Visvesvaraya National Institute of Technology (VNIT), Nagpur

Machine Learning with Python (ECL443)

Lab Report

Submitted by :
Prajyot Jadhav (BT20ECE046)
Semester 7

Submitted to :
Dr. Saugata Sinha
(Course Instructor)
Department of Electronics and Communication Engineering,
VNIT Nagpur

Contents

Experiment-3

Aim: To train a SVM classifier that can distinguish between the different types of iris.

Abstract: The objective of this assignment is to develop a classification model using Support Vector Machines (SVMs) to classify different types of flowers. We divide the dataset into training and testing subsets. We experiment with a variety of kernel functions, including linear, polynomial, and radial basis function (RBF) kernels, while tuning hyperparameters through a comprehensive grid search approach. This aids in the identification of the optimal hyperparameter values, as evidenced by error/accuracy vs. hyperparameter value curves. To comprehensively assess the performance of each SVM model, we report the accuracy, sensitivity and specificity. Finally, we compare the SVM results with those obtained from an Artificial Neural Network (ANN), considering not only accuracy but also the computational time required for model training.

Introduction: In this experiment, the MATLAB dataset "fisheriris_matlab.mat" is used for a multi-class classification problem. We train a SVM classifier for this classification problem. SVMs are particularly well-suited for cases where the decision boundary between classes is not easily linearly separable. The goal of the SVM algorithm is to create the best line or decision boundary that can segregate n-dimensional space into classes so that we can easily put the new data point in the correct category in the future. This best decision boundary is called a hyperplane. SVM chooses the extreme points/vectors that help in creating the hyperplane. These extreme cases are called as support vectors, and hence algorithm is termed as Support Vector Machine. We also experiment with various kernel functions, optimize hyperparameters through grid search, and evaluate model performance in terms of accuracy, sensitivity, specificity.

Method:

- The dataset is loaded from a CSV file using Pandas' read_csv function. The dataset contains information about iris flowers. The dataset is split into two parts - independent variables (features) denoted as X and dependent variable (target) denoted as y. This separation is essential for training and testing machine learning models.
- The dataset is further split into training and testing sets using train_test_split from scikit-learn. It divides the data into training (80%) and testing (20%) subsets, ensuring reproducibility by setting a random seed.

- Grid search is employed to find the best hyperparameters for the Support Vector Classifier (SVC) for the RBF kernel using the GridSearchCV function from sklearn. It explores a range of values for the regularization parameter C and the kernel parameter gamma.Stratified Shuffle Split cross-validation is used to evaluate the model's performance.The best hyperparameters are obtained along with the corresponding score. For obtaining the graphs of score vs the hyperparameter value, one of the hyperparamter is set constant, equal to the optimal value and the other is varied.
- Similar to the RBF kernel, hyperparameter tuning and cross-validation are performed to find the best parameters for the Sigmoid kernel. Grid search is performed for the hyperparmaters C, gamma and coef0. The best parameters and corresponding score are obtained.
- For the polynomial kernel, hyperparameter tuning and cross-validation are carried out to find the optimal parameters for the SVC. The best values for the parameters C, gamma, coef0 and degree and also corresponding score are obtained.
- Three separate SVM classifiers (RBF, Sigmoid, and Polynomial) are trained using the best parameters obtained from the grid search. The models are trained on the training data (X_train and y_train) using the fit method. Predictions are made on the test data (X_test) using the predict method.
- The confusion matrix is computed using confusion_matrix from scikit-learn, which helps in understanding the model's performance. True Positives (TP), True Negatives (TN), False Positives (FP), and False Negatives (FN) are extracted from the confusion matrix. Specificity (True Negative Rate) and Sensitivity (True Positive Rate) are calculated using the derived values.
- Also, the accuracy of each SVM classifier is calculated by dividing the number of correct predictions (corrPred) by the total number of predictions.
- The performance of the SVM classifier is compared with ANN using the network from the previous assignment. The network architecture was slightly modified for multi-class classification to have softmax activation function at the output layer and 102 neurons in the hidden layer. The accuracy of this ANN architecture was obtained.

Results:

- The training and testing set are obtained by splitting the original data in the ratio of 80 % and 20 % respectively. The training set of (120,4) with the labels of (120), the testing set of (30,4) with the labels of (30) are obtained.

```
Training data size:  (120, 4)
Training label size:  (120,)
Testing data size:  (30, 4)
Testing label size:  (30,)
```

Figure 1: Training and testing sets

- The SVM classifier is implemented with the RBF kernel. The best values of hyperparameters was obtained from grid search and the following graphs were obtained. Also, the accuracy, TPR and the TNR were obtained for the RBF kernel.

```
The best parameters are {'C': 4.6415888336127775, 'gamma': 0.01} with a score of 0.97
```

Figure 2: Optimal values of hyperparameters for RBF kernel

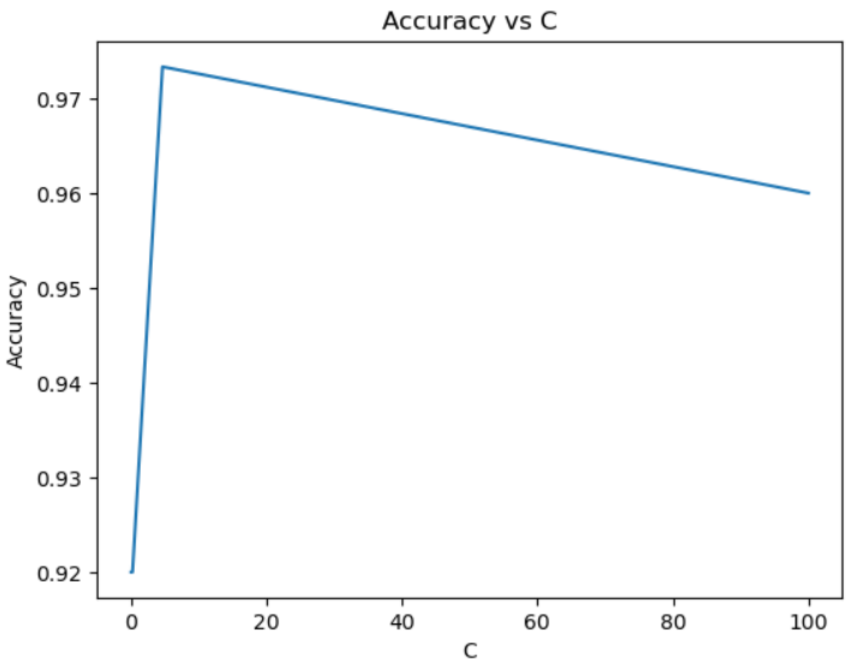

Figure 3: score vs C for RBF kernel

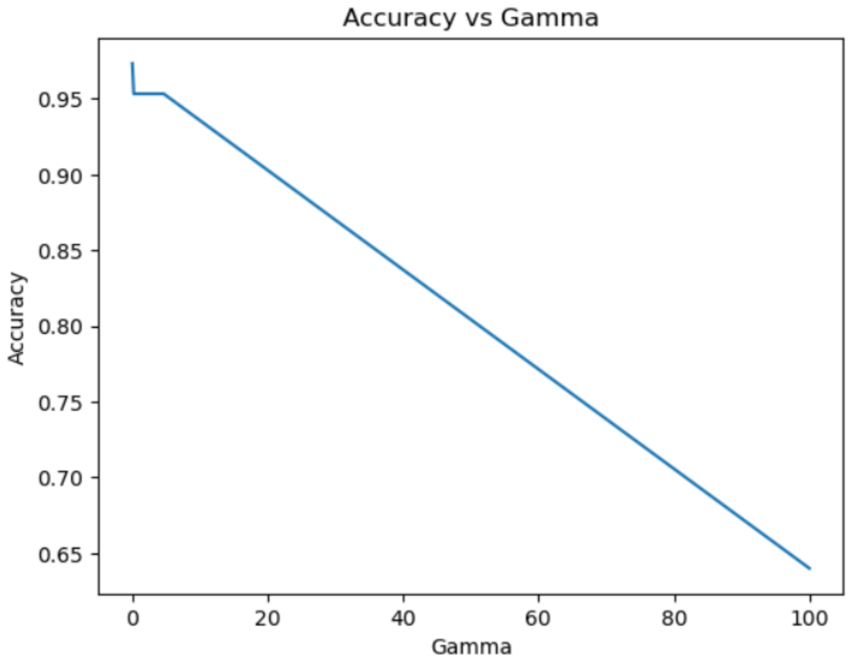

Figure 4: score vs gamma for RBF kernel

```
Confusion Matrix:
[[11  0  0]
 [ 0  8  1]
 [ 0  2  8]]
Class 0: TPR = 1.00, TNR = 1.00
Class 1: TPR = 0.89, TNR = 0.90
Class 2: TPR = 0.80, TNR = 0.95
Correct predictions:  27
False predictions 3
Accuracy of the SVC Clasification is:  0.9
```

Figure 5: Accuracy, sensitivity and specificity for RBF kernel

- The above steps were repeated with Sigmoid kernel, the optimal values of hyperparamters and the following graphs were obtained. Also, the accuracy, TPR and the TNR were obtained for the Sigmoid kernel.

```
The best parameters are {'C': 4.6415888336127775, 'coef0': 0.021544346900318846, 'gamma': 0.01} with a score of 0.86
```

Figure 6: Optimal values of hyperparameters for Sigmoid kernel

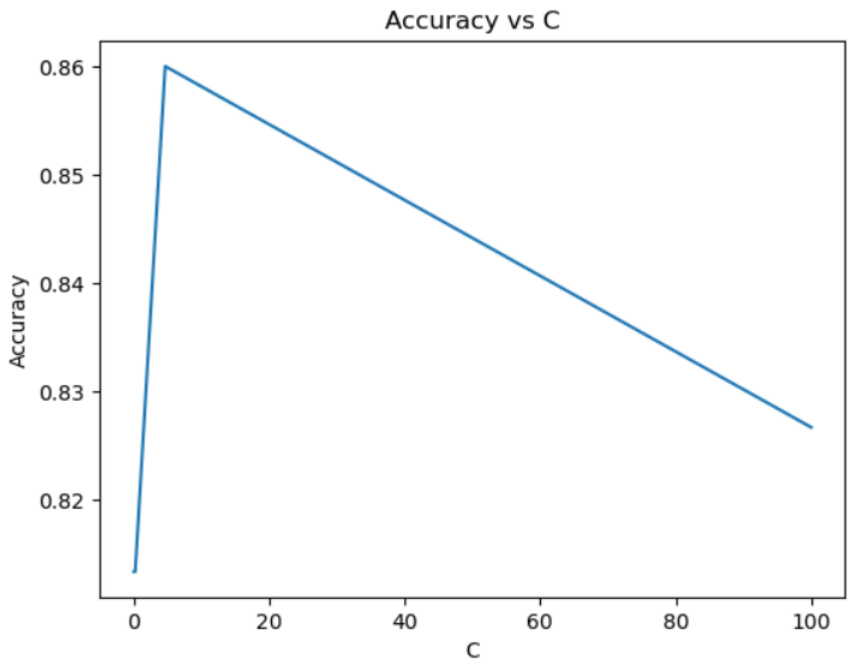

Figure 7: score vs C for Sigmoid kernel

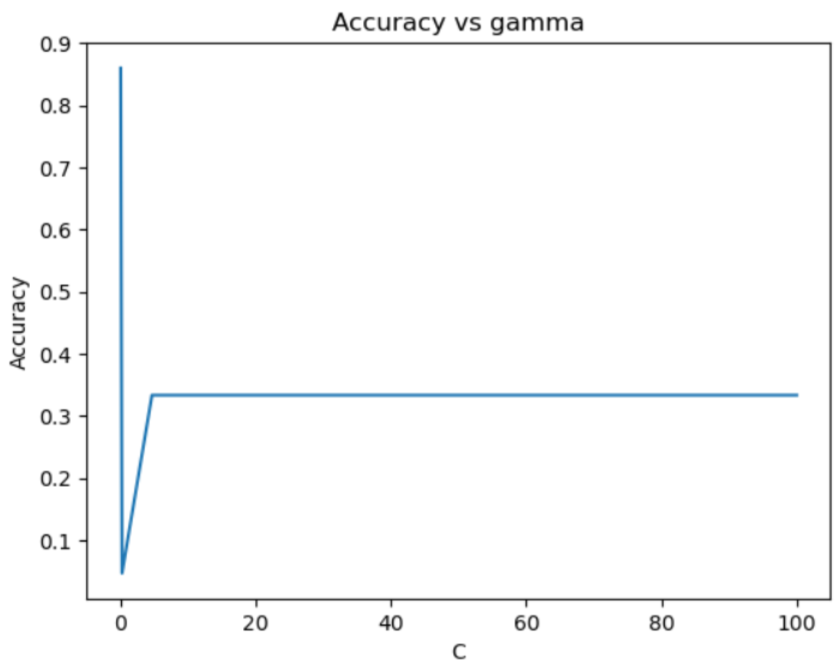

Figure 8: score vs gamma for Sigmoid kernel

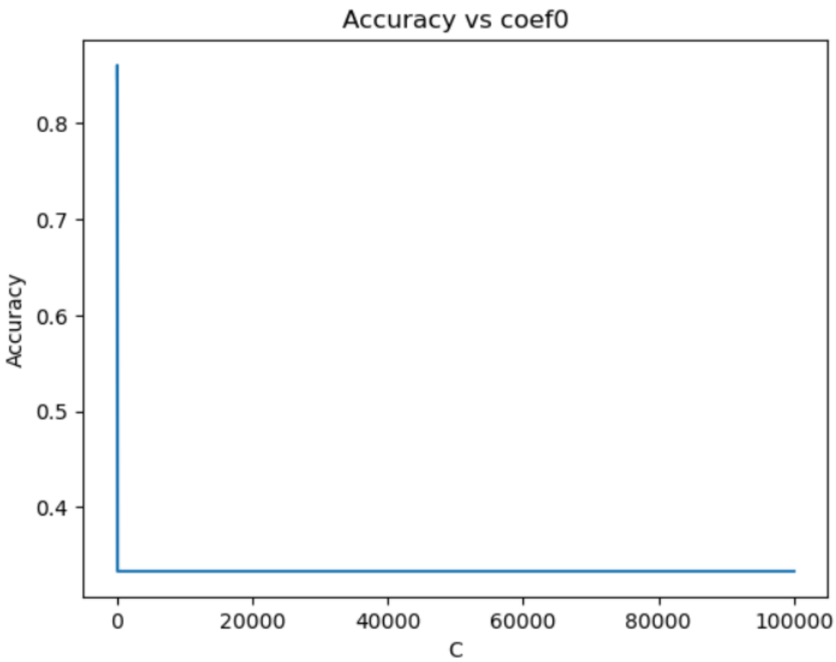

Figure 9: score vs coef for Sigmoid kernel

```
Confusion Matrix:
[[11  0  0]
 [ 0  8  1]
 [ 0  2  8]]
Class 0: TPR = 1.00, TNR = 1.00
Class 1: TPR = 0.89, TNR = 0.90
Class 2: TPR = 0.80, TNR = 0.95
Correct predictions:  27
False predictions 3
Accuracy of the SVC Clasification is:  0.9
```

Figure 10: Accuracy, sensitivity and specificity for Sigmoid kernel

- For the SVM classifier with polynomial kernel, the following optimal hyper-paramter values and the following graphs were obtained. Also, the accuracy, TPR and the TNR were obtained for the Polynomial kernel.

```
The best parameters are {'C': 0.01, 'coef0': 100.0, 'degree': 2, 'gamma': 0.21544346900318834} with a score of 0.9
```

Figure 11: Optimal values of hyperparameters for Polynomial kernel

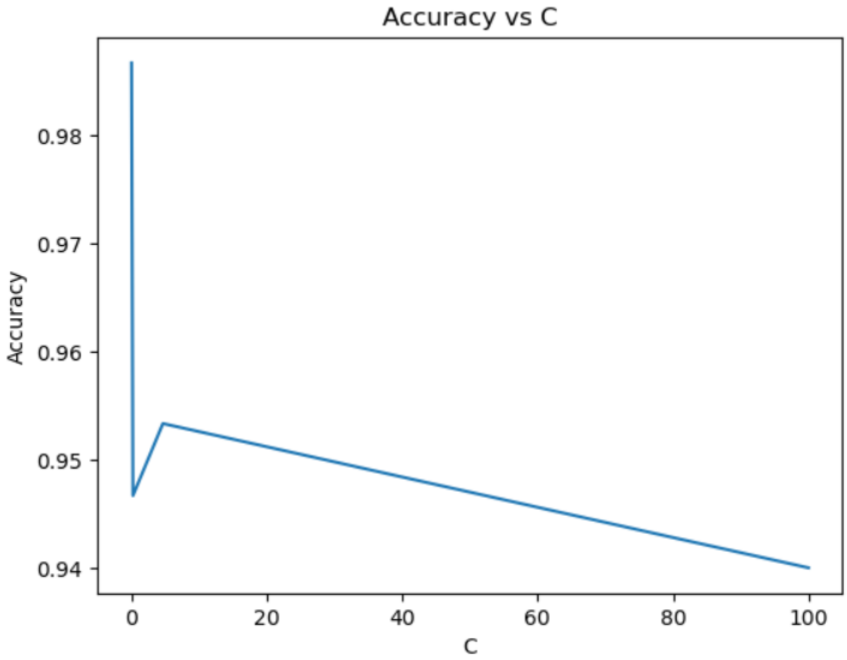

Figure 12: score vs C for Polynomial kernel

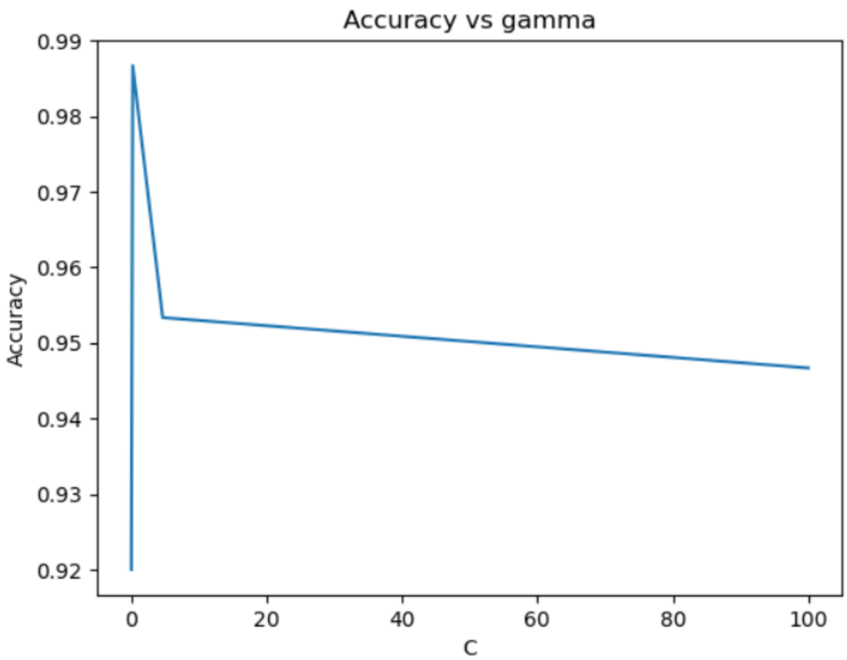

Figure 13: score vs gamma for Polynomial kernel

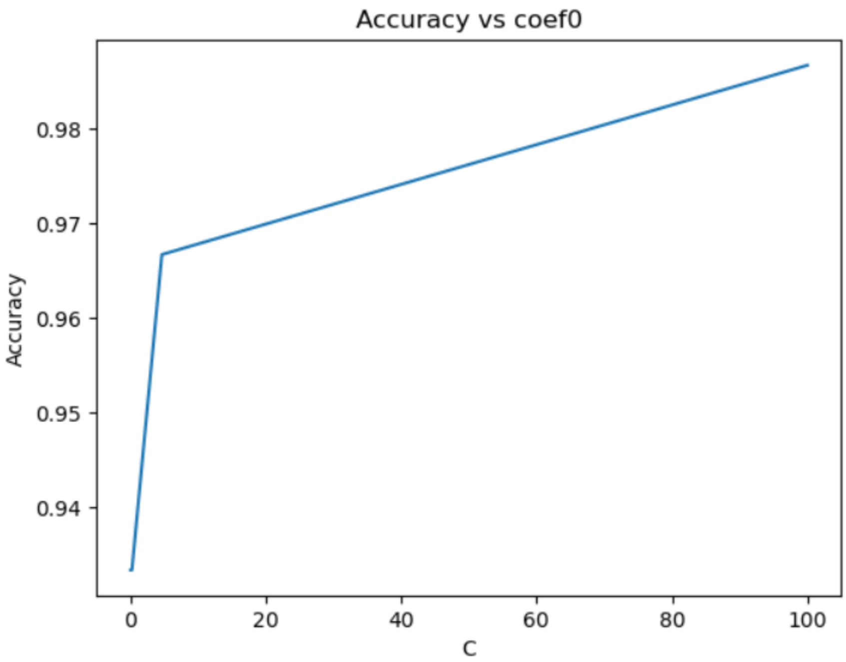

Figure 14: score vs coef for Polynomial kernel

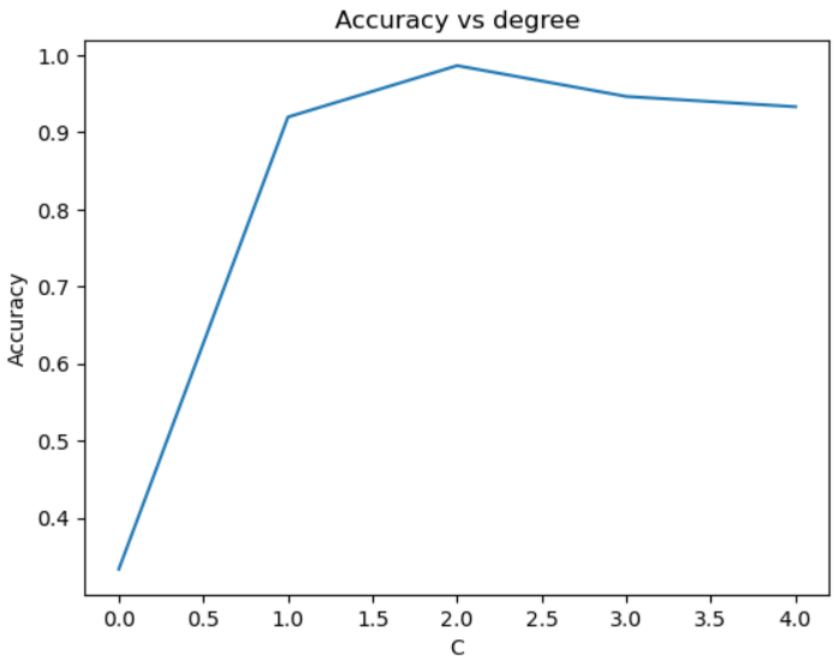

Figure 15: score vs degree for Polynomial kernel

```
Confusion Matrix:
[[11  0  0]
 [ 0  8  1]
 [ 0  0 10]]
Class 0: TPR = 1.00, TNR = 1.00
Class 1: TPR = 0.89, TNR = 1.00
Class 2: TPR = 1.00, TNR = 0.95
Correct predictions:  29
False predictions 1
Accuracy of the SVC Clasification is:  0.9666666666666667
```

Figure 16: Accuracy, sensitivity and specificity for Polynomial kernel

- The same classfication problem was solved with the ANN network from the previous assignment. An accuracy of 86.67 % was obtained with the following confusion matrix.

```
Accuracy: 86.66666666666667%
Confusion Matrix:
 [[11  0  0]
 [ 0  8  1]
 [ 0  3  7]]
Correct predictions:  26
False predictions 4
Accuracy of the SVC Clasification is:  0.8666666666666667
```

Figure 17: ANN accuracy

Discussion:

- The SVM classifier is implemented with different kernels namely, RBF, Sigmoid and Polynomial. When the three SVM classifiers were trained with the best parameters obtained from grid search, it was observed that highest accuracy was obtained for the Polynomial kernel. However, the best scores obtained during grid search for the three kernels RBF, Sigmoid and Polynomial are 0.97, 0.86 and 0.95 respectively. Since the score for Sigmoid kernel is significantly less than the scores for RBF and Polynomial, it is possible that the hyperparamters for Sigmoid could be further optimized. Also, while performing grid search, the number of parameters that need to be optimized for polynomial and sigmoid are 4 and 3 respectively. So, while iterating over a higher number of values for each paramter leads to high number of combinations which is computationally heavy.
- The ANN architecture used for comparing the performance with SVM gave an accuracy of only 86.67 %. However, this ANN network has only one hidden layer with 102 neurons. There are 4 neurons in the input layer and 3 neurons in the output layer. It is possible that such low accuracy is due to the network

being very shallow. A higher accuracy can be obtained by increasing the number of hidden layers in the network.

- Since the ANN has only one hidden layer, a large difference in the training time for SVM and this ANN was not observed. However, training SVM was faster than training the neural network. Training deep ANNs often involves many iterations of forward and backward passes through the network. So deep neural networks, which have many layers and parameters, take a long time to train. So, it can be said that, training even deeper ANNs will require significantly more time than training SVM classifier.

Conclusion: The SVM classifier was trained to distinguish between different types of iris. The optimal values of hyperparameters was obtained for different kernels and also the performance of the SVM classifier was compared with ANNs.

Appendix:

```
# SVC Classification

# Importing the libraries
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
import pandas as pd


# In[352]:


from sklearn.model_selection import GridSearchCV, ...
    StratifiedShuffleSplit
from sklearn.svm import SVC


# In[353]:


# Importing the dataset
dataset = pd.read_csv(r"C:\Users\Prajyot\Downloads\iris.csv")


# In[354]:


#looking at the first 5 values of the dataset
```

```python
dataset.head()


# In[355]:


#Spliting the dataset in independent and dependent variables
X = dataset.iloc[:,:4].values
y = dataset['species'].values


# In[356]:


# Splitting the dataset into the Training set and Test set
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, ...
    test_size = 0.20, random_state = 82)


# In[357]:


print("Training data size: ",X_train.shape)
print("Training label size: ",y_train.shape)
print("Testing data size: ",X_test.shape)
print("Testing label size: ",y_test.shape)


# In[358]:


# Feature Scaling to bring the variable in a single scale
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)


# # RBF Kernel

# In[359]:


from sklearn.model_selection import GridSearchCV, ...
    StratifiedShuffleSplit

C_range = np.logspace(-2, 2, 4)
gamma_range = np.logspace(-2, 2, 4)
```

```
param_grid = dict(gamma=gamma_range, C=C_range)
cv = StratifiedShuffleSplit(n_splits=5, test_size=0.2, ...
    random_state=42)
grid = GridSearchCV(SVC(), param_grid=param_grid, cv=cv)
grid.fit(X, y)

print(
    "The best parameters are %s with a score of %0.2f"
    % (grid.best_params_, grid.best_score_)
)


# In[360]:


# from sklearn.model_selection import GridSearchCV, ...
    StratifiedShuffleSplit

# C_range = np.logspace(-2, 2, 4)
# gamma_range = [0.01]
# param_grid = dict(gamma=gamma_range, C=C_range)
# cv = StratifiedShuffleSplit(n_splits=5, test_size=0.2, ...
    random_state=42)
# grid = GridSearchCV(SVC(), param_grid=param_grid, cv=cv)
# grid.fit(X, y)

# print(
#     "The best parameters are %s with a score of %0.2f"
#     % (grid.best_params_, grid.best_score_)
# )


# In[361]:


# print(grid.cv_results_.keys())


# In[362]:


# print(grid.cv_results_['param_C'])


# In[363]:


# # importing package
# import matplotlib.pyplot as plt
```

```
# x = grid.cv_results_['param_C']
# y = grid.cv_results_['mean_test_score']

# # plot line
# plt.plot(x, y)
# plt.title("Accuracy vs C")
# plt.xlabel("C")
# plt.ylabel("Accuracy")

# plt.show()


# In[364]:


svcclassifier = SVC(kernel = 'rbf', gamma = 0.01, C = ...
    4.6415888336127775, random_state = 0)
svcclassifier.fit(X_train, y_train)

# Predicting the Test set results
y_pred = svcclassifier.predict(X_test)
print(y_pred)


# In[365]:


# Making the Confusion Matrix
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
print('Confusion Matrix:')
print(cm)

#finding accuracy from the confusion matrix.
a = cm.shape
corrPred = 0
falsePred = 0

for row in range(a[0]):
    for c in range(a[1]):
        if row == c:
            corrPred +=cm[row,c]
        else:
            falsePred += cm[row,c]

            n_classes = cm.shape[0]

for i in range(n_classes):
    tp = cm[i, i]
```

```
    fn = sum(cm[i, :]) - tp
    fp = sum(cm[:, i]) - tp
    tn = sum(sum(cm)) - tp - fn - fp

    tpr = tp / (tp + fn)
    tnr = tn / (tn + fp)

    print(f"Class {i}: TPR = {tpr:.2f}, TNR = {tnr:.2f}")

print('Correct predictions: ', corrPred)
print('False predictions', falsePred)
kernelRbfAccuracy = corrPred/(cm.sum())
print ('Accuracy of the SVC Clasification is: ', ...
    corrPred/(cm.sum()))


# # Sigmoid Kernel

# In[366]:


# For the sigmoid kernel

C_range = np.logspace(-2, 2, 4)
gamma_range = np.logspace(-2, 2, 4)
coef0_range = np.logspace(-5, 5, 4)
param_grid = dict(gamma=gamma_range, C=C_range, coef0=coef0_range)
cv = StratifiedShuffleSplit(n_splits=5, test_size=0.2, ...
    random_state=42)
grid = GridSearchCV(SVC(kernel='sigmoid'), ...
    param_grid=param_grid, cv=cv)
grid.fit(X, y)

print(
    "The best parameters are %s with a score of %0.2f"
    % (grid.best_params_, grid.best_score_)
)


# In[367]:


# # For the sigmoid kernel

# C_range = [4.6415888336127775]
# coef0_range = [0.021544346900318846]
# gamma_range = np.logspace(-2, 2, 4)
# param_grid = dict(gamma=gamma_range, C=C_range, coef0=coef0_range)
# cv = StratifiedShuffleSplit(n_splits=5, test_size=0.2, ...
```

```
    random_state=42)
# grid = GridSearchCV(SVC(kernel='sigmoid'), ...
    param_grid=param_grid, cv=cv)
# grid.fit(X, y)

# print(
#     "The best parameters are %s with a score of %0.2f"
#     % (grid.best_params_, grid.best_score_)
# )


# In[368]:


# print(grid.cv_results_.keys())


# In[369]:


# print(grid.cv_results_['param_gamma'])


# In[370]:


# # importing package
# import matplotlib.pyplot as plt
# x = grid.cv_results_['param_gamma']
# y = grid.cv_results_['mean_test_score']

# # plot line
# plt.plot(x, y)
# plt.title("Accuracy vs gamma")
# plt.xlabel("C")
# plt.ylabel("Accuracy")

# plt.show()


# In[371]:


svcclassifier = SVC(kernel = 'sigmoid', C =4.6415888336127775 , ...
    coef0=0.021544346900318846 , gamma = 0.01, random_state = 0)
svcclassifier.fit(X_train, y_train)

# Predicting the Test set results
y_pred = svcclassifier.predict(X_test)
```

```
print(y_pred)


# In[372]:


# Making the Confusion Matrix
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
print('Confusion Matrix:')
print(cm)

#finding accuracy from the confusion matrix.
a = cm.shape
corrPred = 0
falsePred = 0

for row in range(a[0]):
    for c in range(a[1]):
        if row == c:
            corrPred +=cm[row,c]
        else:
            falsePred += cm[row,c]

n_classes = cm.shape[0]

for i in range(n_classes):
    tp = cm[i, i]
    fn = sum(cm[i, :]) - tp
    fp = sum(cm[:, i]) - tp
    tn = sum(sum(cm)) - tp - fn - fp

    tpr = tp / (tp + fn)
    tnr = tn / (tn + fp)

    print(f"Class {i}: TPR = {tpr:.2f}, TNR = {tnr:.2f}")

print('Correct predictions: ', corrPred)
print('False predictions', falsePred)
kernelSigmoidAccuracy = corrPred/(cm.sum())
print ('Accuracy of the SVC Clasification is: ', ...
    corrPred/(cm.sum()))


# # Polynomial Kernel

# In[373]:


```

```
# For the polynomial kernel

C_range = np.logspace(-2, 2, 4)
gamma_range = np.logspace(-2, 2, 4)
coef0_range = np.logspace(-2, 2, 4)
degree_range = [0,1,2,3,4]
param_grid = dict(gamma=gamma_range, C=C_range, ...
    coef0=coef0_range, degree=degree_range)
cv = StratifiedShuffleSplit(n_splits=5, test_size=0.2, ...
    random_state=42)
grid = GridSearchCV(SVC(kernel='poly'), param_grid=param_grid, ...
    cv=cv)
grid.fit(X, y)

print(
    "The best parameters are %s with a score of %0.2f"
    % (grid.best_params_, grid.best_score_)
)


# In[374]:


# # For the polynomial kernel

# C_range = [0.01]
# gamma_range = [0.21544346900318834]
# coef0_range = [100]
# degree_range = [0,1,2,3,4]
# param_grid = dict(gamma=gamma_range, C=C_range, ...
    coef0=coef0_range, degree=degree_range)
# cv = StratifiedShuffleSplit(n_splits=5, test_size=0.2, ...
    random_state=42)
# grid = GridSearchCV(SVC(kernel='poly'), param_grid=param_grid, ...
    cv=cv)
# grid.fit(X, y)

# print(
#     "The best parameters are %s with a score of %0.2f"
#     % (grid.best_params_, grid.best_score_)
# )


# In[375]:


# print(grid.cv_results_.keys())


```

```
# In[376]:


# print(grid.cv_results_['param_degree'])


# In[377]:


# # importing package
# import matplotlib.pyplot as plt
# x = grid.cv_results_['param_degree']
# y = grid.cv_results_['mean_test_score']

# # plot line
# plt.plot(x, y)
# plt.title("Accuracy vs degree")
# plt.xlabel("C")
# plt.ylabel("Accuracy")

# plt.show()


# In[378]:


svcclassifier = SVC(kernel = 'poly', C = 0.01 , coef0 =  ...
    100.0,degree=2, gamma = 0.21544346900318834, random_state = 0)
svcclassifier.fit(X_train, y_train)

# Predicting the Test set results
y_pred = svcclassifier.predict(X_test)
print(y_pred)


# In[379]:


# Making the Confusion Matrix
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
print('Confusion Matrix:')
print(cm)

#finding accuracy from the confusion matrix.
a = cm.shape
corrPred = 0
falsePred = 0

```

```
for row in range(a[0]):
    for c in range(a[1]):
        if row == c:
            corrPred +=cm[row,c]
        else:
            falsePred += cm[row,c]

n_classes = cm.shape[0]

for i in range(n_classes):
    tp = cm[i, i]
    fn = sum(cm[i, :]) - tp
    fp = sum(cm[:, i]) - tp
    tn = sum(sum(cm)) - tp - fn - fp

    tpr = tp / (tp + fn)
    tnr = tn / (tn + fp)

    print(f"Class {i}: TPR = {tpr:.2f}, TNR = {tnr:.2f}")

print('Correct predictions: ', corrPred)
print('False predictions', falsePred)
kernelPolyAccuracy = corrPred/(cm.sum())
print ('Accuracy of the SVC Clasification is: ', ...
    corrPred/(cm.sum()))


# In[332]:


from sklearn.metrics import confusion_matrix, roc_curve, auc
import matplotlib.pyplot as plt

# Making the Confusion Matrix
cm = confusion_matrix(y_test, y_pred)
print(cm)

n_classes = cm.shape[0]

# Calculate ROC and AUROC for each class
fpr = dict()
tpr = dict()
roc_auc = dict()

for i in range(n_classes):
    tp = cm[i, i]
    fn = sum(cm[i, :]) - tp
    fp = sum(cm[:, i]) - tp
    tn = sum(sum(cm)) - tp - fn - fp
```

```
    fpr[i], tpr[i], _ = roc_curve([1 if j == i else 0 for j in ...
        y_test], [1 if j == i else 0 for j in y_pred])
    roc_auc[i] = auc(fpr[i], tpr[i])

    print(f"Class {i}: AUROC = {roc_auc[i]:.2f}")

# Plot ROC curves
plt.figure(figsize=(10, 8))
for i in range(n_classes):
    plt.plot(fpr[i], tpr[i], lw=2, label=f'Class {i} (AUROC = ...
        {roc_auc[i]:.2f})')

plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve for Multi-Class Classification')
plt.legend(loc='lower right')
plt.show()


## ANN Code
import numpy as np
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelBinarizer
from sklearn.metrics import accuracy_score, confusion_matrix

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

# One-hot encode the target variable
lb = LabelBinarizer()
y = lb.fit_transform(y)

# Split the dataset into the Training set and Test set
X_train, X_test, y_train, y_test = train_test_split(X, y, ...
    test_size=0.2, random_state=82)

# Feature Scaling to bring the variable in a single scale
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)

```

```
# Initialize the neural network parameters
def initialize(n_x, C1, C2):
    np.random.seed(10)
    W1 = np.random.randn(n_x, C1) * 0.1
    b1 = np.zeros((1, C1))
    W2 = np.random.randn(C1, C2) * 0.1
    b2 = np.zeros((1, C2))
    return W1, b1, W2, b2

# Sigmoid activation function
def sigmoid(Z):
    return 1 / (1 + np.exp(-Z))

# Softmax activation function
def softmax(Z):
    expZ = np.exp(Z - np.max(Z))
    A = expZ / expZ.sum(axis=1, keepdims=True)
    return A

# Forward propagation
def forward(W, X, b, activation=None):
    Z = np.dot(X, W) + b
    if activation == 'sigmoid':
        A = sigmoid(Z)
    elif activation == 'softmax':
        A = softmax(Z)
    else:
        A = Z
    return Z, A

def cost(A, Y):
    m = Y.shape[0]
    epsilon = 1e-15  # Small epsilon value to avoid log(0)
    logprobs = -np.log(A[np.arange(m), Y.argmax(axis=1)] + epsilon)
    cost = np.sum(logprobs) / m
    return cost

# Backward propagation
def backward(X, Y, A, Z, W, b, activation=None):
    m = X.shape[0]
    if activation == 'softmax':
        dZ = A - Y
        dW = np.dot(X.T, dZ) / m
        db = np.sum(dZ, axis=0, keepdims=True) / m
    elif activation == 'sigmoid':
        dZ = np.dot(cache[1],cache[0])*A*(1-A)
        dW = np.dot(X.T, dZ)
        db = np.sum(dZ, axis=0, keepdims=True)
    else:
```

```
        dZ = A
        dW = np.dot(X.T, dZ)
        db = np.sum(dZ, axis=0, keepdims=True)
        # You might want to implement derivatives for other ...
            activations here if needed
    return dW, db, dZ


# Update the parameters
def update(W, b, dW, db, learning_rate):
    W -= learning_rate * dW
    b -= learning_rate * db
    return W, b

# Train the neural network
def train(X_train, y_train, W1, b1, W2, b2, learning_rate, epochs):
    costs = []
    for epoch in range(epochs):
        # Forward propagation
        Z1, A1 = forward(W1, X_train, b1, 'sigmoid')
        Z2, A2 = forward(W2, A1, b2, 'softmax')

        # Compute the cost
        J = cost(A2, y_train)
        costs.append(J)

        # Backward propagation
        dW2, db2, _ = backward(A1, y_train, A2, Z2, W2, b2, ...
            'softmax')
        dW1, db1, _ = backward(X_train, y_train, A1, Z1, W1, b1)

        # Update parameters
        W1, b1 = update(W1, b1, dW1, db1, learning_rate)
        W2, b2 = update(W2, b2, dW2, db2, learning_rate)

        # Print the cost every 100 epochs
        if epoch % 100 == 0:
            print(f"Epoch {epoch}: Cost {J}")

    return W1, b1, W2, b2, costs

# Train the neural network
input_size = X_train.shape[1]
# hidden_layer_size1 = 72
hidden_layer_size1 = 102
hidden_layer_size2 = 3
# learning_rate = 0.000001
learning_rate = 0.0000000001
epochs = 1500
```

```

W1, b1, W2, b2 = initialize(input_size, hidden_layer_size1, ...
    hidden_layer_size2)
costs = train(X_train, y_train, W1, b1, W2, b2 , learning_rate, ...
    epochs)

# Predict using the trained model
def predict(X, W1, b1, W2, b2):
    _, A1 = forward(W1, X, b1)
    _, A2 = forward(W2, A1, b2, 'softmax')
    return np.argmax(A2, axis=1)

# Make predictions on the test set
y_pred = predict(X_test, W1, b1, W2, b2)

# Calculate accuracy and confusion matrix
accuracy = accuracy_score(np.argmax(y_test, axis=1), y_pred)
cfm = confusion_matrix(np.argmax(y_test, axis=1), y_pred)

print(f"Accuracy: {accuracy * 100}%")
print("Confusion Matrix:\n", cfm)
#finding accuracy from the confusion matrix.
a = cfm.shape
corrPred = 0
falsePred = 0

for row in range(a[0]):
    for c in range(a[1]):
        if row == c:
            corrPred +=cfm[row,c]
        else:
            falsePred += cfm[row,c]
print('Correct predictions: ', corrPred)
print('False predictions', falsePred)
ANNAccuracy = corrPred/(cfm.sum())
print ('Accuracy of the SVC Clasification is: ', ...
    corrPred/(cfm.sum()))
```