Shri Vile Parle Kelavani Mandal's
DWARKADAS J. SANGHVI COLLEGE OF ENGINEERING
(Autonomous College Affiliated to the University of Mumbai)
NAAC Accredited with "A" Grade (CGPA : 3.18)

Department of Computer Science and Engineering (Data Science)

Subject: Machine Learning – I (DJ19DSC402)

AY: 2021-22

Experiment 8

(SVM)

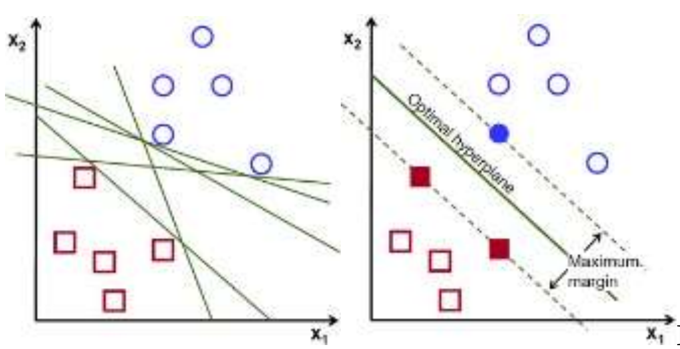
Name: Dev Patel **SAP ID: 60009200016**

Batch: K/K1 **Date: 23/05/2022**

Aim: Perform SVM using soft margin SVC, Kernels and improve the accuracies using hyperparameter tuning.

Theory:

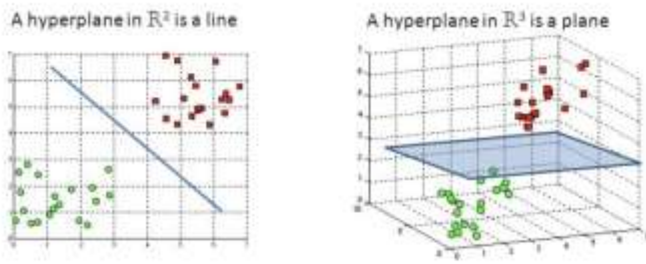
The objective of the support vector machine algorithm is to find a hyperplane in an N-dimensional space (N — the number of features) that distinctly classifies the data points.



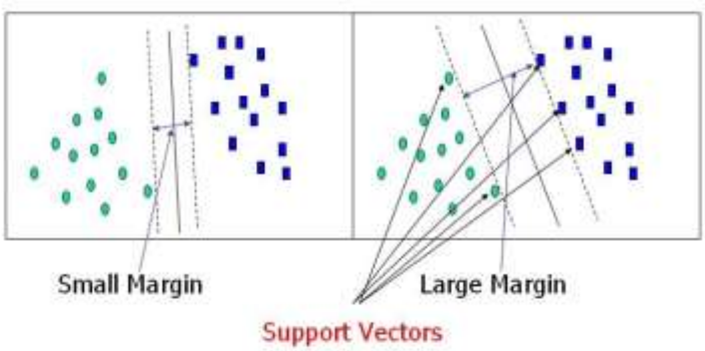
Possible hyperplanes

To separate the two classes of data points, there are many possible hyperplanes that could be chosen. Our objective is to find a plane that has the maximum margin, i.e the maximum distance between data points of both classes. Maximizing the margin distance provides some reinforcement so that future data points can be classified with more confidence.

Hyperplanes and Support Vectors



Hyperplanes in 2D and 3D feature space: Hyperplanes are decision boundaries that help classify the data points. Data points falling on either side of the hyperplane can be attributed to different classes. Also, the dimension of the hyperplane depends upon the number of features. If the number of input features is 2, then the hyperplane is just a line. If the number of input features is 3, then the hyperplane becomes a two-dimensional plane. It becomes difficult to imagine when the number of features exceeds 3.



Support vectors are data points that are closer to the hyperplane and influence the position and orientation of the hyperplane. Using these support vectors, we maximize the margin of the classifier. Deleting the support vectors will change the position of the hyperplane. These are the points that help us build our SVM.

Large Margin Intuition
In logistic regression, we take the output of the linear function and squash the value within the range of [0,1] using the sigmoid function. If the squashed value is greater than a threshold value(0.5) we assign it a label 1, else we assign it a label 0. In SVM, we take the output of the linear function and if that output is greater than 1, we identify it with one class and if the output is -1, we identify is with another class. Since the threshold values are changed to 1 and -1 in SVM, we obtain this reinforcement range of values([-1,1]) which acts as margin.

Cost Function and Gradient Update: In the SVM algorithm, we are looking to maximize the margin between the data points and the hyperplane. The loss function that helps maximize the margin is hinge loss.

$$c(x, y, f(x)) = \begin{cases} 0, & \text{if } y * f(x) \geq 1 \\ 1 - y * f(x), & \text{else} \end{cases}$$

$$c(x, y, f(x)) = (1 - y * f(x))_+$$

Hinge loss function (function on left can be represented as a function on the right)
The cost is 0 if the predicted value and the actual value are of the same sign. If they are not, we then calculate the loss value. We also add a regularization parameter the cost function. The objective of the regularization parameter is to balance the margin maximization and loss. After adding the regularization parameter, the cost functions looks as below.

$$min_w \lambda \parallel w \parallel^2 + \sum_{i=1}^{n} (1 - y_i \langle x_i, w \rangle)_+$$

Loss function for SVM
Now that we have the loss function, we take partial derivatives with respect to the weights to find the gradients. Using the gradients, we can update our weights.

$$\frac{\delta}{\delta w_k} \lambda \parallel w \parallel^2 \quad = 2\lambda w_k$$

$$\frac{\delta}{\delta w_k} \left(1 - y_i \langle x_i, w \rangle\right)_+ \quad = \begin{cases} 0, & \text{if } y_i \langle x_i, w \rangle \geq 1 \\ -y_i x_{ik}, & \text{else} \end{cases}$$

Gradients
When there is no misclassification, i.e our model correctly predicts the class of our data point, we only have to update the gradient from the regularization parameter.

$$w = w - \alpha \cdot (2\lambda w)$$

Gradient Update — No misclassification
When there is a misclassification, i.e our model make a mistake on the prediction of the class of our data point, we include the loss along with the regularization parameter to perform gradient update.

$$w = w + \alpha \cdot (y_i \cdot x_i - 2\lambda w)$$

Gradient Update — Misclassification

Lab Assignments to complete in this session:

Use the given dataset and perform the following tasks:

Dataset 1: IRIS.csv

Dataset 3: mnist_784: The MNIST database of handwritten digits with 784 features, raw data available at: http://yann.lecun.com/exdb/mnist/. It can be split in a training set of the first 60,000 examples, and a test set of 10,000 examples. It is a subset of a larger set available from NIST. The digits have been size normalized and cantered in a fixed-size image. It is a good database for people who want to try learning techniques and pattern recognition methods on real-world data while spending minimal efforts on pre-processing and formatting. The original black and white (bilevel) images from NIST were size normalized to fit in a 20x20 pixel box while preserving their aspect ratio. The resulting images contain grey levels as a result of the anti-aliasing technique used by the normalization algorithm. the images were centred in a 28x28 image by computing the centre of mass of the pixels, and translating the image so as to position this point at the centre of the 28x28 field.

Task 1: Build a linear classifier on Dataset 1 using SVC.

Task 2: Build a classifier on Dataset 1 using Linear, Polynomial and RBF kernel and show the decision boundary using matplotlib.

Task 3. Find the accuracy of svc classifier (M1) built on Dataset 3 using linear SVC and RBF kernel.

Task 4: Improve the accuracy of M1 by varying C and gamma values and using RandomizedSearchCV.

Task 5: Calculate the computational time of Task 3 and 4.

Code and Output:

Name: Dev Patel

SAP ID: 60009200016

In [1]:

```
from sklearn import datasets
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn import svm
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report,accuracy_score
```

Task 1 & 2: Build a linear classifier on Iris Dataset using SVC. Also use Polynomial and RBF kernel and show the decision boundary using matplotlib.

In [ ]:

```
df = datasets.load_iris()
```

In [3]:

```
X= df.data[:,:2]
y = df["target"]
```

In [4]:

```
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.3,random_state=42)
```

In [5]:

```
#Train SVMs with different kernels
svc = svm.SVC(kernel='linear').fit(X_train, y_train)
rbf_svc = svm.SVC(kernel='rbf', gamma=0.7).fit(X_train, y_train)
poly_svc = svm.SVC(kernel='poly', degree=3).fit(X_train, y_train)
```

In [6]:

```
pred_lin = svc.predict(X_test)
pred_poly = poly_svc.predict(X_test)
pred_rbf = rbf_svc.predict(X_test)

trpred_lin = svc.predict(X_train)
trpred_poly = poly_svc.predict(X_train)
trpred_rbf = rbf_svc.predict(X_train)
```

In [7]:

```
tr_acc_lin = accuracy_score(y_train,trpred_lin)
acc_lin = accuracy_score(y_test,pred_lin)
tr_acc_poly = accuracy_score(y_train,trpred_poly)
acc_poly = accuracy_score(y_test,pred_poly)
tr_acc_rbf = accuracy_score(y_train,trpred_rbf)
acc_rbf = accuracy_score(y_test,pred_rbf)
```

Linear SVC

In [8]:

```
print("Linear")
print("On Train test: ",tr_acc_lin)
```

```
print("On Test test: ",acc_lin)
```

```
Linear
On Train test:  0.819047619047619
On Test test:  0.8
```

In [9]:

```
from sklearn.metrics import classification_report
print("Linear SVC")
print("ON TRAINING SET\n")
print(classification_report(y_train, trpred_lin, target_names=df.target_names))
print("ON TEST SET\n")
print(classification_report(y_test, pred_lin, target_names=df.target_names))
```

```
Linear SVC
ON TRAINING SET

              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00        31
  versicolor       0.71      0.81      0.76        37
   virginica       0.78      0.68      0.72        37

    accuracy                           0.82       105
   macro avg       0.83      0.83      0.83       105
weighted avg       0.82      0.82      0.82       105

ON TEST SET

              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00        19
  versicolor       0.70      0.54      0.61        13
   virginica       0.62      0.77      0.69        13

    accuracy                           0.80        45
   macro avg       0.78      0.77      0.77        45
weighted avg       0.81      0.80      0.80        45
```

Polynomial SVC

In [10]:

```
print("Polynomial SVC")
print("On Train test: ",tr_acc_poly)
print("On Test test: ",acc_poly)
```

```
Polynomial SVC
On Train test:  0.8095238095238095
On Test test:  0.7333333333333333
```

In [11]:

```
print("Polynomial SVC")
print("ON TRAINING SET\n")
print(classification_report(y_train, trpred_poly, target_names=df.target_names))
print("ON TEST SET\n")
print(classification_report(y_test, pred_poly, target_names=df.target_names))
```

```
Polynomial SVC
ON TRAINING SET

              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00        31
  versicolor       0.70      0.81      0.75        37
   virginica       0.77      0.65      0.71        37

    accuracy                           0.81       105
   macro avg       0.82      0.82      0.82       105
```

```
    macro avg       0.82      0.82      0.82       105
weighted avg       0.81      0.81      0.81       105

ON TEST SET

              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00        19
  versicolor       0.54      0.54      0.54        13
   virginica       0.54      0.54      0.54        13

    accuracy                           0.73        45
   macro avg       0.69      0.69      0.69        45
weighted avg       0.73      0.73      0.73        45
```

RBF Kernel

In [12]:

```python
print("RBF Kernel")
print("On Train test: ",tr_acc_rbf)
print("On Test test: ",acc_rbf)
```

```
RBF Kernel
On Train test:  0.8285714285714286
On Test test:  0.8
```

In [13]:

```python
print("RBF kernel")
print("ON TRAINING SET\n")
print(classification_report(y_train, trpred_rbf, target_names=df.target_names))
print("ON TEST SET\n")
print(classification_report(y_test, pred_rbf, target_names=df.target_names))
```

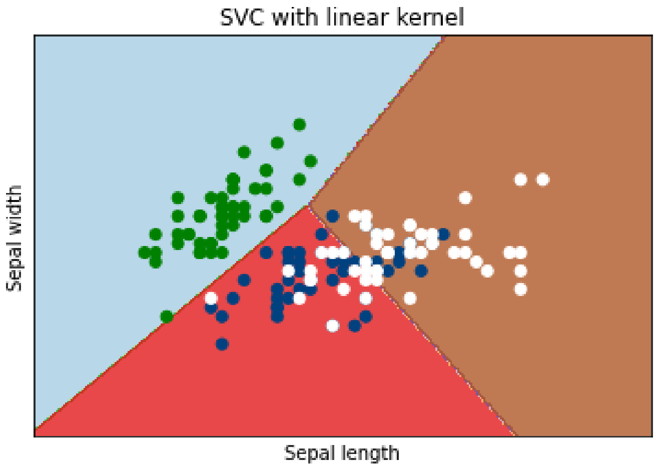
```
RBF kernel
ON TRAINING SET

              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00        31
  versicolor       0.73      0.81      0.77        37
   virginica       0.79      0.70      0.74        37

    accuracy                           0.83       105
   macro avg       0.84      0.84      0.84       105
weighted avg       0.83      0.83      0.83       105

ON TEST SET

              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00        19
  versicolor       0.70      0.54      0.61        13
   virginica       0.62      0.77      0.69        13

    accuracy                           0.80        45
   macro avg       0.78      0.77      0.77        45
weighted avg       0.81      0.80      0.80        45
```

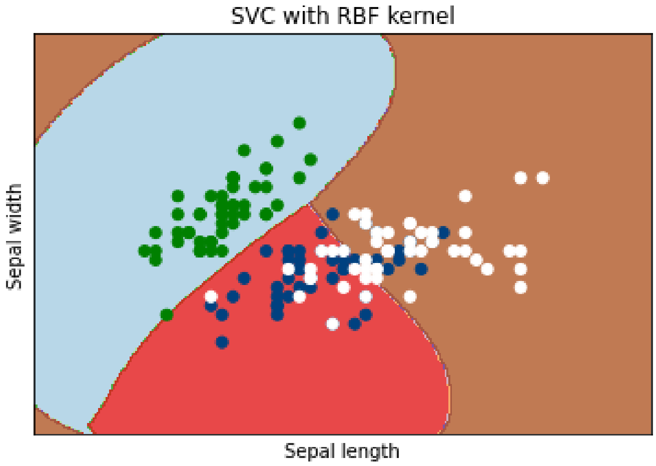
Decisio Boundary Using Matplotlib

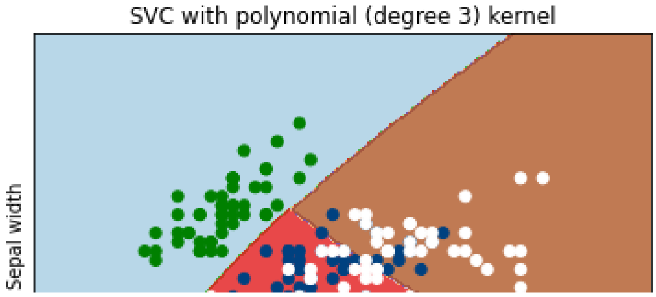
In [14]:

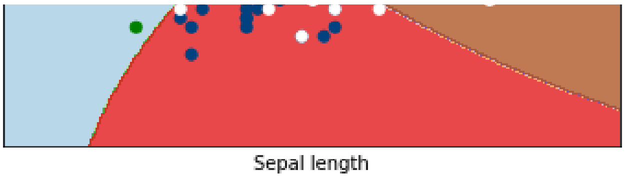
```python
#Create a mesh to plot in
h = .02 # step size in the mesh
x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
xx, yy = np.meshgrid(np.arange(x_min, x_max, h),
```

```
np.arange(y_min, y_max, h))

#Define title for the plots
titles = ['SVC with linear kernel',
'SVC with RBF kernel',
'SVC with polynomial (degree 3) kernel']

for i, clf in enumerate((svc, rbf_svc, poly_svc)):
    # Plot the decision boundary. For that, we will assign a color to each
    # point in the mesh [x_min, m_max]x[y_min, y_max].
    plt.figure(i)

    Z = clf.predict(np.c_[xx.ravel(), yy.ravel()])

    # Put the result into a color plot
    Z = Z.reshape(xx.shape)
    plt.contourf(xx, yy, Z, cmap=plt.cm.Paired, alpha=0.8)

    # Plot also the training points
    plt.scatter(X[:, 0], X[:, 1], c=y, cmap=plt.cm.ocean)
    plt.xlabel('Sepal length')
    plt.ylabel('Sepal width')
    plt.xlim(xx.min(), xx.max())
    plt.ylim(yy.min(), yy.max())
    plt.xticks(())
    plt.yticks(())
    plt.title(titles[i])

plt.show()
```









In [15]:

```
softsvc = svm.LinearSVC(C=1.0, class_weight=None, dual=True, fit_intercept=True,
    intercept_scaling=1, loss='squared_hinge', max_iter=1000,
    multi_class='ovr', penalty='l2', random_state=None, tol=0.0001,
    verbose=0)

softsvc.fit(X_train,y_train)
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/svm/_base.py:1208: ConvergenceWarning: Lib
linear failed to converge, increase the number of iterations.
  ConvergenceWarning,
```

Out[15]:

```
LinearSVC()
```

In [16]:

```
trpred = softsvc.predict(X_train)
pred = softsvc.predict(X_test)
```

In [ ]:

```
from sklearn.model_selection import RepeatedStratifiedKFold, cross_val_score

cv = RepeatedStratifiedKFold(n_splits=5, n_repeats=3, random_state=1)
y_scores = cross_val_score(softsvc, X, y, scoring='accuracy', cv=cv, n_jobs=1, error_sco
re='raise')
```

In [18]:

```
print(f'Accuracy: {np.mean(y_scores)}, STD of all accuracies: {np.std(y_scores)}')
```

```
Accuracy: 0.7888888888888889, STD of all accuracies: 0.07950463919999255
```

In [19]:

```
print("ON TRAINING SET\n")
print(classification_report(y_train, trpred, target_names=df.target_names))
print("ON TEST SET\n")
print(classification_report(y_test, pred, target_names=df.target_names))
```

```
ON TRAINING SET

              precision    recall  f1-score   support

      setosa       0.97      0.97      0.97        31
  versicolor       0.74      0.62      0.68        37
   virginica       0.70      0.81      0.75        37

    accuracy                           0.79       105
   macro avg       0.80      0.80      0.80       105
weighted avg       0.79      0.79      0.79       105

ON TEST SET

              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00        19
  versicolor       0.71      0.38      0.50        13
   virginica       0.58      0.85      0.69        13

    accuracy                           0.78        45
   macro avg       0.76      0.74      0.73        45
```

```
weighted avg       0.80      0.78      0.77        45
```

ROC Curve

In [20]:

```python
from sklearn.multiclass import OneVsRestClassifier
from sklearn.preprocessing import label_binarize
from sklearn.metrics import roc_curve, auc

y = label_binarize(y, classes=[0,1,2])
n_classes = 3
```

In [21]:

```python
# shuffle and split training and test sets
X_train, X_test, y_train, y_test =  train_test_split(X, y, test_size=0.33, random_state=
0)

# classifier
clf = OneVsRestClassifier(svm.LinearSVC(random_state=0))
y_score = clf.fit(X_train, y_train).decision_function(X_test)
```

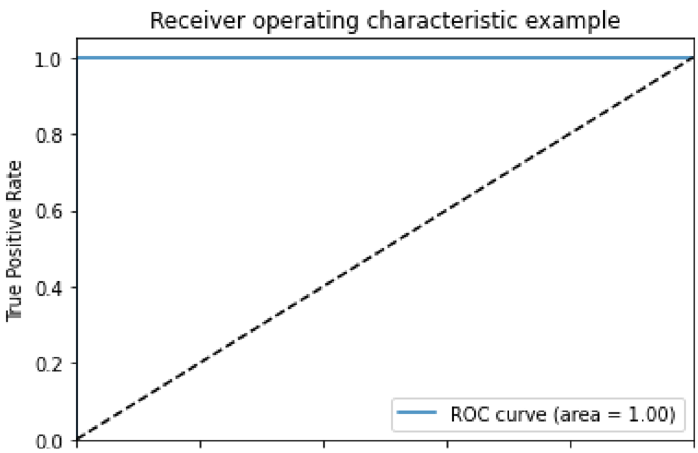
```
/usr/local/lib/python3.7/dist-packages/sklearn/svm/_base.py:1208: ConvergenceWarning: Lib
linear failed to converge, increase the number of iterations.
  ConvergenceWarning,
/usr/local/lib/python3.7/dist-packages/sklearn/svm/_base.py:1208: ConvergenceWarning: Lib
linear failed to converge, increase the number of iterations.
  ConvergenceWarning,
```

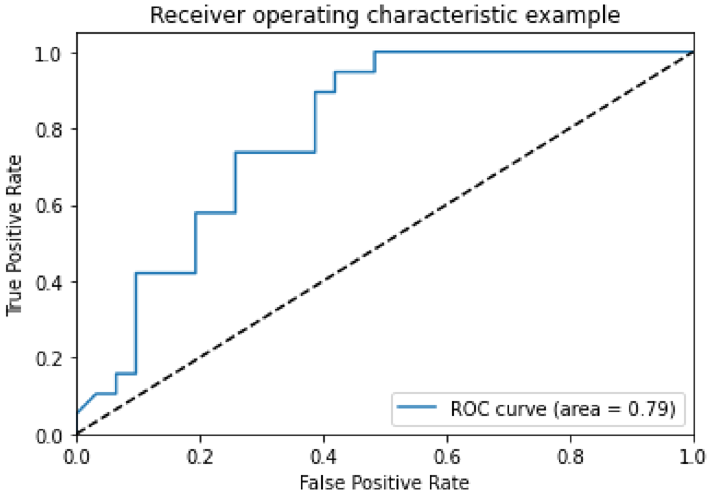
In [22]:

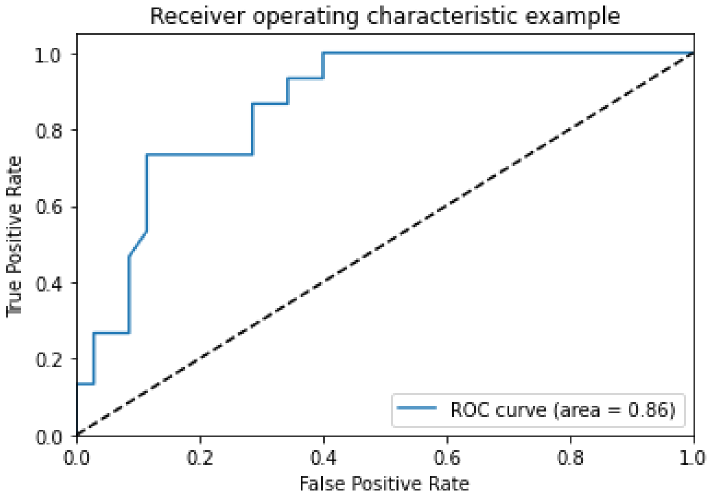
```python
# Compute ROC curve and ROC area for each class
fpr = dict()
tpr = dict()
roc_auc = dict()
for i in range(n_classes):
    fpr[i], tpr[i], _ = roc_curve(y_test[:, i], y_score[:, i])
    roc_auc[i] = auc(fpr[i], tpr[i])
```

In [23]:

```python
# Plot of a ROC curve for a specific class
for i in range(n_classes):
    plt.figure()
    plt.plot(fpr[i], tpr[i], label='ROC curve (area = %0.2f)' % roc_auc[i])
    plt.plot([0, 1], [0, 1], 'k--')
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('Receiver operating characteristic example')
    plt.legend(loc="lower right")
    plt.show()
```







Task 3. Find the accuracy of SVC classifier (M1) built on Dataset 3 using linear SVC and RBF kernel.

In [ ]:

```
#import libraries and load the dataset
import warnings
import matplotlib.pyplot as plt
from sklearn.datasets import fetch_openml
from sklearn.exceptions import ConvergenceWarning
from sklearn.model_selection import train_test_split
import time

# Load data from https://www.openml.org/d/554
X, y = fetch_openml("mnist_784", version=1, return_X_y=True, as_frame=False)
X_train = X[:60000]
y_train = y[:60000]
X_test = X[60000:]
y_test = y[60000:]
```

In [ ]:

```
#perform linear svc with default values
from sklearn.svm import LinearSVC
start = time.time()
lin_clf = LinearSVC(random_state=42)
lin_clf.fit(X_train, y_train)
lin_time = time.time()-start
from sklearn.metrics import accuracy_score
y_pred = lin_clf.predict(X_train)
accuracy_score(y_train, y_pred)
```

```
D:\Anaconda\lib\site-packages\sklearn\svm\_base.py:1225: ConvergenceWarning: Liblinear fa
iled to converge, increase the number of iterations.
  warnings.warn(
```

Out[ ]:

```
0.8348666666666666
```

Accuracy using Linear SVC is 83.49%

In [ ]:

```
import numpy as np
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train.astype(np.float32))
X_test_scaled = scaler.transform(X_test.astype(np.float32))
start = time.time()
lin_clf = LinearSVC(random_state=42)
lin_clf.fit(X_train_scaled, y_train)
lin_scal_time = time.time() - start
y_pred = lin_clf.predict(X_train_scaled)
accuracy_score(y_train, y_pred)
```

```
D:\Anaconda\lib\site-packages\sklearn\svm\_base.py:1225: ConvergenceWarning: Liblinear fa
iled to converge, increase the number of iterations.
  warnings.warn(
```

Out[ ]:

```
0.9214
```

Accuracy using Linear SVC after feature scaling is 92.14%

In [ ]:

```
from __future__ import division, print_function
import numpy as np
from sklearn import datasets, svm
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
%matplotlib inline
```

In [ ]:

```
from sklearn.svm import SVC
start = time.time()
svm_clf = SVC(kernel='rbf', gamma='scale')
svm_clf.fit(X_train_scaled[:10000], y_train[:10000]) # We use an SVC with an RBF kernel
rbf_time = time.time()- start
y_pred = svm_clf.predict(X_train_scaled)
accuracy_score(y_train, y_pred)
```

Out[ ]:

```
0.9455333333333333
```

Accuracy using RBF Kernel is 94.55%

Task 4: Improve the accuracy of M1 by varying C and gamma values and using RandomizedSearchCV.

In [ ]:

```
from sklearn.model_selection import RandomizedSearchCV
from scipy.stats import reciprocal, uniform
start = time.time()
param_distributions = {"gamma": reciprocal(0.001, 0.1), "C": uniform(1, 10)}
```

```
rnd_search_cv = RandomizedSearchCV(svm_clf, param_distributions, n_iter=10, verbose=2, c
v=3)
rnd_search_cv.fit(X_train_scaled[:1000], y_train[:1000])
search_time = time.time() - start
#Adding all values of hyperparameters in a list from which the values of hyperparameter w
ill randomly inserted as hyperparameter
```

```
Fitting 3 folds for each of 10 candidates, totalling 30 fits
[CV] END ...C=9.488249185602088, gamma=0.0016950102517403448; total time=   0.1s
[CV] END ...C=9.488249185602088, gamma=0.0016950102517403448; total time=   0.1s
[CV] END ...C=9.488249185602088, gamma=0.0016950102517403448; total time=   0.2s
[CV] END .....C=5.722434918537763, gamma=0.06816203809183274; total time=   0.2s
[CV] END .....C=5.722434918537763, gamma=0.06816203809183274; total time=   0.2s
[CV] END .....C=5.722434918537763, gamma=0.06816203809183274; total time=   0.2s
[CV] END ...C=5.5365949778274715, gamma=0.012195965597750335; total time=   0.1s
[CV] END ...C=5.5365949778274715, gamma=0.012195965597750335; total time=   0.1s
[CV] END ...C=5.5365949778274715, gamma=0.012195965597750335; total time=   0.1s
[CV] END ...C=10.603960657589235, gamma=0.009708355952104947; total time=   0.1s
[CV] END ...C=10.603960657589235, gamma=0.009708355952104947; total time=   0.1s
[CV] END ...C=10.603960657589235, gamma=0.009708355952104947; total time=   0.1s
[CV] END ....C=2.967125980906471, gamma=0.006091994599711698; total time=   0.2s
[CV] END ....C=2.967125980906471, gamma=0.006091994599711698; total time=   0.1s
[CV] END ....C=2.967125980906471, gamma=0.006091994599711698; total time=   0.1s
[CV] END ....C=4.874444123466981, gamma=0.008227571361234563; total time=   0.1s
[CV] END ....C=4.874444123466981, gamma=0.008227571361234563; total time=   0.1s
[CV] END ....C=4.874444123466981, gamma=0.008227571361234563; total time=   0.2s
[CV] END ....C=4.498088898564092, gamma=0.002006199435372042; total time=   0.1s
[CV] END ....C=4.498088898564092, gamma=0.002006199435372042; total time=   0.1s
[CV] END ....C=4.498088898564092, gamma=0.002006199435372042; total time=   0.1s
[CV] END ....C=6.831062163835136, gamma=0.052485306126574136; total time=   0.1s
[CV] END ....C=6.831062163835136, gamma=0.052485306126574136; total time=   0.2s
[CV] END ....C=6.831062163835136, gamma=0.052485306126574136; total time=   0.1s
[CV] END ...C=4.861478847419128, gamma=0.0031106638580062564; total time=   0.2s
[CV] END ...C=4.861478847419128, gamma=0.0031106638580062564; total time=   0.1s
[CV] END ...C=4.861478847419128, gamma=0.0031106638580062564; total time=   0.2s
[CV] END ......C=9.86888145722606, gamma=0.02318179716298416; total time=   0.2s
[CV] END ......C=9.86888145722606, gamma=0.02318179716298416; total time=   0.2s
[CV] END ......C=9.86888145722606, gamma=0.02318179716298416; total time=   0.2s
```

In [ ]:

```
rnd_search_cv.best_estimator_
```

Out[ ]:

```
▾ SVC
SVC(C=9.488249185602088, gamma=0.0016950102517403448)
```

In [ ]:

```
# from sklearn.svm import SVC
start = time.time()
svm_clf = SVC(C=9.488249185602088,gamma=0.0016950102517403448, kernel='rbf')
svm_clf.fit(X_train_scaled[:10000], y_train[:10000]) # We use an SVC with an RBF kernel
opt_time = time.time() - start
y_pred = svm_clf.predict(X_train_scaled)
accuracy_score(y_train, y_pred)
```

Out[ ]:

```
0.95465
```

Accuracy after optimising gamma and C values is 95.465%

Task 5: Calculate the computational time of Task 3 and 4.

In [ ]:

```
print("Computational Time")
```

```
print("Linear SVC: ",lin_time)
print("Linear SVC after Feature Scaling: ",lin_scal_time)
print("RBF Kernel: ",lin_scal_time)
print("Searching for optimum value of gamma and C: ",search_time)
print("Optimised SVC: ",opt_time)
```

```
Computational Time
Linear SVC:  225.48418402671814
Linear SVC after Feature Scaling:  667.6223611831665
RBF Kernel:  667.6223611831665
Searching for optimum value of gamma and C:  7.8033607006073
Optimised SVC:  30.32535982131958
```

The computation time reduces significantly for SVC with optimised C and gamma values.