,j]), horizontalalignment='center', verticalalignment='center')
plt.title('WINE: (RFE & DT)', size=15)
plt.colorbar()
tick_marks = np.arange(len(set(y)))
plt.xticks(tick_marks, sorted(set(y)), size=10)
plt.yticks(tick_marks, sorted(set(y)), size=10)
plt.xlabel('Predicted label', size=12)
plt.ylabel('True label', size=12)
plt.tight_layout()
plt.savefig('WINE_RFE_DT.png')
plt.show()
```

```
Accuracy of Decision Tree Classifier: 50.43695380774032
```



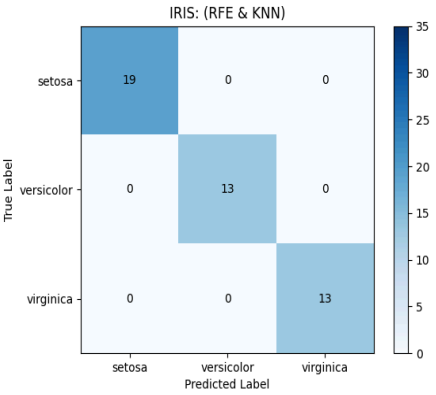

```
#SVM
X_train_svm, X_test_svm, y_train_svm, y_test_svm = train_test_split(x,y, test_size=0.3)
# create a support vector machine classifier with a linear kernel
clf = svm.SVC(kernel='linear')

# train the classifier on the training data
clf.fit(X_train_svm, y_train_svm)

# make predictions on the testing data
y_pred_svm = clf.predict(X_test_svm)

# print the accuracy score
print("Accuracy of SVM:", clf.score(X_test_svm, y_test_svm)*100)
# Create the confusion matrix
cm_svm = confusion_matrix(y_test_svm, y_pred_svm)

# Visualize the confusion matrix
plt.figure(figsize=(8,8))
plt.imshow(cm_svm, interpolation='nearest', cmap='Blues',vmin=0,vmax=225)
for i in range(cm_svm.shape[0]):
    for j in range(cm_svm.shape[1]):
        plt.text(j, i, str(cm_svm[i,j]), horizontalalignment='center', verticalalignment='center')
plt.title('Wine: (RFE & SVM)', size=15)
plt.colorbar()
tick_marks = np.arange(len(set(y)))
plt.xticks(tick_marks, sorted(set(y)), size=10)
plt.yticks(tick_marks, sorted(set(y)), size=10)
plt.xlabel('Predicted label', size=12)
plt.ylabel('True label', size=12)
plt.tight_layout()
plt.savefig('WINE_RFE_SVM.png')
plt.show()
```

```
Accuracy of SVM: 55.39358600583091
```

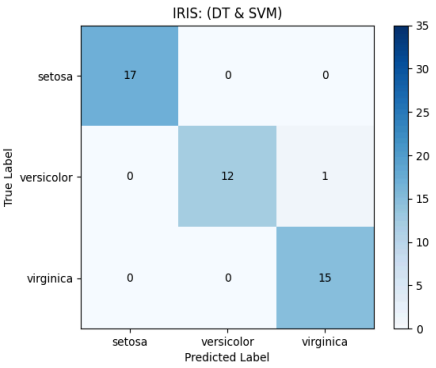

Wine: (RFE & SVM)
True label
Predicted label
3
4
5
6
7
8
0 0 2 0 0 0
0 0 7 1 0 0
0 0 107 36 0 0
0 0 56 83 0 0
0 0 4 43 0 0
0 0 0 4 0 0
225
200
175
150
125
100
75
50
25
0

7. RESULTS

CONFUSION MATRIX:IRIS

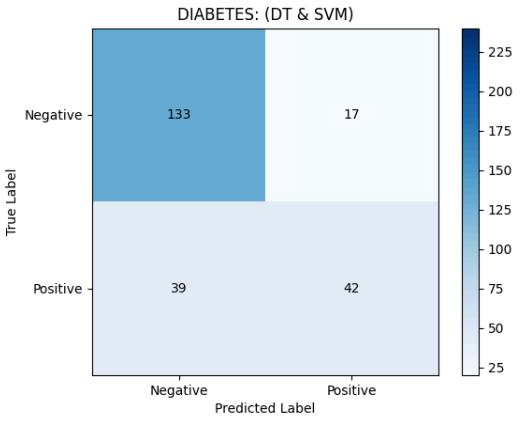


(Best case)

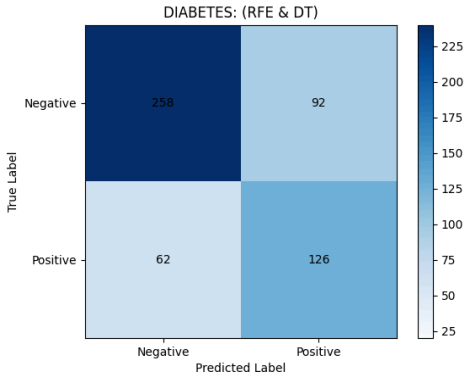


(Worst case)

CONFUSION MATRIX:DIABETES

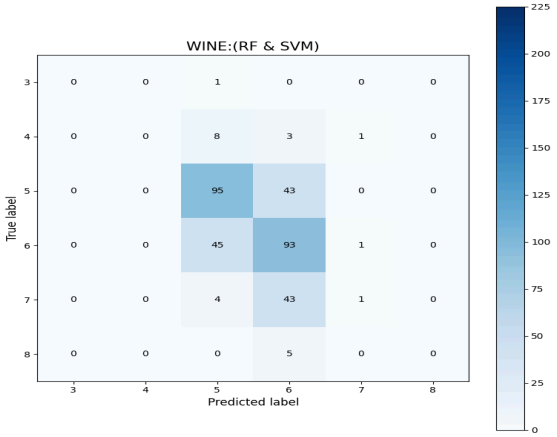


(Best case)

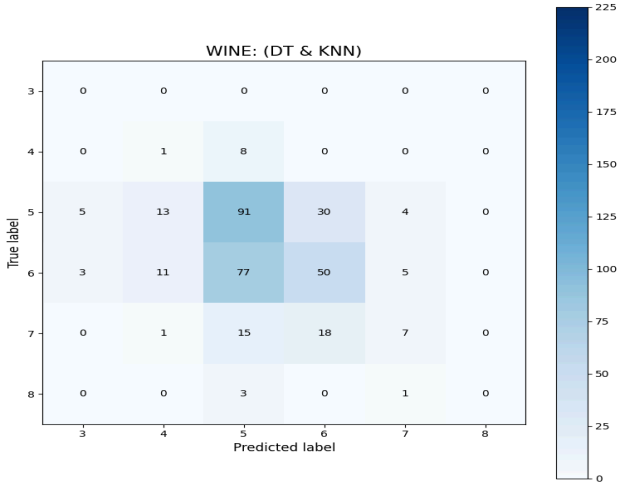


(Worst case)

CONFUSION MATRIX:WINE

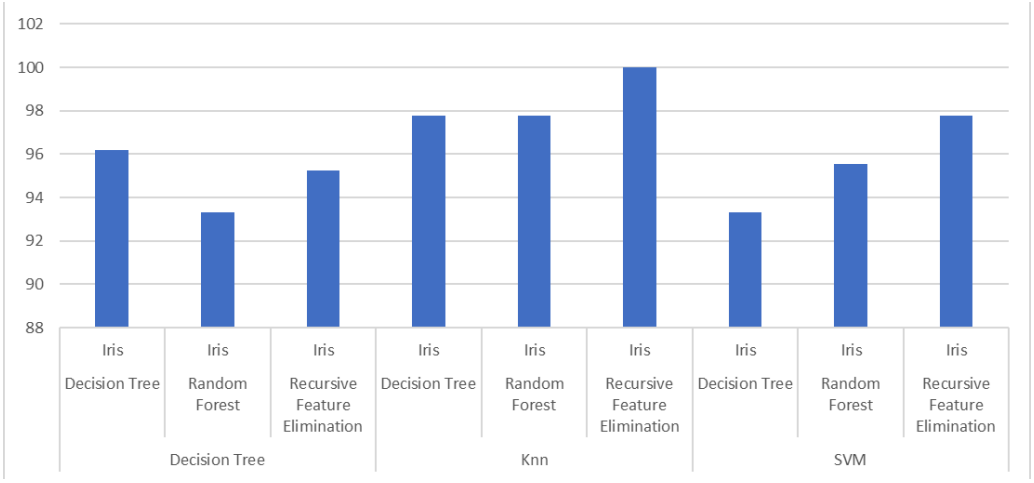


(Best case)

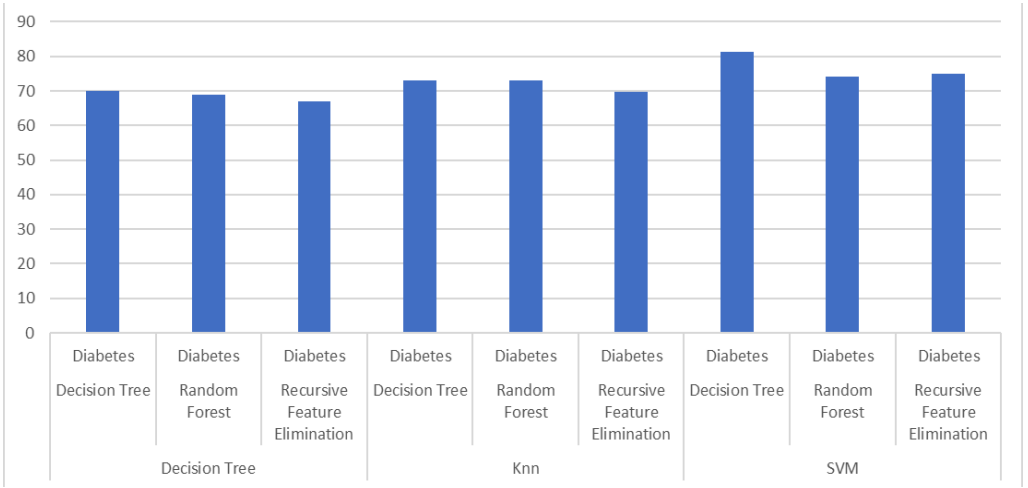


(Worst case)

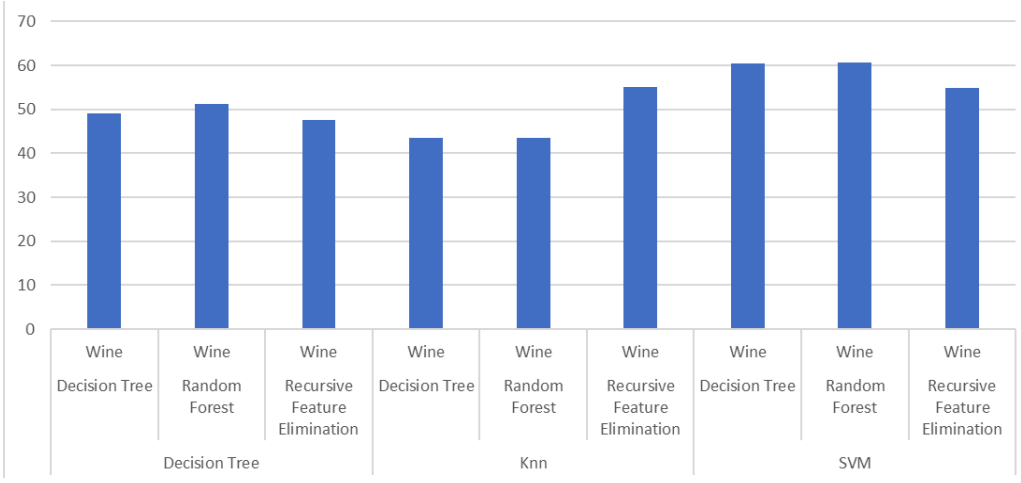
ACCURACY OF IRIS DATA



ACCURACY OF DIABETES DATA



ACCURACY OF WINE DATA



8. CONCLUSION

- In the case of Iris data, the combination of feature selection algorithm and classifier that gives the best accuracy is Recursive Feature Elimination and KNN. Hence, for similar **multivariate** datasets this combination can be applied to obtain the best classification result.
- For Diabetes data, the best combination is Decision Tree and Support Vector Machine. This combination can be applied to similar **numeric** datasets for best results.
- For Wine data, Random Forest with Support Vector Machine gives the most accurate result. Therefore, this combination is suitable for similar kind of **categorical data.**

9. REFERENCES

- Jundong Li, Kewei Cheng, Suhang Wang, Fred Morstatter, Robert P. Trevino, Jiliang Tang, and Huan Liu. 2017. Feature Selection: A Data Perspective. ACM Comput. Surv. 50, 6, Article 94 (November 2018), 45 pages. https://doi.org/10.1145/3136625
- Utkarsh Mahadeo Khaire, R. Dhanalakshmi,Stability of feature selection algorithm: A review,Journal of King Saud University - Computer and Information Sciences,Volume 34, Issue 4,2022,Pages 1060-1073,ISSN 1319-1578, https://doi.org/10.1016/j.jksuci.2019.06.012.
- edureka! (2018, June 14). *Decision Tree Algorithm | Decision Tree in Python | Machine Learning Algorithms | Edureka* [Video]. YouTube. https://www.youtube.com/watch?v=qDcl-FRnwSU
- Cloud and ML Online. (2019, June 22). *Support Vector Machine - SVM - Classification Implementation for Beginners (using python) - Detailed* [Video]. YouTube. https://www.youtube.com/watch?v=7sz4WpkUIIs
- Simplilearn. (2018, June 6). *KNN Algorithm In Machine Learning | KNN Algorithm Using Python | K Nearest Neighbor | Simplilearn* [Video]. YouTube. https://www.youtube.com/watch?v=4HKqjENq9OU
- Hackers Realm. (2022, June 9). *Recursive Feature Elimination (RFE) | Feature Selection | Python* [Video]. YouTube. https://www.youtube.com/watch?v=vxdVKbAv6as
- Unfold Data Science. (2020, January 31). *Implementing Random Forest In Python|How to Implement Random Forest In Python|Random Forest ML* [Video]. YouTube. https://www.youtube.com/watch?v=MxiktOPmhV8