



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

1 Experiment-3 2

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 hyperparameter 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 hyperparameters 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

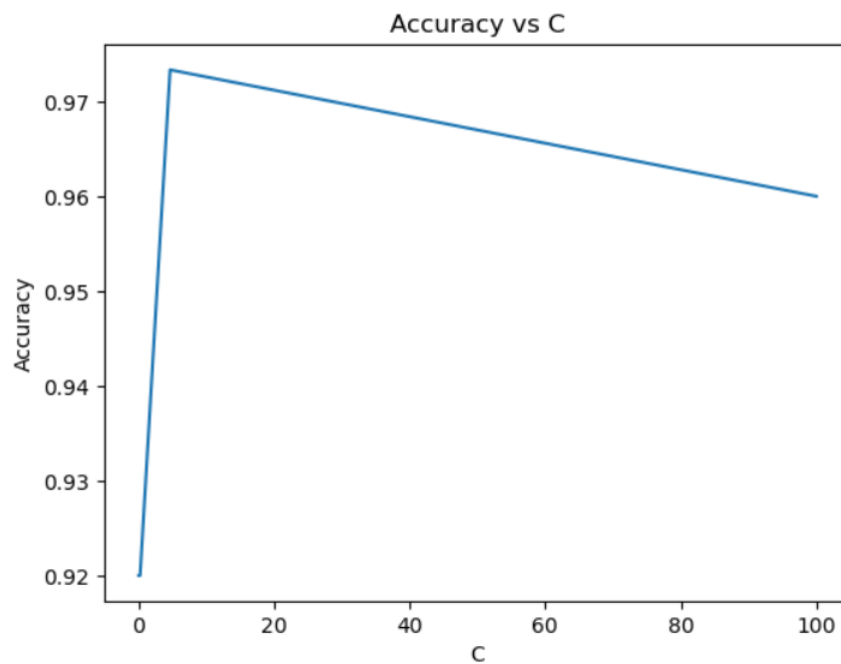


Figure 3: score vs C for RBF kernel

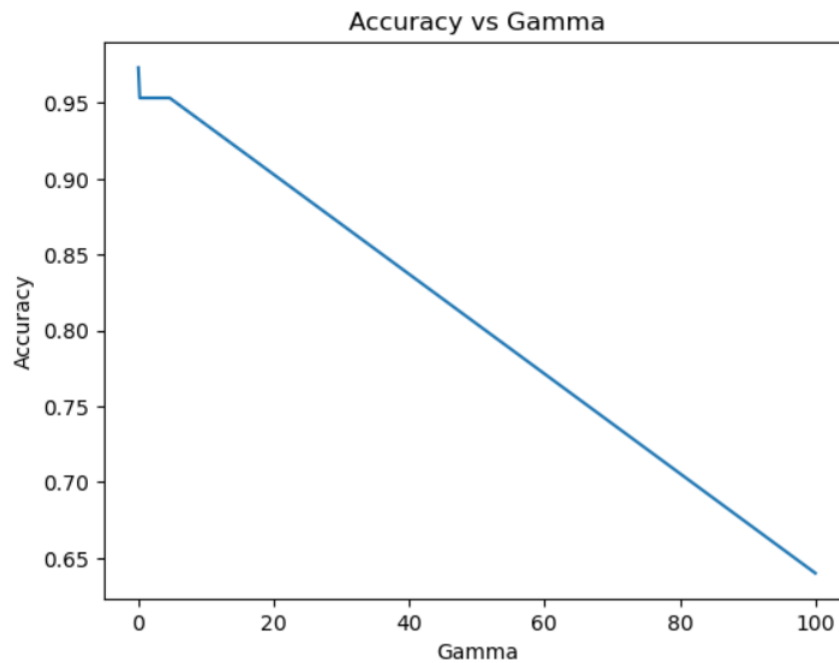


Figure 4: score vs gamma for RBF kernel

```
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

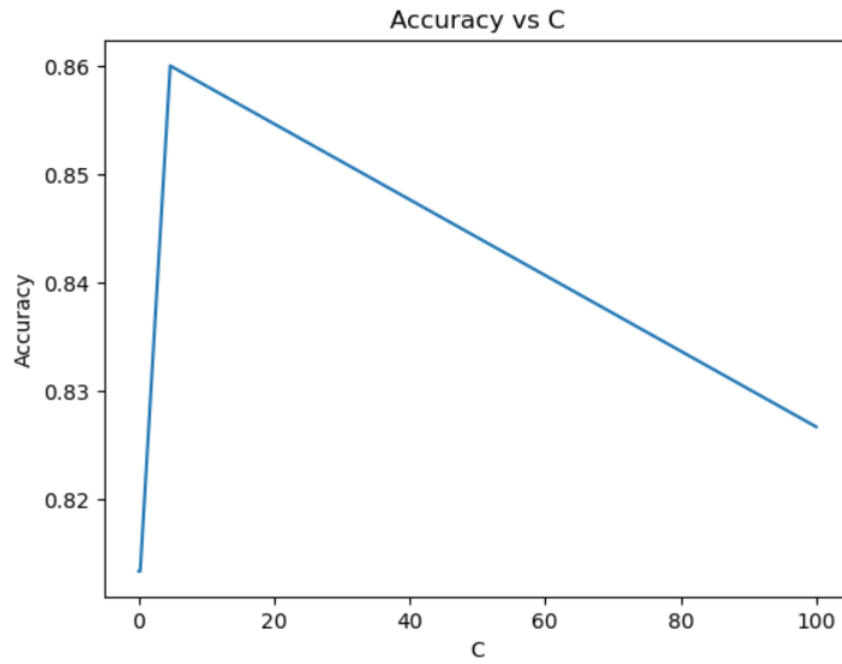


Figure 7: score vs C for Sigmoid kernel

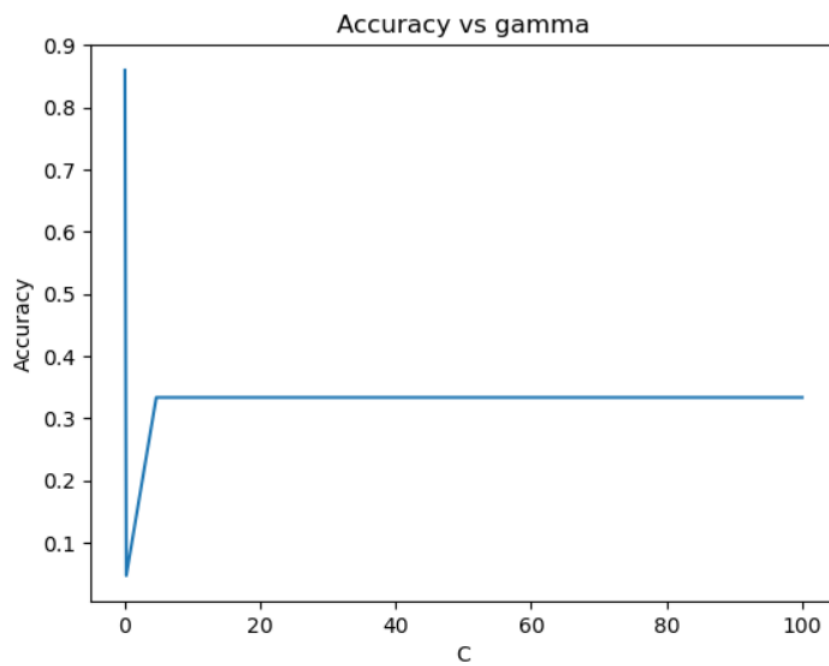


Figure 8: score vs gamma for Sigmoid kernel

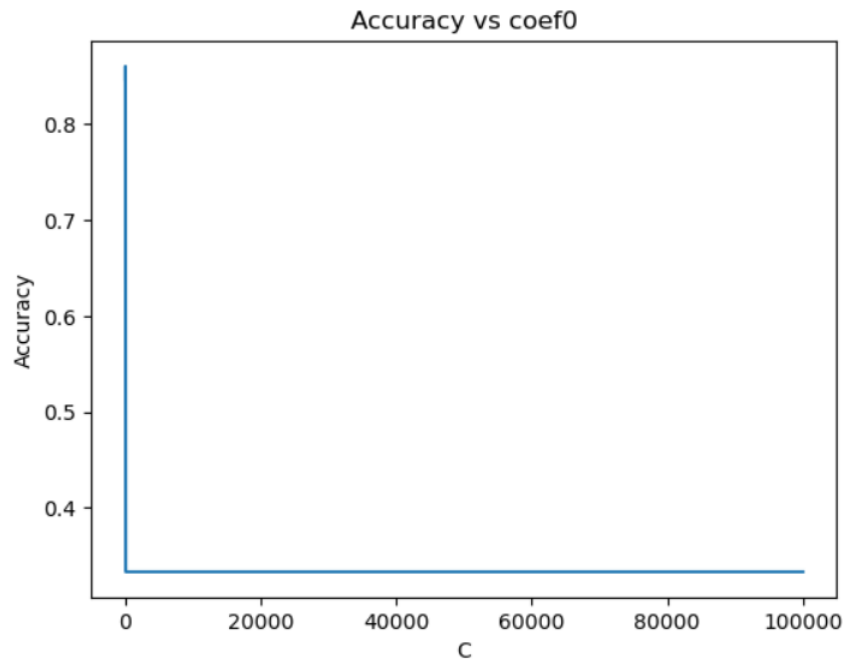


Figure 9: score vs coef for Sigmoid kernel

```
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 hyperparameter 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 0.95

Figure 11: Optimal values of hyperparameters for Polynomial kernel

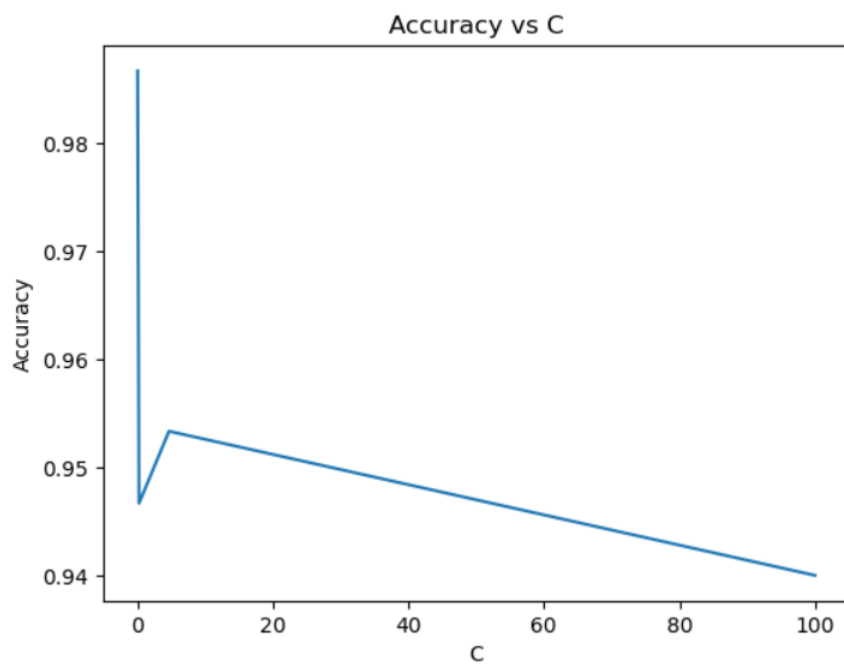


Figure 12: score vs C for Polynomial kernel

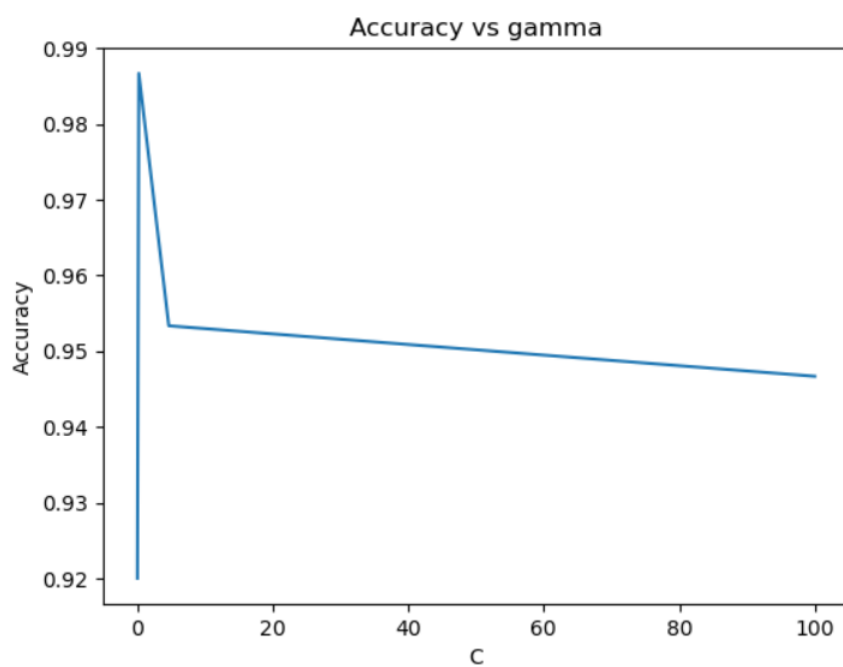


Figure 13: score vs gamma for Polynomial kernel

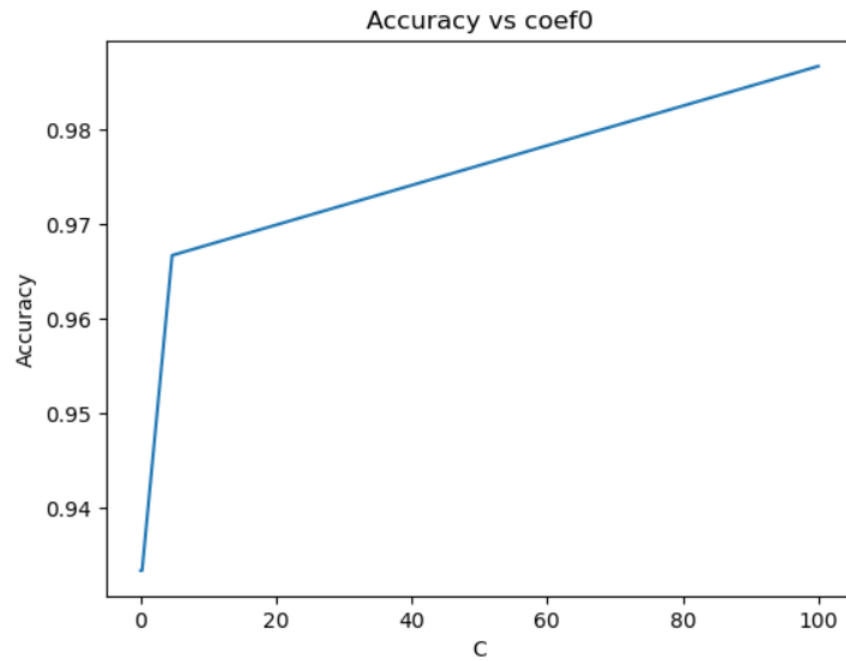


Figure 14: score vs coef for Polynomial kernel

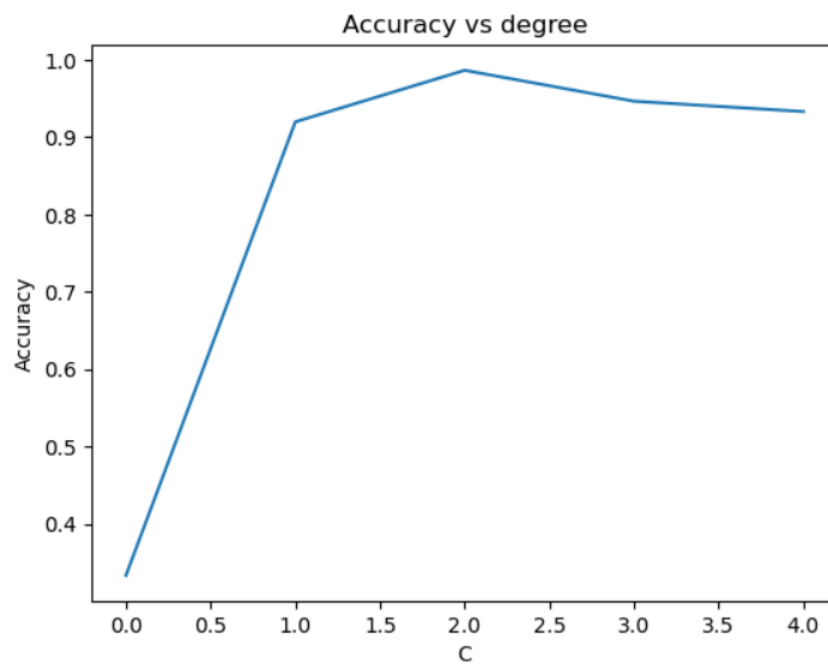


Figure 15: score vs degree for Polynomial kernel

```

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 classification 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 hyperparameters 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 parameter 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:

```
1  # SVC Classification
2
3  # Importing the libraries
4  import numpy as np
5  import matplotlib.pyplot as plt
6  import matplotlib.image as mpimg
7  import pandas as pd
8
9
10 # In[352]:
11
12
13 from sklearn.model_selection import GridSearchCV, ...
    StratifiedShuffleSplit
14 from sklearn.svm import SVC
15
16
17 # In[353]:
18
19
20 # Importing the dataset
21 dataset = pd.read_csv(r"C:\Users\Prajyot\Downloads\iris.csv")
22
23
24 # In[354]:
25
26
27 #looking at the first 5 values of the dataset
```

```
28 dataset.head()
29
30
31 # In[355]:
32
33
34 #Splitting the dataset in independent and dependent variables
35 X = dataset.iloc[:, :4].values
36 y = dataset['species'].values
37
38
39 # In[356]:
40
41
42 # Splitting the dataset into the Training set and Test set
43 from sklearn.model_selection import train_test_split
44 X_train, X_test, y_train, y_test = train_test_split(X, y, ...
    test_size = 0.20, random_state = 82)
45
46
47 # In[357]:
48
49
50 print("Training data size: ", X_train.shape)
51 print("Training label size: ", y_train.shape)
52 print("Testing data size: ", X_test.shape)
53 print("Testing label size: ", y_test.shape)
54
55
56 # In[358]:
57
58
59 # Feature Scaling to bring the variable in a single scale
60 from sklearn.preprocessing import StandardScaler
61 sc = StandardScaler()
62 X_train = sc.fit_transform(X_train)
63 X_test = sc.transform(X_test)
64
65
66 # # RBF Kernel
67
68 # In[359]:
69
70
71 from sklearn.model_selection import GridSearchCV, ...
    StratifiedShuffleSplit
72
73 C_range = np.logspace(-2, 2, 4)
74 gamma_range = np.logspace(-2, 2, 4)
```

```
75 param_grid = dict(gamma=gamma_range, C=C_range)
76 cv = StratifiedShuffleSplit(n_splits=5, test_size=0.2, ...
    random_state=42)
77 grid = GridSearchCV(SVC(), param_grid=param_grid, cv=cv)
78 grid.fit(X, y)
79
80 print(
81     "The best parameters are %s with a score of %0.2f"
82     % (grid.best_params_, grid.best_score_)
83 )
84
85
86 # In[360]:
87
88
89 # from sklearn.model_selection import GridSearchCV, ...
    StratifiedShuffleSplit
90
91 # C_range = np.logspace(-2, 2, 4)
92 # gamma_range = [0.01]
93 # param_grid = dict(gamma=gamma_range, C=C_range)
94 # cv = StratifiedShuffleSplit(n_splits=5, test_size=0.2, ...
    random_state=42)
95 # grid = GridSearchCV(SVC(), param_grid=param_grid, cv=cv)
96 # grid.fit(X, y)
97
98 # print(
99 #     "The best parameters are %s with a score of %0.2f"
100 #     % (grid.best_params_, grid.best_score_)
101 # )
102
103
104 # In[361]:
105
106
107 # print(grid.cv_results_.keys())
108
109
110 # In[362]:
111
112
113 # print(grid.cv_results_['param_C'])
114
115
116 # In[363]:
117
118
119 # # importing package
120 # import matplotlib.pyplot as plt
```

```
121 # x = grid.cv_results_['param_C']
122 # y = grid.cv_results_['mean_test_score']
123
124 # # plot line
125 # plt.plot(x, y)
126 # plt.title("Accuracy vs C")
127 # plt.xlabel("C")
128 # plt.ylabel("Accuracy")
129
130 # plt.show()
131
132
133 # In[364]:
134
135
136 svcclassifier = SVC(kernel = 'rbf', gamma = 0.01, C = ...
    4.6415888336127775, random.state = 0)
137 svcclassifier.fit(X_train, y_train)
138
139 # Predicting the Test set results
140 y_pred = svcclassifier.predict(X_test)
141 print(y_pred)
142
143
144 # In[365]:
145
146
147 # Making the Confusion Matrix
148 from sklearn.metrics import confusion_matrix
149 cm = confusion_matrix(y_test, y_pred)
150 print('Confusion Matrix:')
151 print(cm)
152
153 #finding accuracy from the confusion matrix.
154 a = cm.shape
155 corrPred = 0
156 falsePred = 0
157
158 for row in range(a[0]):
159     for c in range(a[1]):
160         if row == c:
161             corrPred +=cm[row,c]
162         else:
163             falsePred += cm[row,c]
164
165     n_classes = cm.shape[0]
166
167 for i in range(n_classes):
168     tp = cm[i, i]
```

```

169     fn = sum(cm[i, :]) - tp
170     fp = sum(cm[:, i]) - tp
171     tn = sum(sum(cm)) - tp - fn - fp
172
173     tpr = tp / (tp + fn)
174     tnr = tn / (tn + fp)
175
176     print(f"Class {i}: TPR = {tpr:.2f}, TNR = {tnr:.2f}")
177
178 print('Correct predictions: ', corrPred)
179 print('False predictions', falsePred)
180 kernelRbfAccuracy = corrPred/(cm.sum())
181 print ('Accuracy of the SVC Clasification is: ', ...
        corrPred/(cm.sum()))
182
183
184 # # Sigmoid Kernel
185
186 # In[366]:
187
188
189 # For the sigmoid kernel
190
191 C_range = np.logspace(-2, 2, 4)
192 gamma_range = np.logspace(-2, 2, 4)
193 coef0_range = np.logspace(-5, 5, 4)
194 param_grid = dict(gamma=gamma_range, C=C_range, coef0=coef0_range)
195 cv = StratifiedShuffleSplit(n_splits=5, test_size=0.2, ...
        random.state=42)
196 grid = GridSearchCV(SVC(kernel='sigmoid'), ...
        param_grid=param_grid, cv=cv)
197 grid.fit(X, y)
198
199 print(
200     "The best parameters are %s with a score of %0.2f"
201     % (grid.best_params_, grid.best_score_)
202 )
203
204
205 # In[367]:
206
207
208 # # For the sigmoid kernel
209
210 # C_range = [4.6415888336127775]
211 # coef0_range = [0.021544346900318846]
212 # gamma_range = np.logspace(-2, 2, 4)
213 # param_grid = dict(gamma=gamma_range, C=C_range, coef0=coef0_range)
214 # cv = StratifiedShuffleSplit(n_splits=5, test_size=0.2, ...

```



```
        random_state=42)
215 # grid = GridSearchCV(SVC(kernel='sigmoid'), ...
        param_grid=param_grid, cv=cv)
216 # grid.fit(X, y)
217
218 # print(
219 #     "The best parameters are %s with a score of %0.2f"
220 #     % (grid.best_params_, grid.best_score_)
221 # )
222
223
224 # In[368]:
225
226
227 # print(grid.cv_results_.keys())
228
229
230 # In[369]:
231
232
233 # print(grid.cv_results_['param-gamma'])
234
235
236 # In[370]:
237
238
239 # # importing package
240 # import matplotlib.pyplot as plt
241 # x = grid.cv_results_['param-gamma']
242 # y = grid.cv_results_['mean-test-score']
243
244 # # plot line
245 # plt.plot(x, y)
246 # plt.title("Accuracy vs gamma")
247 # plt.xlabel("C")
248 # plt.ylabel("Accuracy")
249
250 # plt.show()
251
252
253 # In[371]:
254
255
256 svcclassifier = SVC(kernel = 'sigmoid', C =4.6415888336127775 , ...
        coef0=0.021544346900318846 , gamma = 0.01, random_state = 0)
257 svcclassifier.fit(X_train, y_train)
258
259 # Predicting the Test set results
260 y_pred = svcclassifier.predict(X_test)
```

```
261 print(y_pred)
262
263
264 # In[372]:
265
266
267 # Making the Confusion Matrix
268 from sklearn.metrics import confusion_matrix
269 cm = confusion_matrix(y_test, y_pred)
270 print('Confusion Matrix:')
271 print(cm)
272
273 #finding accuracy from the confusion matrix.
274 a = cm.shape
275 corrPred = 0
276 falsePred = 0
277
278 for row in range(a[0]):
279     for c in range(a[1]):
280         if row == c:
281             corrPred +=cm[row,c]
282         else:
283             falsePred += cm[row,c]
284
285 n_classes = cm.shape[0]
286
287 for i in range(n_classes):
288     tp = cm[i, i]
289     fn = sum(cm[i, :]) - tp
290     fp = sum(cm[:, i]) - tp
291     tn = sum(sum(cm)) - tp - fn - fp
292
293     tpr = tp / (tp + fn)
294     tnr = tn / (tn + fp)
295
296     print(f"Class {i}: TPR = {tpr:.2f}, TNR = {tnr:.2f}")
297
298 print('Correct predictions: ', corrPred)
299 print('False predictions', falsePred)
300 kernelSigmoidAccuracy = corrPred/(cm.sum())
301 print ('Accuracy of the SVC Clasification is: ', ...
        corrPred/(cm.sum()))
302
303
304 # # Polynomial Kernel
305
306 # In[373]:
307
308
```

```
309 # For the polynomial kernel
310
311 C_range = np.logspace(-2, 2, 4)
312 gamma_range = np.logspace(-2, 2, 4)
313 coef0_range = np.logspace(-2, 2, 4)
314 degree_range = [0,1,2,3,4]
315 param_grid = dict(gamma=gamma_range, C=C_range, ...
                    coef0=coef0_range, degree=degree_range)
316 cv = StratifiedShuffleSplit(n_splits=5, test_size=0.2, ...
                             random_state=42)
317 grid = GridSearchCV(SVC(kernel='poly'), param_grid=param_grid, ...
                     cv=cv)
318 grid.fit(X, y)
319
320 print(
321     "The best parameters are %s with a score of %0.2f"
322     % (grid.best_params_, grid.best_score_)
323 )
324
325
326 # In[374]:
327
328
329 # # For the polynomial kernel
330
331 # C_range = [0.01]
332 # gamma_range = [0.21544346900318834]
333 # coef0_range = [100]
334 # degree_range = [0,1,2,3,4]
335 # param_grid = dict(gamma=gamma_range, C=C_range, ...
                    coef0=coef0_range, degree=degree_range)
336 # cv = StratifiedShuffleSplit(n_splits=5, test_size=0.2, ...
                             random_state=42)
337 # grid = GridSearchCV(SVC(kernel='poly'), param_grid=param_grid, ...
                     cv=cv)
338 # grid.fit(X, y)
339
340 # print(
341 #     "The best parameters are %s with a score of %0.2f"
342 #     % (grid.best_params_, grid.best_score_)
343 # )
344
345
346 # In[375]:
347
348
349 # print(grid.cv_results_.keys())
350
351
```

```
352 # In[376]:
353
354
355 # print(grid.cv_results_['param-degree'])
356
357
358 # In[377]:
359
360
361 # # importing package
362 # import matplotlib.pyplot as plt
363 # x = grid.cv_results_['param-degree']
364 # y = grid.cv_results_['mean-test-score']
365
366 # # plot line
367 # plt.plot(x, y)
368 # plt.title("Accuracy vs degree")
369 # plt.xlabel("C")
370 # plt.ylabel("Accuracy")
371
372 # plt.show()
373
374
375 # In[378]:
376
377
378 svcclassifier = SVC(kernel = 'poly', C = 0.01 , coef0 = ...
    100.0, degree=2, gamma = 0.21544346900318834, random_state = 0)
379 svcclassifier.fit(X_train, y_train)
380
381 # Predicting the Test set results
382 y_pred = svcclassifier.predict(X_test)
383 print(y_pred)
384
385
386 # In[379]:
387
388
389 # Making the Confusion Matrix
390 from sklearn.metrics import confusion_matrix
391 cm = confusion_matrix(y_test, y_pred)
392 print('Confusion Matrix:')
393 print(cm)
394
395 #finding accuracy from the confusion matrix.
396 a = cm.shape
397 corrPred = 0
398 falsePred = 0
399
```

```
400 for row in range(a[0]):
401     for c in range(a[1]):
402         if row == c:
403             corrPred +=cm[row,c]
404         else:
405             falsePred += cm[row,c]
406
407 n_classes = cm.shape[0]
408
409 for i in range(n_classes):
410     tp = cm[i, i]
411     fn = sum(cm[i, :]) - tp
412     fp = sum(cm[:, i]) - tp
413     tn = sum(sum(cm)) - tp - fn - fp
414
415     tpr = tp / (tp + fn)
416     tnr = tn / (tn + fp)
417
418     print(f"Class {i}: TPR = {tpr:.2f}, TNR = {tnr:.2f}")
419
420 print('Correct predictions: ', corrPred)
421 print('False predictions', falsePred)
422 kernelPolyAccuracy = corrPred/(cm.sum())
423 print ('Accuracy of the SVC Clasification is: ', ...
         corrPred/(cm.sum()))
424
425
426 # In[332]:
427
428
429 from sklearn.metrics import confusion_matrix, roc_curve, auc
430 import matplotlib.pyplot as plt
431
432 # Making the Confusion Matrix
433 cm = confusion_matrix(y_test, y_pred)
434 print(cm)
435
436 n_classes = cm.shape[0]
437
438 # Calculate ROC and AUROC for each class
439 fpr = dict()
440 tpr = dict()
441 roc_auc = dict()
442
443 for i in range(n_classes):
444     tp = cm[i, i]
445     fn = sum(cm[i, :]) - tp
446     fp = sum(cm[:, i]) - tp
447     tn = sum(sum(cm)) - tp - fn - fp
```

```
448
449     fpr[i], tpr[i], _ = roc_curve([1 if j == i else 0 for j in ...
450                                   y_test], [1 if j == i else 0 for j in y_pred])
451     roc_auc[i] = auc(fpr[i], tpr[i])
452
453     print(f"Class {i}: AUROC = {roc_auc[i]:.2f}")
454
455 # Plot ROC curves
456 plt.figure(figsize=(10, 8))
457 for i in range(n_classes):
458     plt.plot(fpr[i], tpr[i], lw=2, label=f'Class {i} (AUROC = ...
459             {roc_auc[i]:.2f})')
460
461 plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
462 plt.xlim([0.0, 1.0])
463 plt.ylim([0.0, 1.05])
464 plt.xlabel('False Positive Rate')
465 plt.ylabel('True Positive Rate')
466 plt.title('ROC Curve for Multi-Class Classification')
467 plt.legend(loc='lower right')
468 plt.show()
469
470 ## ANN Code
471 import numpy as np
472 from sklearn import datasets
473 from sklearn.model_selection import train_test_split
474 from sklearn.preprocessing import LabelBinarizer
475 from sklearn.metrics import accuracy_score, confusion_matrix
476
477 # Load the Iris dataset
478 iris = datasets.load_iris()
479 X = iris.data
480 y = iris.target
481
482 # One-hot encode the target variable
483 lb = LabelBinarizer()
484 y = lb.fit_transform(y)
485
486 # Split the dataset into the Training set and Test set
487 X_train, X_test, y_train, y_test = train_test_split(X, y, ...
488                                                     test_size=0.2, random_state=82)
489
490 # Feature Scaling to bring the variable in a single scale
491 from sklearn.preprocessing import StandardScaler
492 sc = StandardScaler()
493 X_train = sc.fit_transform(X_train)
494 X_test = sc.transform(X_test)
```

```

494 # Initialize the neural network parameters
495 def initialize(n_x, C1, C2):
496     np.random.seed(10)
497     W1 = np.random.randn(n_x, C1) * 0.1
498     b1 = np.zeros((1, C1))
499     W2 = np.random.randn(C1, C2) * 0.1
500     b2 = np.zeros((1, C2))
501     return W1, b1, W2, b2
502
503 # Sigmoid activation function
504 def sigmoid(Z):
505     return 1 / (1 + np.exp(-Z))
506
507 # Softmax activation function
508 def softmax(Z):
509     expZ = np.exp(Z - np.max(Z))
510     A = expZ / expZ.sum(axis=1, keepdims=True)
511     return A
512
513 # Forward propagation
514 def forward(W, X, b, activation=None):
515     Z = np.dot(X, W) + b
516     if activation == 'sigmoid':
517         A = sigmoid(Z)
518     elif activation == 'softmax':
519         A = softmax(Z)
520     else:
521         A = Z
522     return Z, A
523
524 def cost(A, Y):
525     m = Y.shape[0]
526     epsilon = 1e-15 # Small epsilon value to avoid log(0)
527     logprobs = -np.log(A[np.arange(m), Y.argmax(axis=1)] + epsilon)
528     cost = np.sum(logprobs) / m
529     return cost
530
531 # Backward propagation
532 def backward(X, Y, A, Z, W, b, activation=None):
533     m = X.shape[0]
534     if activation == 'softmax':
535         dZ = A - Y
536         dW = np.dot(X.T, dZ) / m
537         db = np.sum(dZ, axis=0, keepdims=True) / m
538     elif activation == 'sigmoid':
539         dZ = np.dot(cache[1], cache[0]) * A * (1 - A)
540         dW = np.dot(X.T, dZ)
541         db = np.sum(dZ, axis=0, keepdims=True)
542     else:

```

```

543         dZ = A
544         dW = np.dot(X.T, dZ)
545         db = np.sum(dZ, axis=0, keepdims=True)
546         # You might want to implement derivatives for other ...
           activations here if needed
547     return dW, db, dZ
548
549
550 # Update the parameters
551 def update(W, b, dW, db, learning_rate):
552     W -= learning_rate * dW
553     b -= learning_rate * db
554     return W, b
555
556 # Train the neural network
557 def train(X_train, y_train, W1, b1, W2, b2, learning_rate, epochs):
558     costs = []
559     for epoch in range(epochs):
560         # Forward propagation
561         Z1, A1 = forward(W1, X_train, b1, 'sigmoid')
562         Z2, A2 = forward(W2, A1, b2, 'softmax')
563
564         # Compute the cost
565         J = cost(A2, y_train)
566         costs.append(J)
567
568         # Backward propagation
569         dW2, db2, _ = backward(A1, y_train, A2, Z2, W2, b2, ...
           'softmax')
570         dW1, db1, _ = backward(X_train, y_train, A1, Z1, W1, b1)
571
572         # Update parameters
573         W1, b1 = update(W1, b1, dW1, db1, learning_rate)
574         W2, b2 = update(W2, b2, dW2, db2, learning_rate)
575
576         # Print the cost every 100 epochs
577         if epoch % 100 == 0:
578             print(f"Epoch {epoch}: Cost {J}")
579
580     return W1, b1, W2, b2, costs
581
582 # Train the neural network
583 input_size = X_train.shape[1]
584 # hidden_layer_size1 = 72
585 hidden_layer_size1 = 102
586 hidden_layer_size2 = 3
587 # learning_rate = 0.000001
588 learning_rate = 0.0000000001
589 epochs = 1500

```



```
590
591 W1, b1, W2, b2 = initialize(input_size, hidden_layer_size1, ...
    hidden_layer_size2)
592 costs = train(X_train, y_train, W1, b1, W2, b2 , learning_rate, ...
    epochs)
593
594 # Predict using the trained model
595 def predict(X, W1, b1, W2, b2):
596     _, A1 = forward(W1, X, b1)
597     _, A2 = forward(W2, A1, b2, 'softmax')
598     return np.argmax(A2, axis=1)
599
600 # Make predictions on the test set
601 y_pred = predict(X_test, W1, b1, W2, b2)
602
603 # Calculate accuracy and confusion matrix
604 accuracy = accuracy_score(np.argmax(y_test, axis=1), y_pred)
605 cfm = confusion_matrix(np.argmax(y_test, axis=1), y_pred)
606
607 print(f"Accuracy: {accuracy * 100}%")
608 print("Confusion Matrix:\n", cfm)
609 #finding accuracy from the confusion matrix.
610 a = cfm.shape
611 corrPred = 0
612 falsePred = 0
613
614 for row in range(a[0]):
615     for c in range(a[1]):
616         if row == c:
617             corrPred += cfm[row,c]
618         else:
619             falsePred += cfm[row,c]
620 print('Correct predictions: ', corrPred)
621 print('False predictions', falsePred)
622 ANNAccuracy = corrPred/(cfm.sum())
623 print ('Accuracy of the SVC Clasification is: ', ...
    corrPred/(cfm.sum()))
```