Machine Learning Course Project

ICS3206

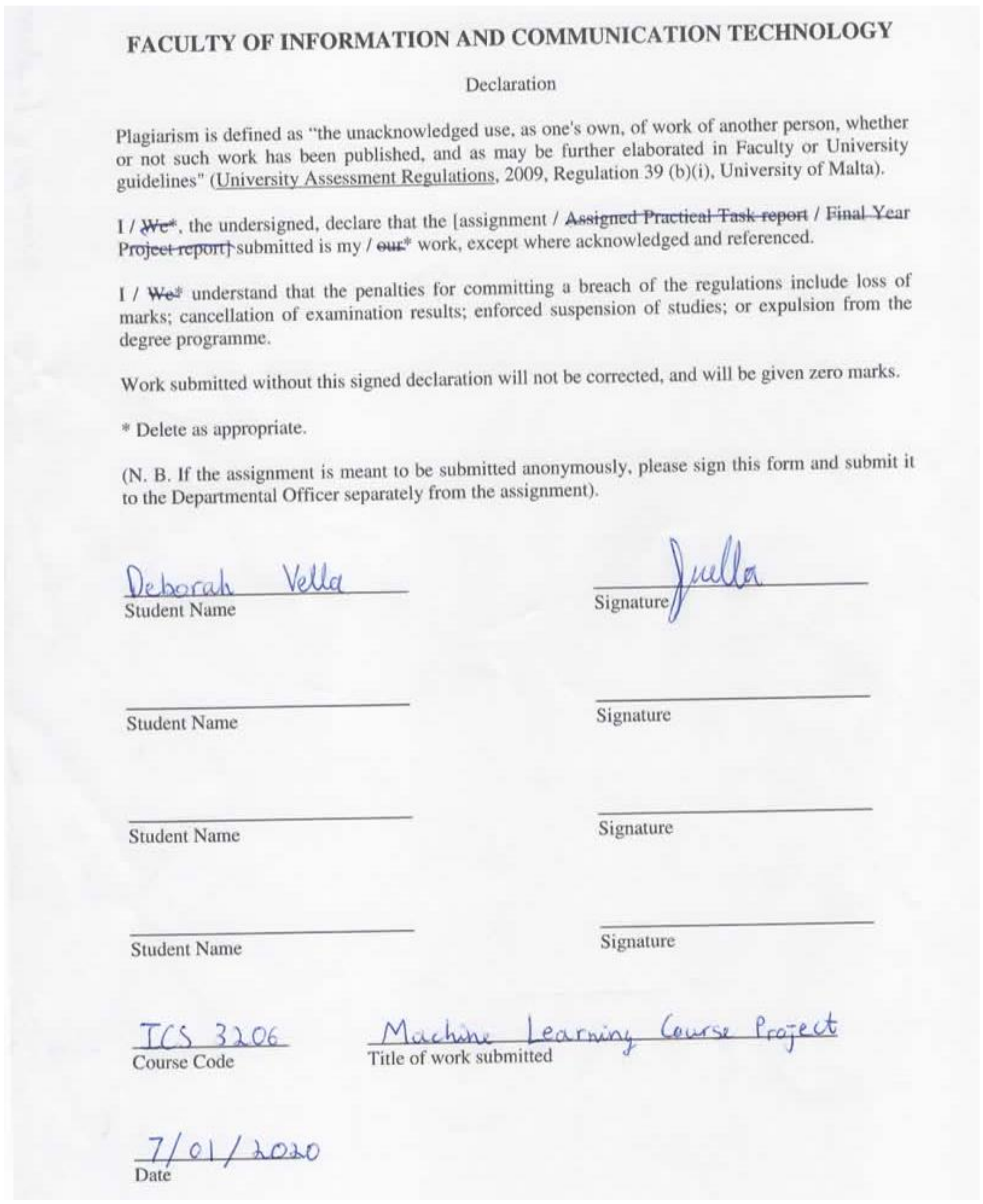
Deborah Vella

Table of Contents

Statement of completion

Item	Completed (Yes/No/Partial)
SVM technical discussion	Yes
A good comparison to alternative methods	Yes
Artefact	Yes
Experimentation with different SVM params	Yes
Experiments and their evaluation	Yes
Overall conclusions	Yes

Plagiarism Declaration Form



FACULTY OF INFORMATION AND COMMUNICATION TECHNOLOGY

Declaration

Plagiarism is defined as "the unacknowledged use, as one's own, of work of another person, whether or not such work has been published, and as may be further elaborated in Faculty or University guidelines" (University Assessment Regulations, 2009, Regulation 39 (b)(i), University of Malta).

I / ~~We~~*, the undersigned, declare that the [assignment / ~~Assigned Practical Task report~~ / ~~Final Year Project report]~~ submitted is my / ~~our~~* work, except where acknowledged and referenced.

I / ~~We~~* understand that the penalties for committing a breach of the regulations include loss of marks; cancellation of examination results; enforced suspension of studies; or expulsion from the degree programme.

Work submitted without this signed declaration will not be corrected, and will be given zero marks.

* Delete as appropriate.

(N. B. If the assignment is meant to be submitted anonymously, please sign this form and submit it to the Departmental Officer separately from the assignment).

Deborah Vella	[illegible]
Student Name	Signature
Student Name	Signature
Student Name	Signature
Student Name	Signature
ICS 3206	Machine Learning Course Project
Course Code	Title of work submitted

7/01/2020
Date

Introduction

This assignment contains a technical discussion on support vector machines (SVM) as well as a comparison of the algorithm with other five classification techniques which are logistic regression, k-nearest neighbour, decision trees, random forest, and neural networks together with convolutional neural networks. An SVM was implemented in Python. Two python files are provided:

- svm_all_kernels.py : this python file runs all the parameter variations discussed in this report by the use of nested loops.
- svm_one_kernel.py : contains the same SVM implementation of the above file, but lets the user choose his own desired kernel, gamma and C value.

This way, either all experiments are run one after the other (which takes 36 iterations) or else, single experiments which need to be tested can also be run. Two .cmd files are also provided to be able to run the programs:

- run_svm_all_kernels.cmd : this runs svm_all_kernels.py
- run_svm_one_kernel.cmd : this runs svm_one_kernel.py

For the programs to run successfully the following libraries must be already installed on the device: numpy, timeit, pandas, matplotlib, seaborn, and sklearn. Furthermore, when an image is successfully displayed in its window, the window should be closed for the program to continue running.

This report discusses the code implemented inside svm_all_kernels.py (given that the other file contains the same SVM implementation). Then the results obtained from all experiments are shown and analysed in-depth to be able to obtain some conclusions on the model. After analysing the results of each kernel with its own variations individually, all the kernels are analysed and compared with each other. The report proceeds with more experimentation done on the whole dataset, instead of a subset of training data. It finishes with the overall conclusions of what is discussed and evaluated in the whole documentation.

SVM technical discussion

SVM overview

Support vector machines are used for solving classification or regression problems. The basic explanation of SVMs is that, given annotated data the support vector machine divides the data with a hyperplane, resulting in different groups of data, one for each class. The SVM's objective is to find the **optimal hyperplane** for the best classification results [1]. The optimal hyperplane is the one which has the maximum distance between itself and the data points of each class, usually referred to as the **maximum margin** [2]. An illustration of this is shown in Figure 1.

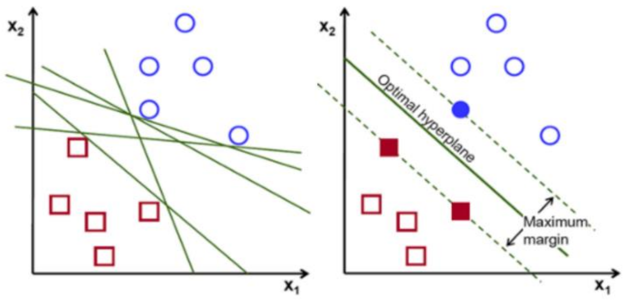


Figure 1 The left-hand side shows the possible hyperplanes. The right-hand side shows the optimal hyperplane [2].

The hyperplane takes up different forms based on the number of features passed as input. For example, as shown in Figure 2 in the case of two features (two-dimensional space), the hyperplane takes the shape of a line, while in the case of three features (three-dimensional space), the hyperplane takes the form of a plane. The more features an SVM takes as input, the more complicated the hyperplane's shape gets [2].

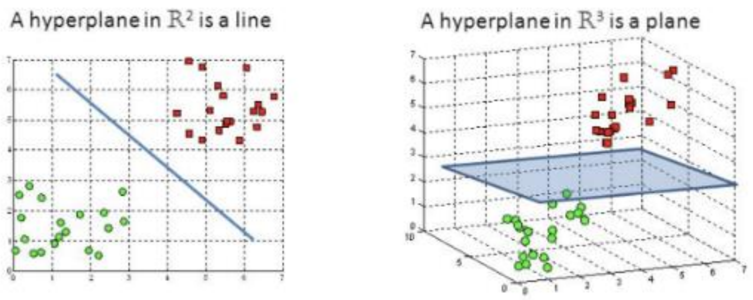


Figure 2 a hyperplane in 2-D space and a hyperplane in 3-D space [2]

The optimal hyperplane is calculated according to the **support vectors**, which in simpler words are the data points nearest to the hyperplane. In fact, support vectors impact the orientation and position the hyperplane takes. In a worst-case scenario, the number of support vectors is equal to the number of training examples. To combine the information so far, as shown in Figure 3 a hyperplane H, is usually defined as:

$$H = w.x_i + b = 0 \text{ [3],}$$

where w is a weight vector, x_i is an input vector and b is a bias.

Furthermore, the planes H1 and H2 which lie on the tips of the support vectors are defined as

$$H_1 = w.x_i + b = 1 \text{ [4] and}$$

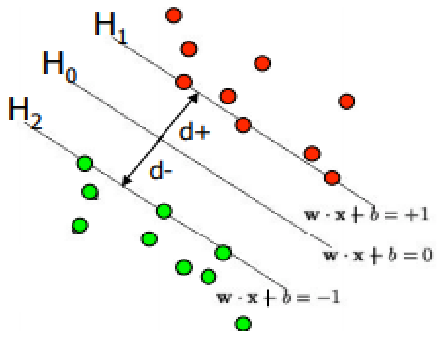
$$H_2 = w.x_i + b = -1 \text{ [4]}$$

Therefore,

$$w.x_i + b \geq 0 \text{ for } d_i = 1 \text{ [4] and}$$

$$w.x_i + b < 0 \text{ for } d_i = -1\text{[4]}$$

where d_i is the shortest distance to the nearest support vector [4].



H_1
H_0
H_2
d+
d-
$\mathbf{w}\cdot\mathbf{x}+b=+1$
$\mathbf{w}\cdot\mathbf{x}+b=0$
$\mathbf{w}\cdot\mathbf{x}+b=-1$

Figure 3 Illustrates the hyperplane, the margin, and the planes on the closest support vectors. This image compliments the formulas discussed. [4]

In some scenarios, it becomes difficult for the SVM to find a hyperplane, especially when the data points are not linearly separable. To address this obstacle, the SVM maps the same data points onto a higher dimensional space, and computes a hyperplane on that particular mapping [3]. A good example is shown in the Figures below, firstly, having all data points mapped on a two-dimensional space in which no linear hyperplane can be found as shown in Figure 5, secondly, the z- axis is added having the point mapped based on their distance from the z-origin. As it may be noted, in Figure 4 a hyperplane could now be easily computed, and then both the hyperplane and the data points are transformed back onto the original dimensional space as seen in Figure 6.

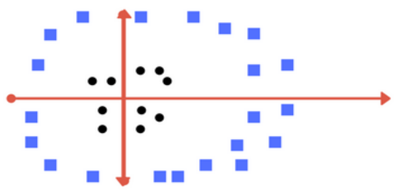


Figure 5 Data points in the original dimensional [1] space

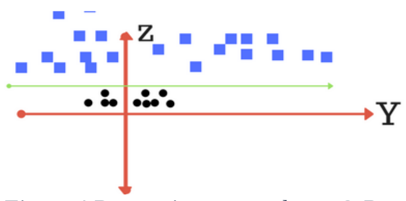


Figure 4 Data points mapped onto 3-D space [1]

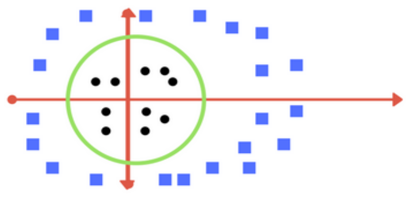


Figure 6 data points and hyperplane mapped back to 2-D space [1]

Kernels

The role of a kernel is to transform the problem using particular functions, so that eventually the support vector machine learns the hyperplane [1]. The term **kernel trick** refers to a kernel which is able to transform from a particular dimensional space to a higher dimensional space, with the aim of making the problem separable [5]. There exist different types of kernels such as linear, polynomial and radial basis function (rbf) . The following are their respective formulas:

Linear:

$$f(x) = B(0) + sum(a_i.(x.x_i))[1],$$

where x is the input, x_i is each support vector while B(0) and a_i are estimated from the training set[1].

Polynomial:

$$K(x_i, x_j) = (1 + x_i . x_j)^d \text{ [3]},$$

where x_i and x_j are feature vectors and d is a tuneable parameter.

RBF:

$$K(x_i, x_j) \exp\left\{\frac{-\|x_i - x_j\|^2}{2\sigma^2}\right\} \text{ [4]}$$

The dot product inside kernel functions are used to be able to compute the similarity of the data points. This is because when the vectors are parallel to each other, the dot product returns one which means that they are highly similar therefore, fall under the same class. On the other hand, when the dot product gives the result of a zero it shows that the points are perpendicular, hence, not similar at all and are not a member of the same class [4].

Regularization

In SVMs regularization is usually denoted as the C parameter. This parameter can be fine-tuned with the SVM to help balancing between “margin maximization and classification violation” [3]. In fact, the larger the C value, the smaller the maximum margin is (Figure 8), while the smaller the C value, the bigger the maximum margin is (Figure 7). This type of tuning may affect the number of misclassifications done [1].

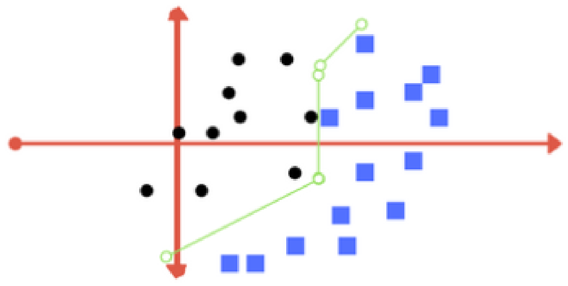


Figure 8 High value of C [1]

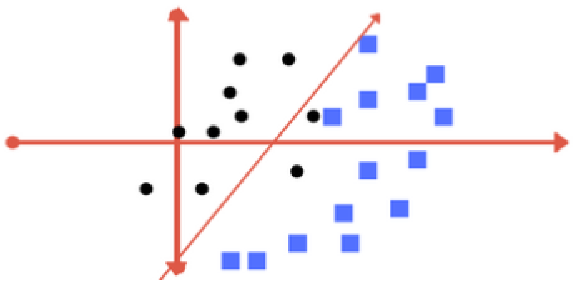


Figure 7 Low value of C [1]

Gamma

The gamma parameter determines a range of which data points should influence the hyperplane. When the gamma is high (Figure 10), only the closest data points to the hyperplane are taken into consideration. In contrast, a small gamma (Figure 9) means that data points further away from the hyperplane will also be considered together with the nearest data points [1].

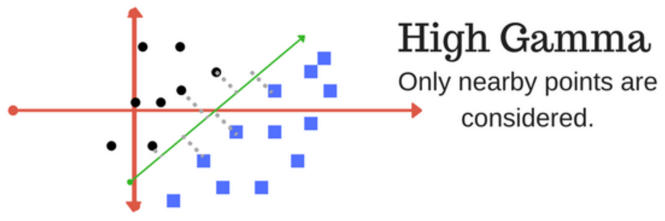


Figure 10 High gamma [1]

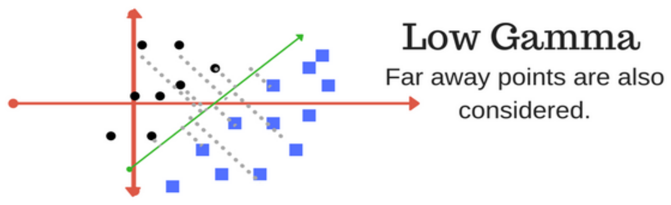


Figure 9 Low gamma [1]

Comparing SVM with other models

Each comparison starts by a small explanation of the technique the SVM is being compared to and proceeds by discussing the difference between the two.

SVM vs Logistic Regression

Logistic regression is also used for classification problems as it takes in an input which is then converted to give a discrete output 0 or 1. To do so, the sigmoid function is used as it transforms any input into a probability between zero and one, as shown in Figure 12. As the output of the sigmoid function is not discrete, a threshold is set assigning anything above it to one and anything below it to zero [6]. This process is shown in Figure 11 below.

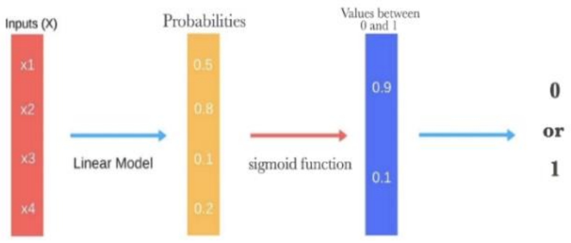


Figure 11 This is the basic process of logistic regression, starting from a set of input and outputting one label [6]

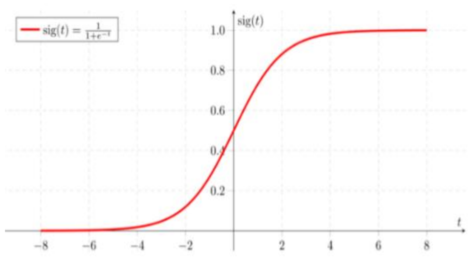


Figure 12 The graph of the sigmoid function [6]

The objective of Logistic regression is that of maximising the likelihood P(y|x) during the training [7], as opposed to that of a support vector machine which is of maximising the margin. Another difference is the loss function. Logistic regression calculates **logistic loss** while an SVM calculates **hinge loss** (used for maximum margin). In both cases the total error is the sum of misclassifications done in a specific iteration [8].

The **logistic loss function** is the following:

$$Logistic\ Loss = \sum_{(x,)\in D} -y\log(y') - (1-y)\log(1-y')\ [9]$$

Where (x,y) is the pair of a training example x and label y (either 0 or 1) from the dataset D and y' is the predicted value [9].

The **hinge Loss function** is the following:

$$max\big(0,1-yf(x)\big)\ [10]$$

Where y is the actual class, x is the input and f(x) of the SVM [10].

Given that the loss functions are different, the optimization problems are the same except for the loss function part. The following are their respective optimization formulas:

SVM optimization problem:

$$min_w \lambda\|w\|^2 + \sum_i max\{0,1-y_i w^t x_i\}\ \ [6]$$

Logistic Regression optimization problem

$$min_w \lambda\|w\|^2 + \sum_i log\big\{1+e^{1-y_i w^t x_i}\big\}\ [6]$$

Figure 13 below, contains the hinge loss and the logistic loss graphs plotted on the same axis to give a more vivid picture of the difference between them. From the graph, it can be noticed that logistic loss diverges earlier than the hinge loss, which implies that logistic loss is more sensitive to outliers. As a result, Logistic Regression tends to classify outliers with its correct class, while an SVM seeks to classify the input as fairly as possible as shown in Figure 14. This stems out from the fact that, in Logistic Regression each and every point influences the line separating the data into classes, as opposed to SVMs in which only the support vectors influence the hyperplane. Furthermore, when analysing Figure 13 one may realise that the logistic loss curve approaches zero but never reaches it, and hence, the x-axis becomes its asymptote. On the other hand, the hinge loss does reach zero, which means that at some point the SVM model will have no loss [8]. Another difference is that logistic regression only works on **linearly separable data**, while an SVM can work on **non-linear separable data** with the use of kernel tricks mentioned before.

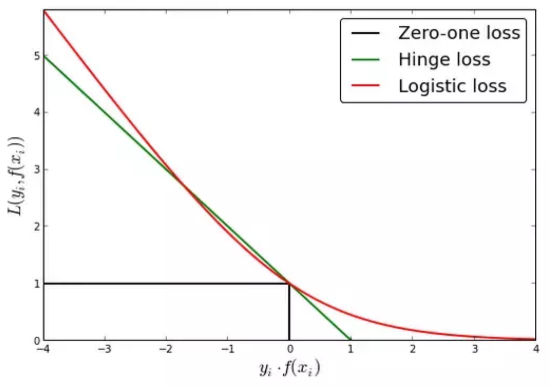


Figure 13 A plot of hinge loss and logistic loss for comparison [6]

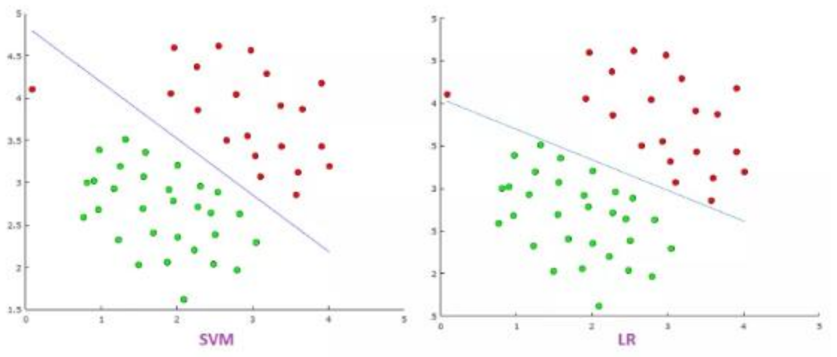


Figure 14 On the left- hand side the SVM is classifying one outlier wrongly but the hyperplane is fair. On the right hand-side Logistic regression classified the outlier right but the line depends on the outlier.

SVM vs KNN

K-nearest neighbour (KNN) is another classification technique used in machine learning. KNN classifies the input data according to the closest training data (nearest neighbours). Each of the training data is assigned with a class, and hence, during classification the algorithm analyses which class is mostly present in the nearest data points and classifies the object with that class. The value of *k* denotes the number of nearest neighbours the algorithm should analyse, so if *k*=3 the closest three data points will be considered. The distance between the object to be classified and its neighbours can be calculated by using different metrics such as Euclidean distance and Manhattan distance. The mostly used metric is the Euclidean distance shown below:

$$d(x,y) = \sum_{i=1}^{N} \sqrt{x_i^2 - y_i^2}$$

Where x and y are two different points, and N is the number of features [11]

As can be seen in Figure 15, the green circle is a new object that should be classified with either the blue squares class or with the red triangles class. If *k*=3 (inner circle) then it will find the closest points which in this case are the two red triangles and one blue square (the two blue squares are equidistant so it picks one of them), so it will be classified as a red triangle. In contrast if *k*=7 (outer and inner circle together), the algorithm would find that the closest are five blue squares and two red triangles, so the object would now be classified as a blue square [12].

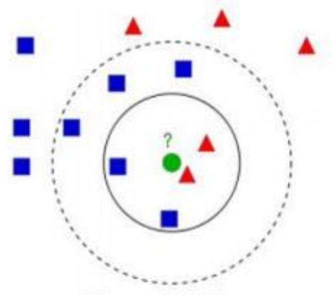


Figure 2: kNN

Figure 15 An illustration of KNN [12]

One main difference is that KNN performs classification with the use of a distance metric, while an SVM needs some time for training prior to classification. Moreover, KNN is mostly used for multiple class classification, as opposed to standard SVM which can classify binary classes (one of two). There are two different approaches that can be taken to classify multiple classes with SVMs, which are the one-vs-one approach or the one-vs-all approach. The first approach trains an SVM for every pair of classes (n*(n-1/2) SVMs), while the second approach requires that an SVM is trained for each and every class (n number of SVMs). SVMs are harder to implement but after training, it can easily classify new unannotated data. On the other hand, the KNN algorithm is much easier to implement, but it has to compute the distance metric every time new unlabelled data is inputted.

In both models some type of parameter should be decided upon, assigning the *k* parameter and distance metric in KNN algorithm and assigning regularization, gamma and kernel for SVM models. It could also be concluded that in most cases, SVM models do a better job at classification [12] and dealing with outliers [13], but the KNN algorithm still gets rather high accuracy and good enough performance in particular situations [12]. KNN is found to be better when the training data is a lot larger than the number of features, but an SVM is much better when there is less training data and a huge number of features [13].

SVM vs Decision Trees

A decision tree is a tree-like structure which like an SVM, can be used both for classifications and regression. As the SVM model is used for classification in this assignment, this section will discuss only the classification aspect of decision trees. In classification situations, the root node contains all the data while the leaf nodes contain the class labels [14]. Between the root and the leaf nodes, there are multiple sets of nodes, each one splitting the tree into sub-trees. This means that at every node, a binary decision is made with the aim of separating classes from each other. The tree is traversed using a top-down approach, thus, starting from the root node and ending at a leaf node by traversing internal nodes downwards [15]. Figure 16 is an example of a decision tree, having the data firstly split based on the colour, then split based on whether there is an underline or not.

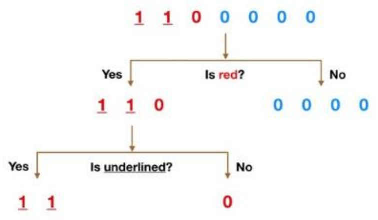


Figure 16 A simple decision tree [16]

Both decision trees and support vector machines are not influenced by outliers in the data. Moreover, decision trees work well with non-linearity as they develop hyper-rectangles for such situations. As already discussed, SVMs require the use of kernel trick to address a non-linear problem. Apart from this, decision trees are also very useful for multi-class predictions, as multiple leaves represent multiple classes. when comparing SVM with KNN, it was discussed that a support vector machine has to take one of two approaches one-vs-one or one-vs-all, to classify multiple classes [13].

SVM vs Random Forest

Random Forest model builds upon decision trees. This is because Random Forest is a collection of decision trees, having each one outputting its own prediction. The most common result provided by the forest is then chosen to be the final prediction of the given data [16]. Random Forest gain higher accuracies than decision trees, and is also a more robust model. Overfitting can occur more in a decision tree alone than a Random Forest [13]. Figure 17 is an illustration of what a Random Forest is, having each decision tree predicting a class and then choosing the most popular one, which in this case is 1.

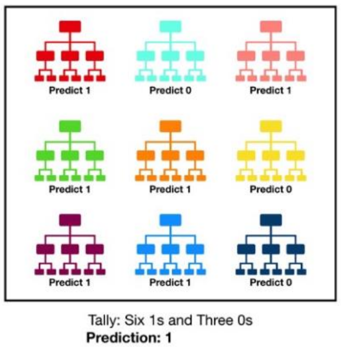


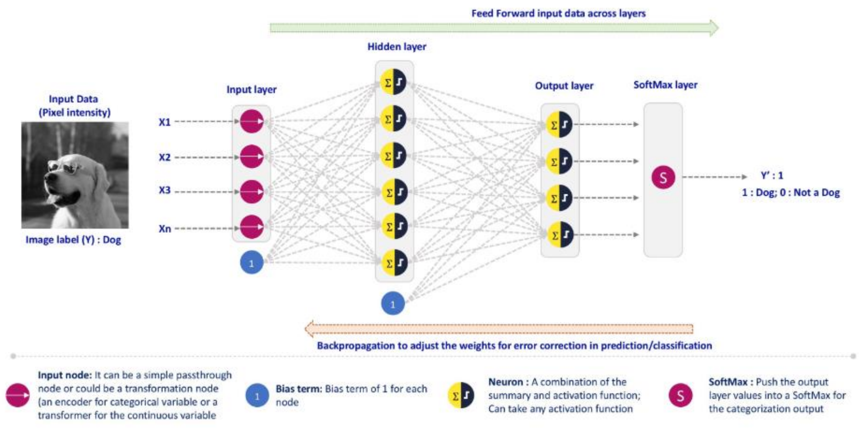
Figure 17 A representation of Random Forest [16]

Just like decision tree, Random Forest is well suited in the case of multiclass problems, while an SVM is more suited for binary class classifications. SVMs are better when used on sparse data than Random Forest, as it only considers a small set of the data which is probably the most relevant of all. Random Forest is a faster algorithm than SVM, as the training is not as long and does not need that much pre-processing of the data. On the other hand, Random Forest has a higher risk of overfitting than SVM. Random Forest can handle a mixture of categorical and numerical features, while an SVM can handle a very large number of features. Another difference is that Random Forest does not have any hyperparameters that need tuning, while SVM has three. Both of these models can handle high dimensional spaces and are also non-parametric [17].

SVM vs CNN and Neural Networks in general

A Convolutional Neural Networks (CNN) is a deep-feed-forward neural network which is able to solve classification problems. CNNs make use of convolutional layers which are filters that move along the image's height and width, to compute the dot product of that region and the weights. CNNs also contain the pooling layer, which down samples the input images together with the number of parameters. Each feed-forward network inside the CNN ends with an activation function to be able to deal with non-linearity such as tanh, ReLU (mostly used) and sigmoid function [18].

Neural Networks in general are considered to be better than SVMs due to various factors. A CNN can contain **multiple neural layers** stacked together, which give the CNN a higher learning capacity [19]. The aforementioned **activation functions** play a good role in making the CNNs even better than SVMs, usually using ReLU function for the inner layers as it accelerates stochastic gradient descent convergence and is less computationally expensive. The output layer on the other hand, usually uses a softmax function [18].



Feed Forward input data across layers
Hidden layer
Input layer
Output layer
SoftMax layer
Input Data
(Pixel intensity)
X1
X2
X3
Xn
Image label (Y) : Dog
Y' : 1
1 : Dog; 0 : Not a Dog
Backpropagation to adjust the weights for error correction in prediction/classification
Input node: It can be a simple passthrough node or could be a transformation node (an encoder for categorical variable or a transformer for the continuous variable
Bias term: Bias term of 1 for each node
Neuron : A combination of the summary and activation function; Can take any activation function
SoftMax : Push the output layer values into a SoftMax for the categorization output

Figure 18 A representation of a Neural Network [19]

CNNs also make use of some kind of **optimization step**, so that the weights and parameters are optimized accordingly. Such optimizers are, gradient descent, stochastic gradient descent, momentum and Adam optimizer. **Backpropagation** is also used in some cases, to get a better performance of the neural network. This algorithm calculates the output of each layer including the output layer, and then computes the prediction error. It continues by modifying the weights of the neurons, starting from the output layer and finishing at the input layer based on the prediction layer.

Neural Networks, may contain some type of regularization technique such as dropout. When a dropout is used, at each iteration, a fraction of the neurons inside the layers are dropped from the entire network. This is mainly used to prevent the model from overfitting to the dataset.

Neural Networks have become way more popular than SVMs and this can be seen in Figure 19, which illustrates the number of articles written on different types of neural networks and the amount of articles written on SVM, throughout the years [19].

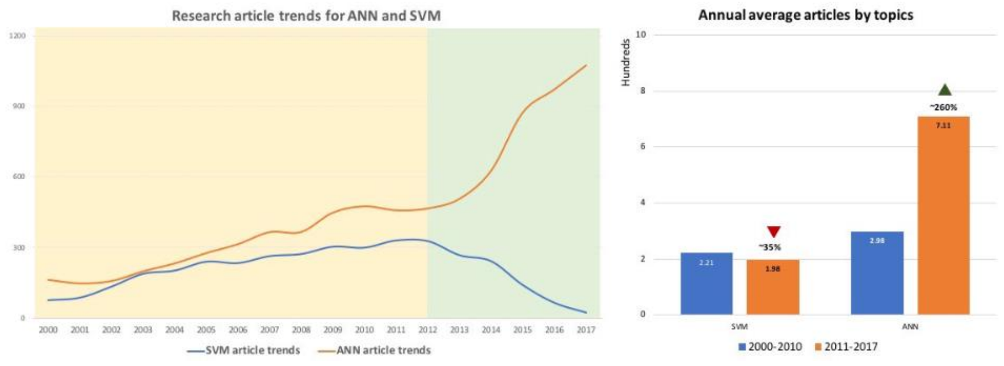


Figure 19 Statistics of the volume of articles written on neural networks, and the volume of articles written on SVMs from 2000 to 2017 [19]

Artefact

Python files

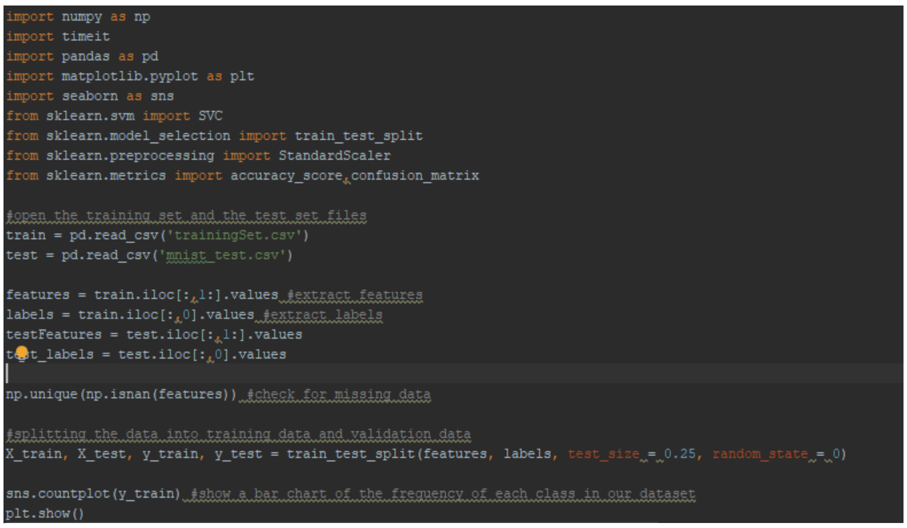
Two python files are submitted: svm_all_kernels.py which computes all variations on all kernels, and svm_one_kernel.py which runs the SVM model with the parameters inputted by the user. The latter is submitted because the first python file might take long to run and might also run out of memory during run-time. It also allows the user to just run the SVM on the parameters he would like to test out.

Datasets

It was decided that the MNIST dataset will be used for this assignment, containing labelled images from 0 to 9. The training and test sets used are csv files generated by Joseph Redmon, which is available online[1]. The training set was then altered to contain only 12,100 training examples to decrease the computation time of the SVM during training. The accuracy of the model might drop a bit as it has less data to train upon. At the end, the whole training set of 60,000 training examples is also used but using only the best parameters of the SVM, to analyse any difference in accuracies. On the other hand, the test set was not modified as it contains 10,000 test examples.

Implementation

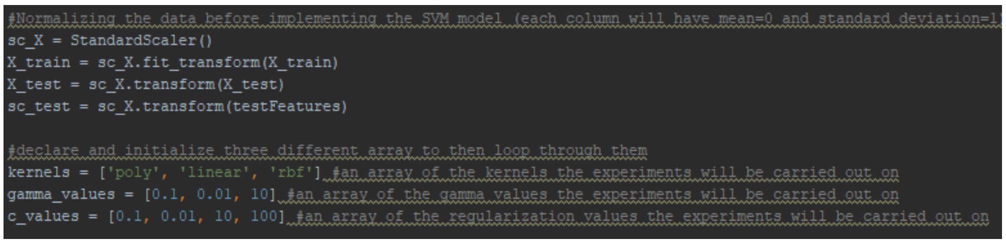
To implement the SVM, Python's sklearn library was used, from which SVC is imported. The code starts by importing the required packages which are numpy, timeit, pandas, matplotlib, seaborn, and sklearn together with its sub-packages. It then proceeds by opening the csv files containing the datasets, extracting the labels and images and stores them separately, and continues by splitting the data into a training set and a validation set. A bar chart is then plotted to show the user how many training examples there are in the data set for each 10 different classes.



```python
import numpy as np
import timeit
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score,confusion_matrix

#open the training set and the test set files
train = pd.read_csv('trainingSet.csv')
test = pd.read_csv('mnist_test.csv')

features = train.iloc[:,1:].values #extract features
labels = train.iloc[:,0].values #extract labels
testFeatures = test.iloc[:,1:].values
test_labels = test.iloc[:,0].values

np.unique(np.isnan(features)) #check for missing data

#splitting the data into training data and validation data
X_train, X_test, y_train, y_test = train_test_split(features, labels, test_size = 0.25, random_state = 0)

sns.countplot(y_train) #show a bar chart of the frequency of each class in our dataset
plt.show()
```

Figure 20 The handling of data sets, splitting them into labels, and images, and splitting them into validation and training sets

Prior to the implementation and training of the SVM the data is normalised, so that each column has its mean equal to zero and standard deviation equal to one. Afterwards, three different arrays are declared and initialised, each one containing different values to be able to compute different variations. In fact, each array contains values for the three different parameters of an SVM which are: kernel, gamma value and a regularization value.

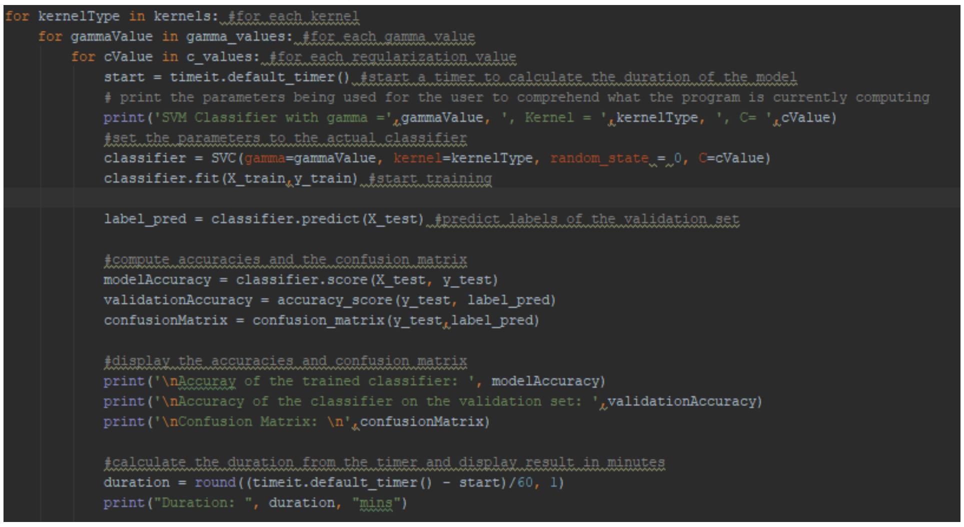


```python
#Normalizing the data before implementing the SVM model (each column will have mean=0 and standard deviation=1
sc_X = StandardScaler()
X_train = sc_X.fit_transform(X_train)
X_test = sc_X.transform(X_test)
sc_test = sc_X.transform(testFeatures)

#declare and initialize three different array to then loop through them
kernels = ['poly', 'linear', 'rbf'] #an array of the kernels the experiments will be carried out on
gamma_values = [0.1, 0.01, 10] #an array of the gamma values the experiments will be carried out on
c_values = [0.1, 0.01, 10, 100] #an array of the regularization values the experiments will be carried out on
```

Figure 21 Normalization of dataset and initialisation of three arrays, one for each parameter

[1] https://pjreddie.com/projects/mnist-in-csv/

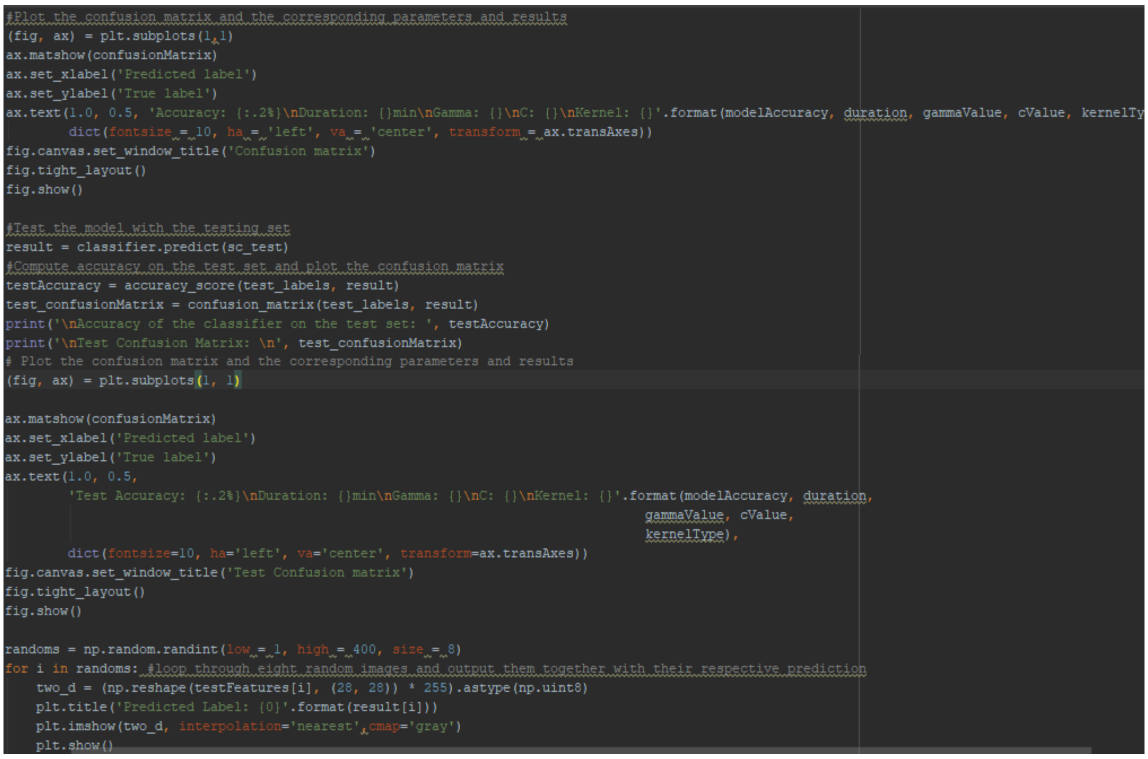
The nested loops which iterate through all three arrays mentioned above, are implemented to be able to test the SVM model on all possible combinations of the three parameters. Hence, the program will loop for 36 iterations (no. of kernels * no. of gamma values * no. of C values = 3*3*4). At each iteration were a different C parameter is used, the time at that particular moment will be taken, to eventually be able to calculate the duration of the model. Each time a parameter is changed, the user is informed about which new parameters are being used at that point. The SVM is then trained with the same parameters displayed to the user, and the validation set is used to test out the model. The model accuracy and the validation accuracy are then computed and displayed followed by a confusion matrix. After this, the time is calculated and printed, as the SVM training and validation would be finished by then.



```
for kernelType in kernels: #for each kernel
    for gammaValue in gamma_values: #for each gamma value
        for cValue in c_values: #for each regularization value
            start = timeit.default_timer() #start a timer to calculate the duration of the model
            # print the parameters being used for the user to comprehend what the program is currently computing
            print('SVM Classifier with gamma =',gammaValue, ', Kernel = ',kernelType, ', C= ',cValue)
            #set the parameters to the actual classifier
            classifier = SVC(gamma=gammaValue, kernel=kernelType, random_state = 0, C=cValue)
            classifier.fit(X_train,y_train) #start training

            label_pred = classifier.predict(X_test) #predict labels of the validation set

            #compute accuracies and the confusion matrix
            modelAccuracy = classifier.score(X_test, y_test)
            validationAccuracy = accuracy_score(y_test, label_pred)
            confusionMatrix = confusion_matrix(y_test,label_pred)

            #display the accuracies and confusion matrix
            print('\nAccuray of the trained classifier: ', modelAccuracy)
            print('\nAccuracy of the classifier on the validation set: ',validationAccuracy)
            print('\nConfusion Matrix: \n',confusionMatrix)

            #calculate the duration from the timer and display result in minutes
            duration = round((timeit.default_timer() - start)/60, 1)
            print("Duration: ", duration, "mins")
```

Figure 22 The nested loops, containing the SVM model and statistical evaluation.

The confusion matrix is plotted for a better visualization of the results. Next to the plot, the respective parameters used, time taken and model accuracy are displayed to provide the user the plot's context. The code ends by passing the test set into the SVM for classification. The accuracy and confusion matrix are computed and outputted again but this time on the testing results. At last, eight random images are picked from the test set and displayed together with its corresponding prediction, so that the user has a sample of the classifications.

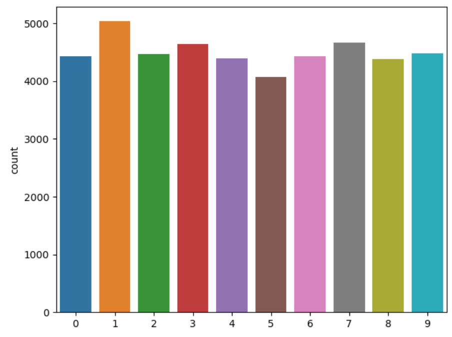


```
#Plot the confusion matrix and the corresponding parameters and results
(fig, ax) = plt.subplots(1,1)
ax.matshow(confusionMatrix)
ax.set_xlabel('Predicted label')
ax.set_ylabel('True label')
ax.text(1.0, 0.5, 'Accuracy: {:.2%}\nDuration: {}min\nGamma: {}\nC: {}\nKernel: {}'.format(modelAccuracy, duration, gammaValue, cValue, kernelTy
        dict(fontsize = 10, ha = 'left', va = 'center', transform = ax.transAxes))
fig.canvas.set_window_title('Confusion matrix')
fig.tight_layout()
fig.show()

#Test the model with the testing set
result = classifier.predict(sc_test)
#Compute accuracy on the test set and plot the confusion matrix
testAccuracy = accuracy_score(test_labels, result)
test_confusionMatrix = confusion_matrix(test_labels, result)
print('\nAccuracy of the classifier on the test set: ', testAccuracy)
print('\nTest Confusion Matrix: \n', test_confusionMatrix)
# Plot the confusion matrix and the corresponding parameters and results
(fig, ax) = plt.subplots(1, 1)

ax.matshow(confusionMatrix)
ax.set_xlabel('Predicted label')
ax.set_ylabel('True label')
ax.text(1.0, 0.5,
        'Test Accuracy: {:.2%}\nDuration: {}min\nGamma: {}\nC: {}\nKernel: {}'.format(modelAccuracy, duration,
                                                                                      gammaValue, cValue,
                                                                                      kernelType),

        dict(fontsize=10, ha='left', va='center', transform=ax.transAxes))
fig.canvas.set_window_title('Test Confusion matrix')
fig.tight_layout()
fig.show()

randoms = np.random.randint(low = 1, high = 400, size = 8)
for i in randoms: #loop through eight random images and output them together with their respective prediction
    two_d = (np.reshape(testFeatures[i], (28, 28)) * 255).astype(np.uint8)
    plt.title('Predicted Label: {0}'.format(result[i]))
    plt.imshow(two_d, interpolation='nearest',cmap='gray')
    plt.show()
```

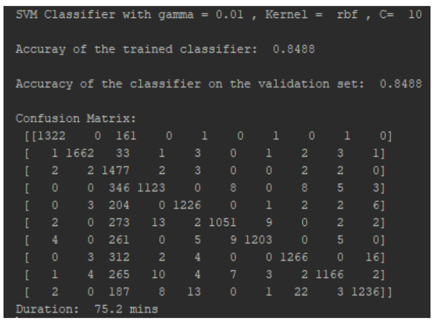
Figure 23 The program computing accuracies and confusion matrices on the validation set and the testing set. It finished by outputting a sample of eight images from the test results.

Example Output

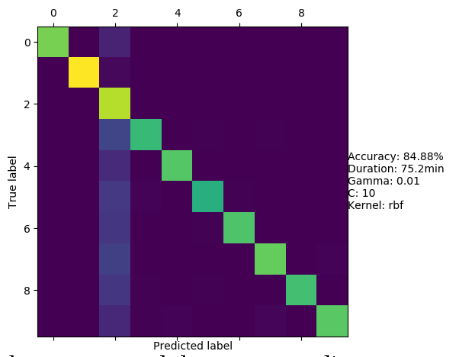
The first thing the program outputs is a bar chart of the distribution of data in the dataset, as shown in the Figure below. This assists the user to visualise any discrepancies in the data.



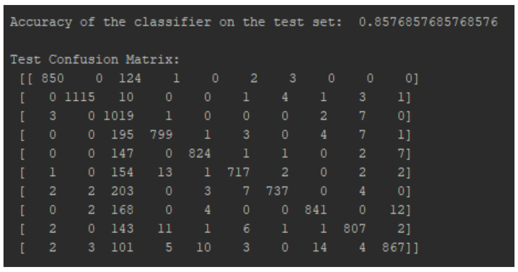
It then prints the accuracy and the confusion matrix on the validation set to the console, together with its corresponding plot, such as the images below.



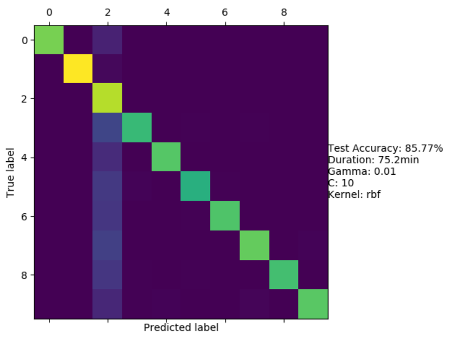
```
SVM Classifier with gamma = 0.01 , Kernel =  rbf , C=  10

Accuray of the trained classifier:  0.8488

Accuracy of the classifier on the validation set:  0.8488

Confusion Matrix:
 [[1322    0  161    0    1    0    1    0    1    0]
 [   1 1662   33    1    3    0    1    2    3    1]
 [   2    2 1477    2    3    0    0    2    2    0]
 [   0    0  346 1123    0    8    0    8    5    3]
 [   0    3  204    0 1226    0    1    2    2    6]
 [   2    0  273   13    2 1051    9    0    2    2]
 [   4    0  261    0    5    9 1203    0    5    0]
 [   0    3  312    2    4    0    0 1266    0   16]
 [   1    4  265   10    4    7    3    2 1166    2]
 [   2    0  187    8   13    0    1   22    3 1236]]
Duration:  75.2 mins
```



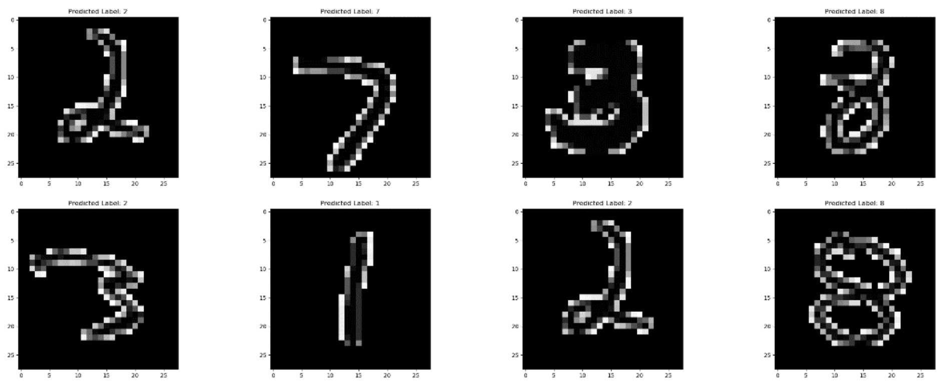
Then, the program proceeds by classifying the data inside the test set, and the corresponding accuracy and confusion matrix are computed and plotted as well.



```
Accuracy of the classifier on the test set:  0.8576857685768576

Test Confusion Matrix:
 [[ 850    0  124    1    0    2    3    0    0    0]
 [   0 1115   10    0    0    1    4    1    3    1]
 [   3    0 1019    1    0    0    0    2    7    0]
 [   0    0  195  799    1    3    0    4    7    1]
 [   0    0  147    0  824    1    1    0    2    7]
 [   1    0  154   13    1  717    2    0    2    2]
 [   2    2  203    0    3    7  737    0    4    0]
 [   0    2  168    0    4    0    0  841    0   12]
 [   2    0  143   11    1    6    1    1  807    2]
 [   2    3  101    5   10    3    0   14    4  867]]
```



It finishes by outputting eight sample images from the results, showing their respective prediction made by the SVM such as the below.



Experimentation and Evaluation of different parameters

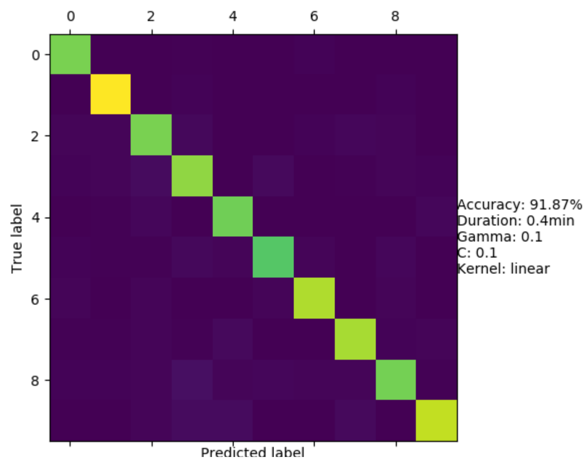
The confusion matrices computed on the test set are very similar to ones computed on the validation set, therefore, in this section only the ones printed on validation are shown, for each kernel. When running the program both of them are displayed for each combination of the parameters. Two tables are shown for each kernel, one containing the accuracies of different variations on tested on validation set and the other on the test set, to be able to compare them with each other. Furthermore, the parameters' variations are:

- *Variations of kernel:* polynomial, linear and RBF.
- *Variations of gamma:* 0.1, 0.01, 10
- *Variations of C:* 0.1, 0.01, 10, 100

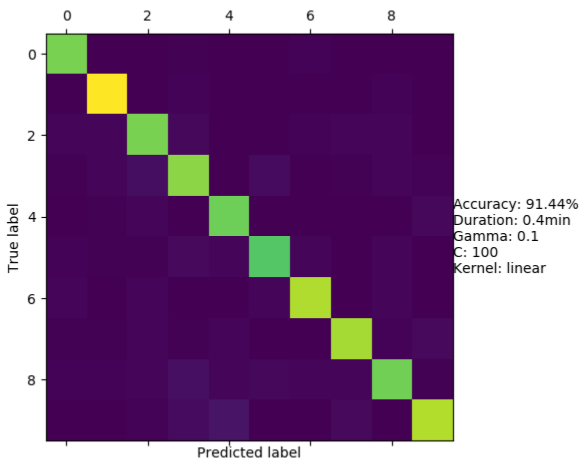
Linear Kernel

Confusion Matrices and Accuracy tables

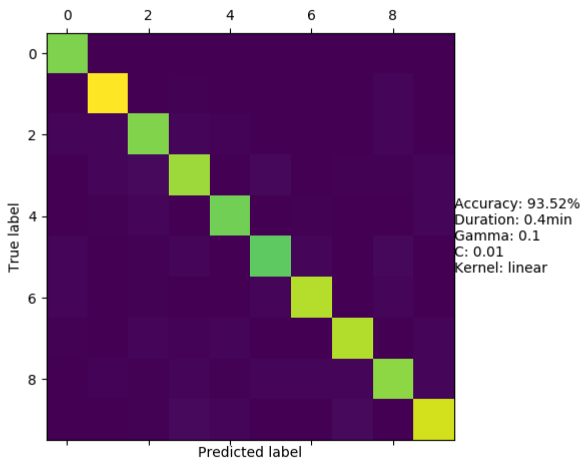
The following are the confusion matrices computed on the validation set, when using a linear kernel.



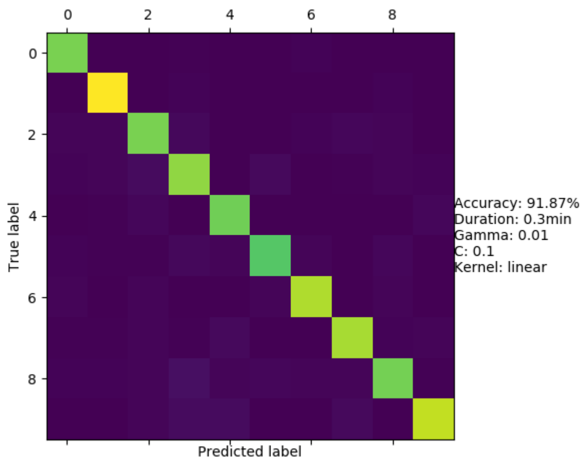
Accuracy: 91.87%
Duration: 0.4min
Gamma: 0.1
C: 0.1
Kernel: linear



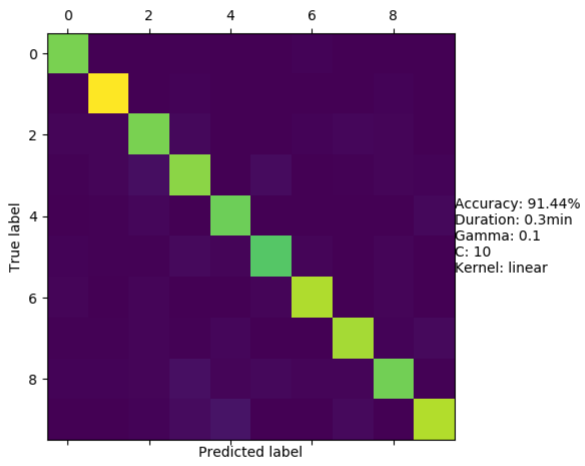
Accuracy: 91.44%
Duration: 0.4min
Gamma: 0.1
C: 100
Kernel: linear



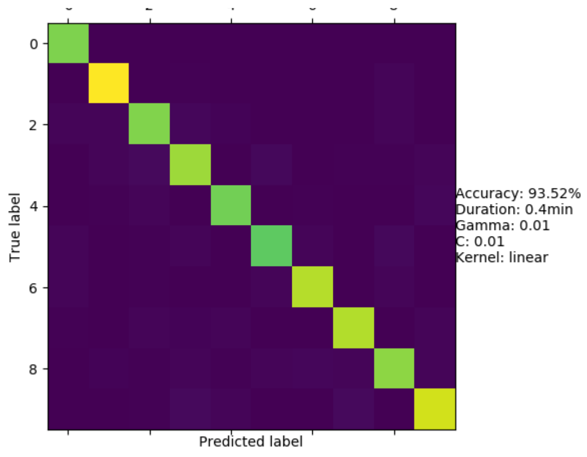
Accuracy: 93.52%
Duration: 0.4min
Gamma: 0.1
C: 0.01
Kernel: linear



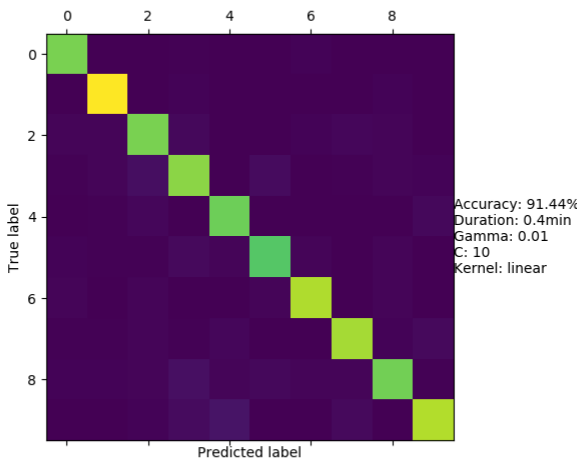
Accuracy: 91.87%
Duration: 0.3min
Gamma: 0.01
C: 0.1
Kernel: linear

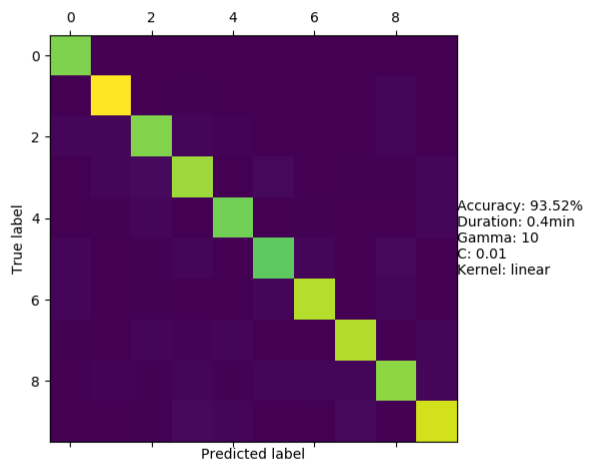


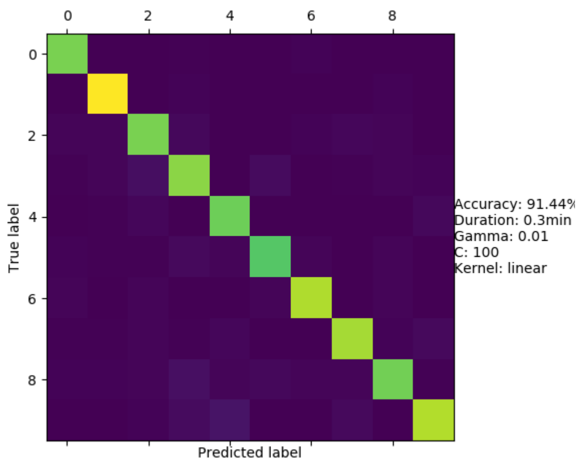
Accuracy: 91.44%
Duration: 0.3min
Gamma: 0.1
C: 10
Kernel: linear

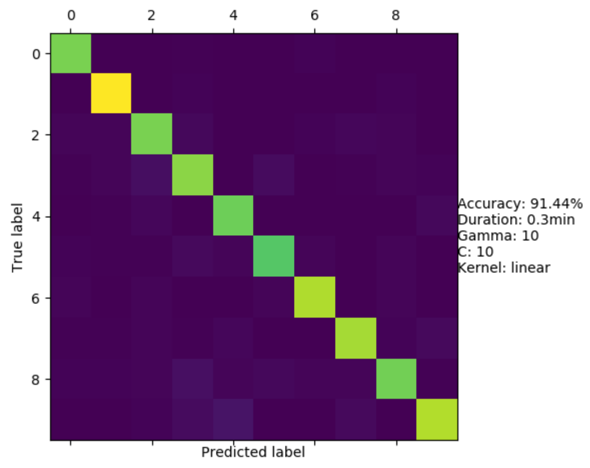


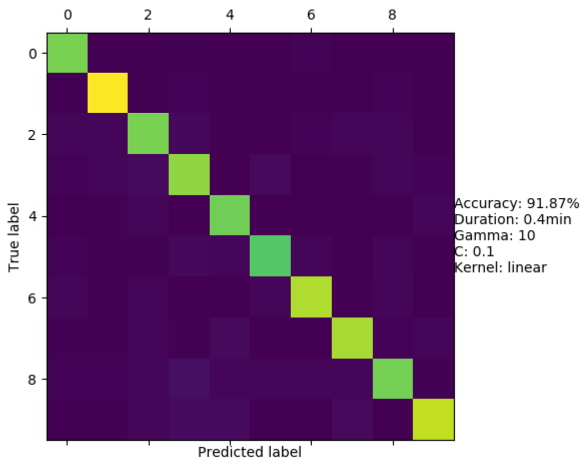
Accuracy: 93.52%
Duration: 0.4min
Gamma: 0.01
C: 0.01
Kernel: linear

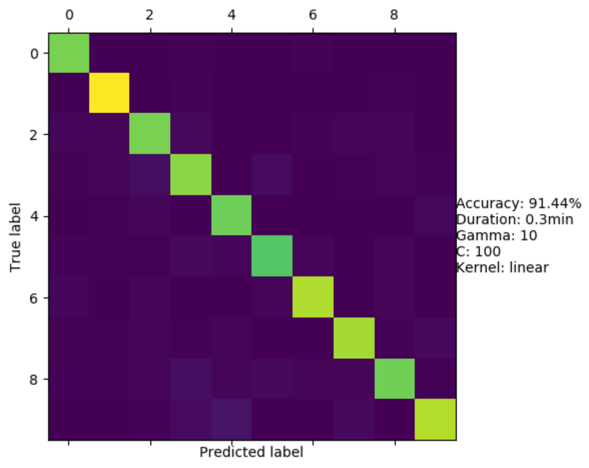












The following is a table of the accuracies (in %) of the SVM model on the validation set given different gamma and C parameters.

Gamma \ C	0.01	0.1	10	100
0.1	93.52	91.87	91.44	91.44
0.01	93.52	91.87	91.44	91.44
10	93.52	91.87	91.44	91.44

The following is a table of the accuracies (in %) of the SVM model on the test set given different gamma and C parameters.

Gamma \ C	0.01	0.1	10	100
0.1	92.78	91.68	91.38	91.38
0.01	92.78	91.68	91.38	91.38
10	92.78	91.68	91.38	91.38

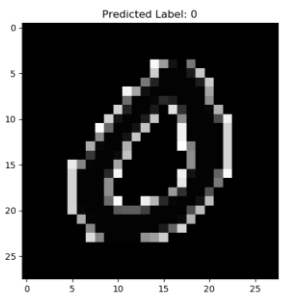
Evaluation of confusion matrices and tables

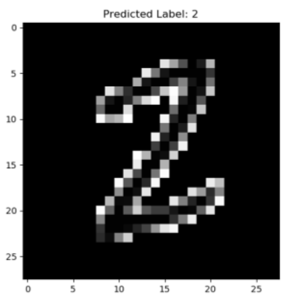
When analysing the tables above, which were built based on the confusion matrices outputted by the program, it can be concluded that the C value 0.01 gave the best accuracy. It can also be noted that for every C value, the gamma parameter did not affect the accuracy, for example the cells in the column of 0.1 in the first table, all contain the value 91.87%. Moreover, when the regularization value is increased, the accuracy does not increase with it. In fact, it decreases a bit, and there is no difference between the accuracies when the C=10 and C=100. All confusion matrices plots contain diagonals of green and yellow squares which show that the majority of the input data is classified with its correct class. When taking into consideration the duration, one may realize that the SVM is not very time consuming to train using a linear kernel, as the highest duration is 0.4min.

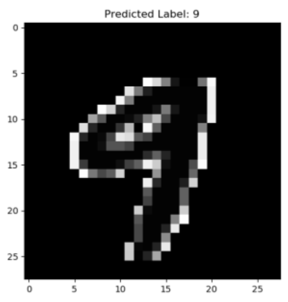
Sample images

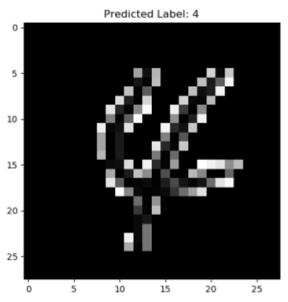
For each combination of variations, four sample images (out of the eight outputted by the program) of the classifications done on the test set are illustrated below.

Gamma = 0.1, C=0.1

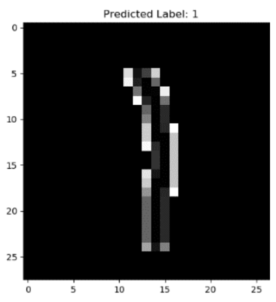


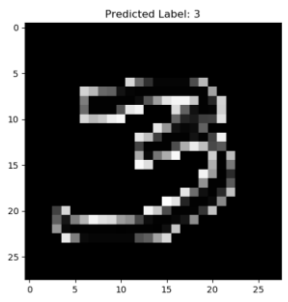


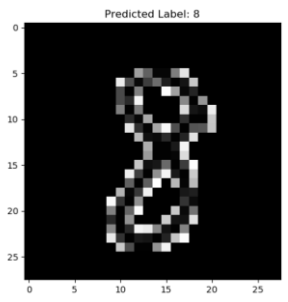


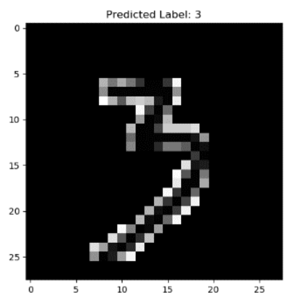


Gamma= 0.1, C = 0.01

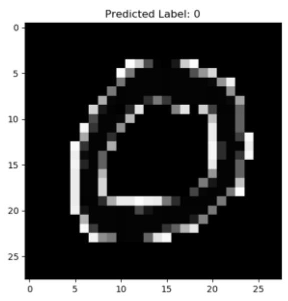


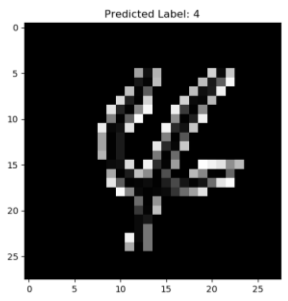


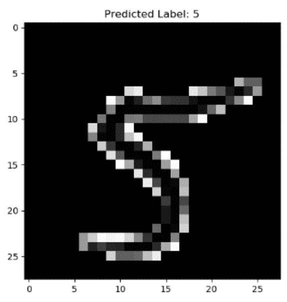


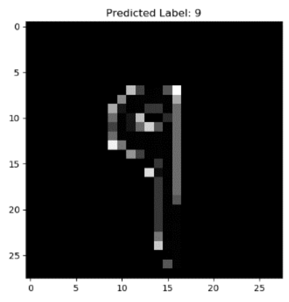


Gamma=0.1, C=10

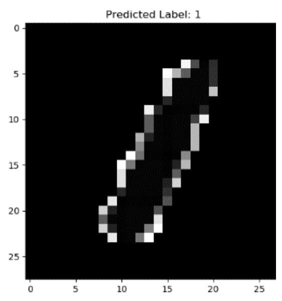


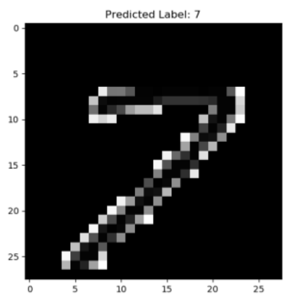


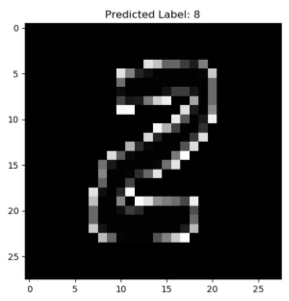


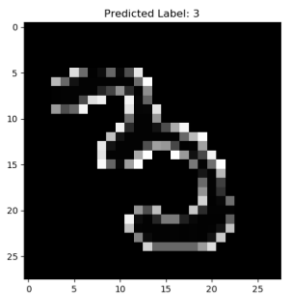


Gamma=0.1, C=100

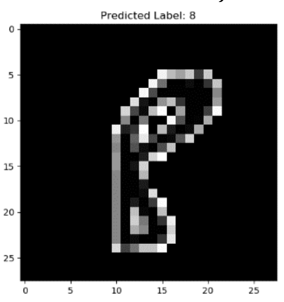


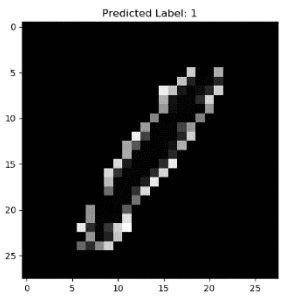


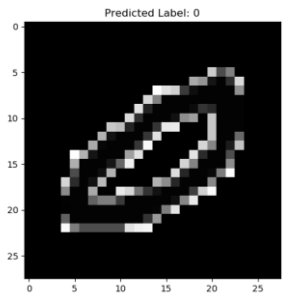


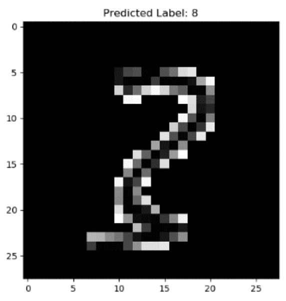


Gamma=0.01, C=0.1

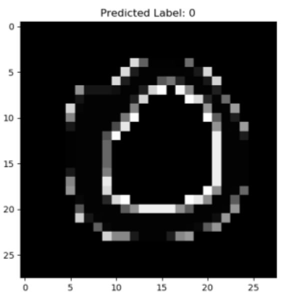


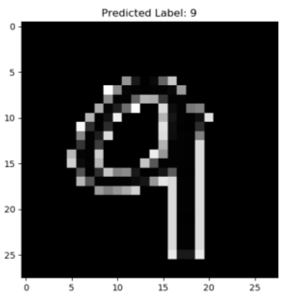


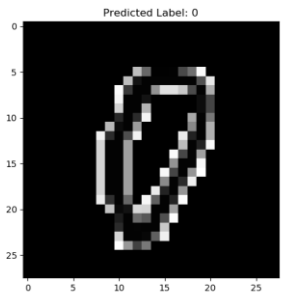


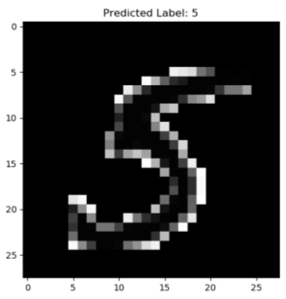


Gamma-0.01, C=0.01

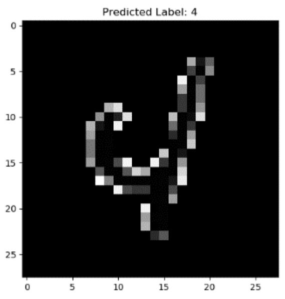


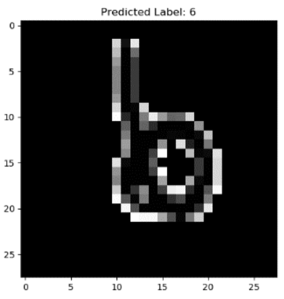


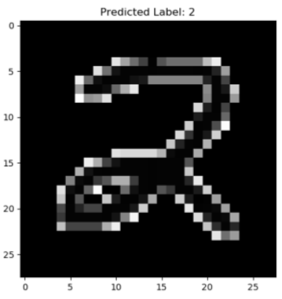


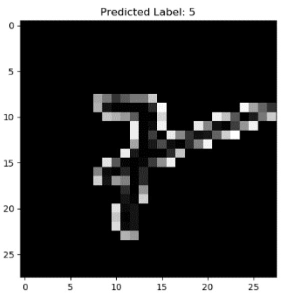


Gamma=0.01, C=10

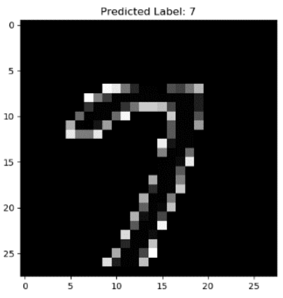


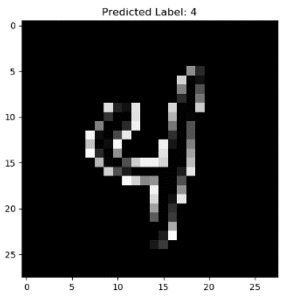


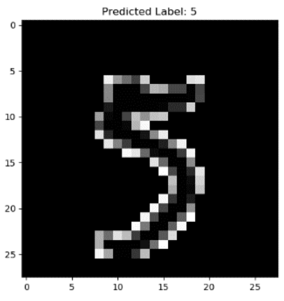


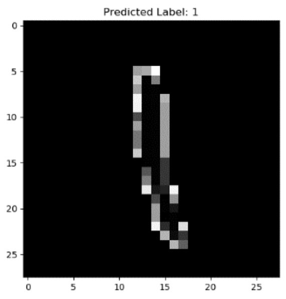


Gamma=0.01, C=100

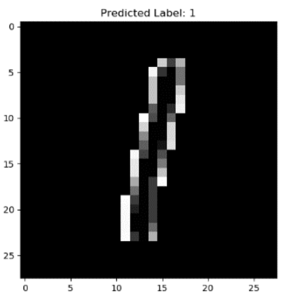


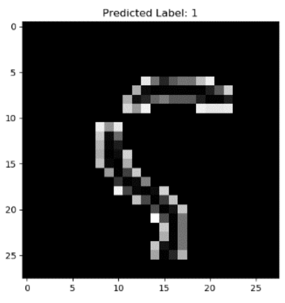


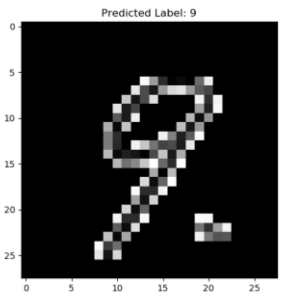


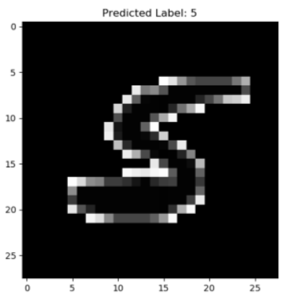


Gamma=10, C=0.1

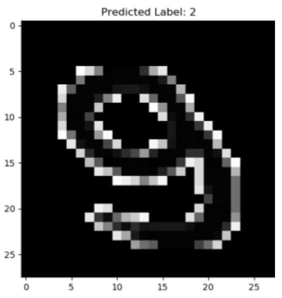


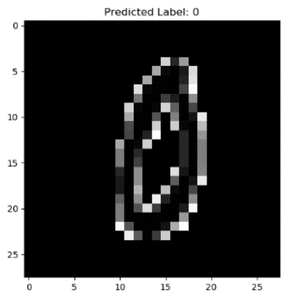


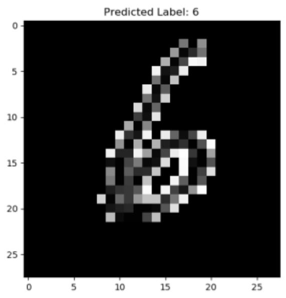


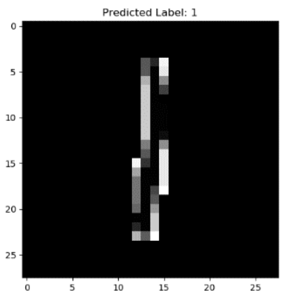


Gamma=10, C=0.01

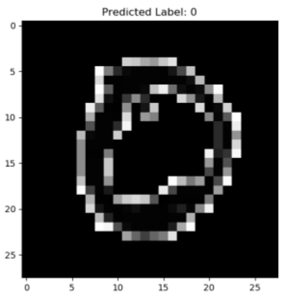


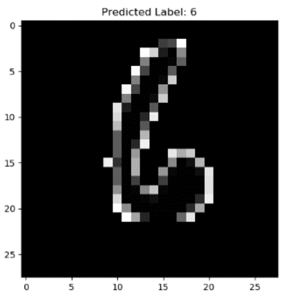


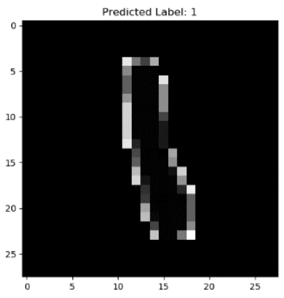


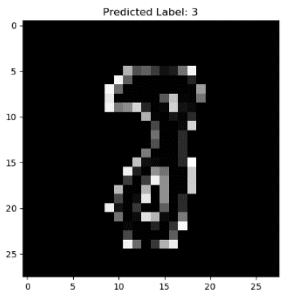


Gamma=10, C=10

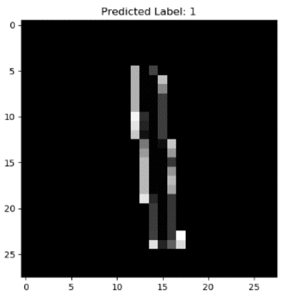


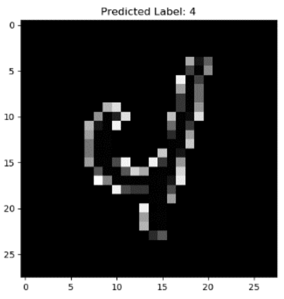


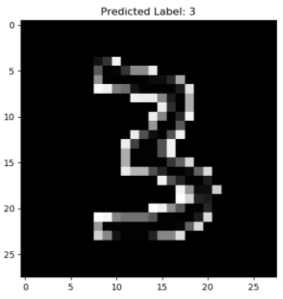


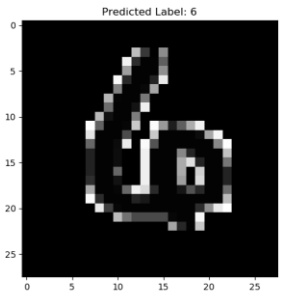


Gamma=10, C=100









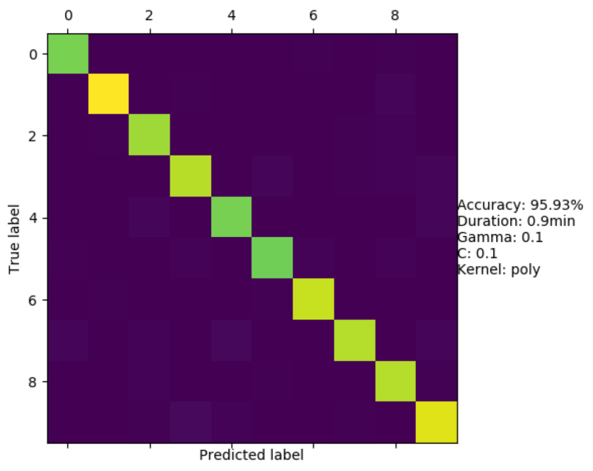
Evaluation of sample images

Most of the sample images above, which were randomly printed by the program, are correctly classified. This reflects the results shown by the confusion matrices and the accuracies, which imply that the performance is always very high when using a linear kernel.

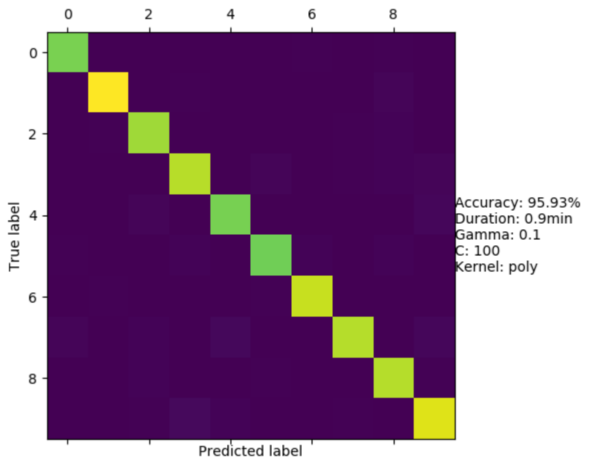
Polynomial Kernel

Confusion Matrices and Accuracy tables

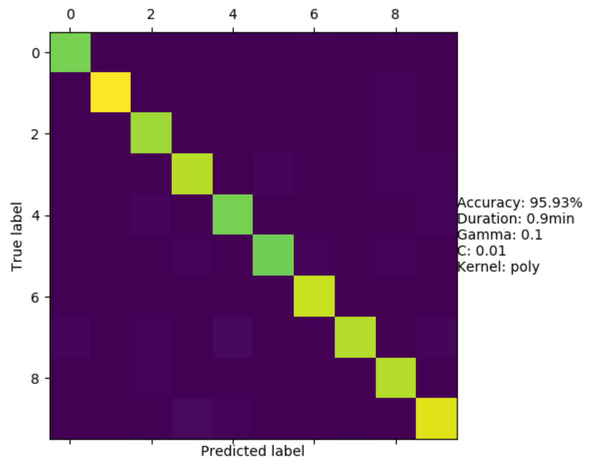
The following are the confusion matrices computed on the validation set, when using a polynomial kernel.



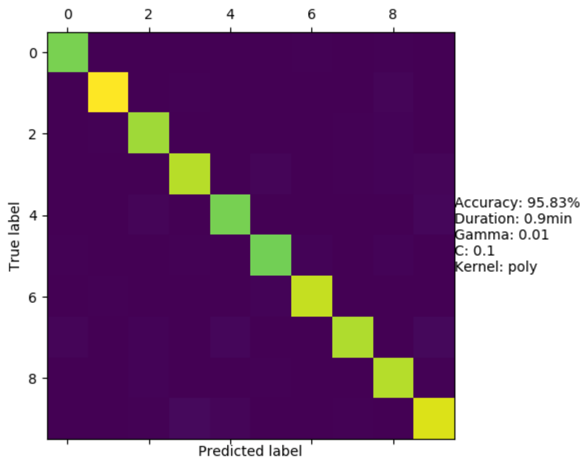
Accuracy: 95.93%
Duration: 0.9min
Gamma: 0.1
C: 0.1
Kernel: poly



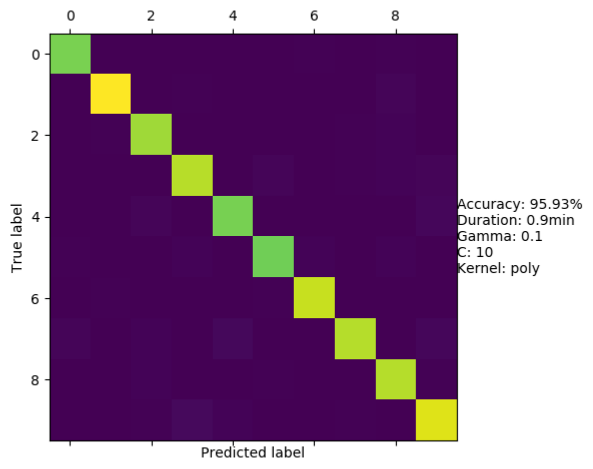
Accuracy: 95.93%
Duration: 0.9min
Gamma: 0.1
C: 100
Kernel: poly



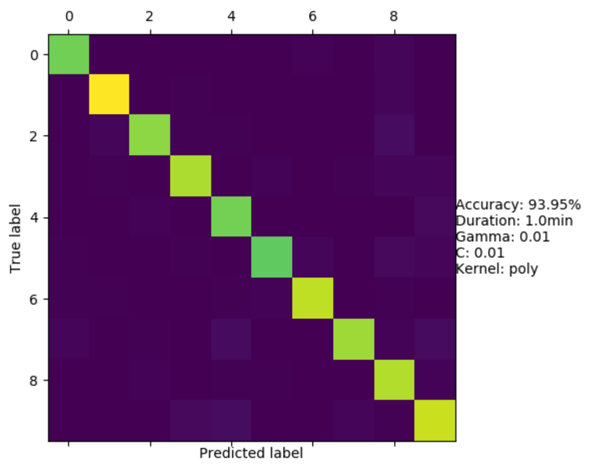
Accuracy: 95.93%
Duration: 0.9min
Gamma: 0.1
C: 0.01
Kernel: poly



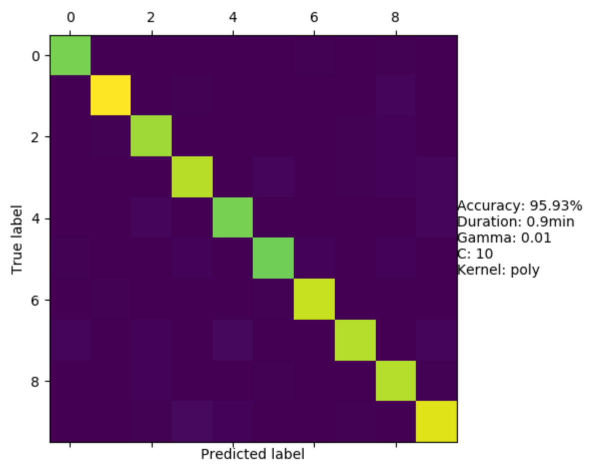
Accuracy: 95.83%
Duration: 0.9min
Gamma: 0.01
C: 0.1
Kernel: poly



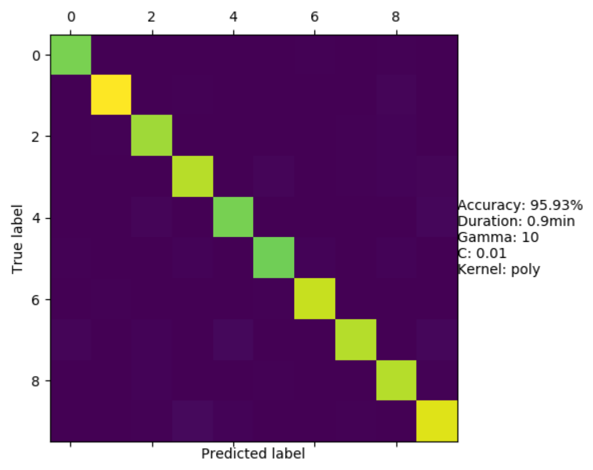
Accuracy: 95.93%
Duration: 0.9min
Gamma: 0.1
C: 10
Kernel: poly



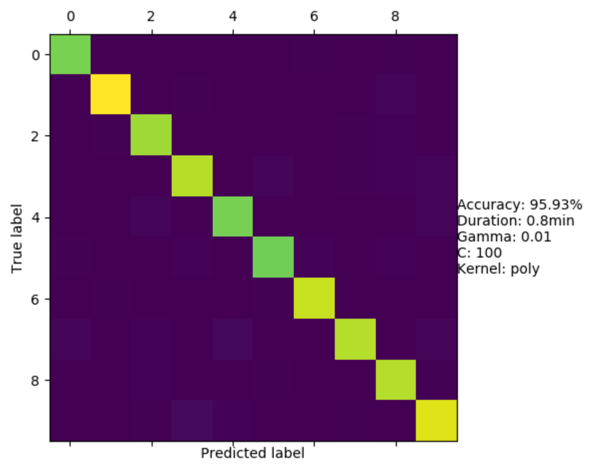
Accuracy: 93.95%
Duration: 1.0min
Gamma: 0.01
C: 0.01
Kernel: poly



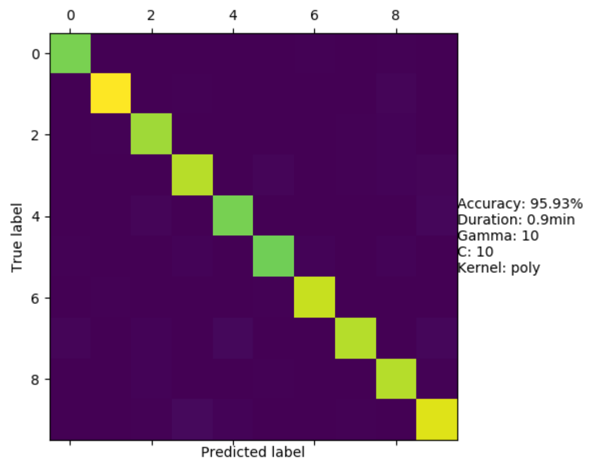
Accuracy: 95.93%
Duration: 0.9min
Gamma: 0.01
C: 10
Kernel: poly



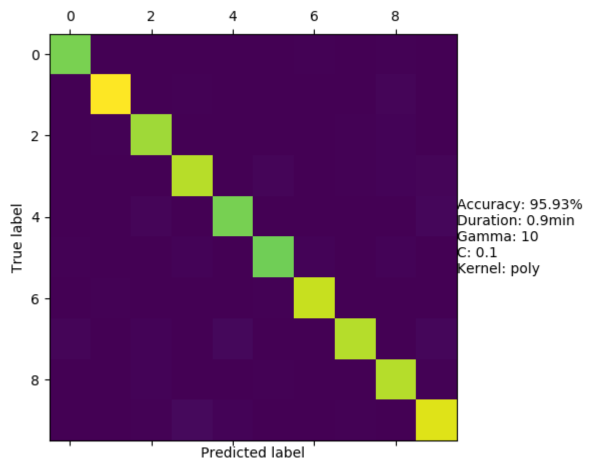
Accuracy: 95.93%
Duration: 0.9min
Gamma: 10
C: 0.01
Kernel: poly



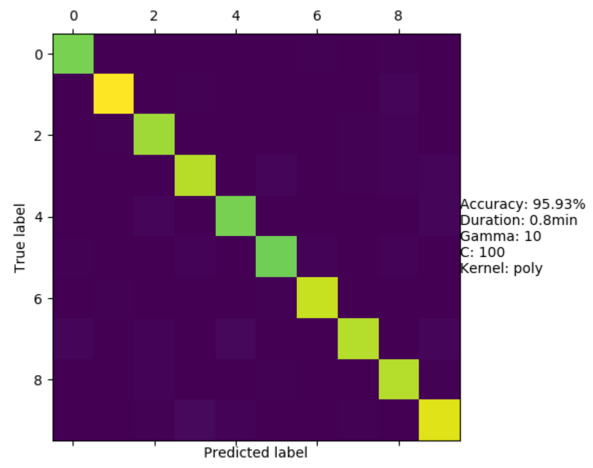
Accuracy: 95.93%
Duration: 0.8min
Gamma: 0.01
C: 100
Kernel: poly



Accuracy: 95.93%
Duration: 0.9min
Gamma: 10
C: 10
Kernel: poly



Accuracy: 95.93%
Duration: 0.9min
Gamma: 10
C: 0.1
Kernel: poly



Accuracy: 95.93%
Duration: 0.8min
Gamma: 10
C: 100
Kernel: poly

The following is a table of the accuracies (in %) of the SVM model on the validation set given different gamma and C parameters.

Gamma C	0.01	0.1	10	100
0.1	95.93	95.93	95.93	95.93
0.01	93.95	95.83	95.93	95.93
10	95.93	95.93	95.93	95.93

The following is a table of the accuracies (in %) of the SVM model on the test set given different gamma and C parameters.

Gamma C	0.01	0.1	10	100
0.1	95.56	95.56	95.56	95.56
0.01	93.74	95.53	95.56	95.56
10	95.56	95.56	95.56	95.56

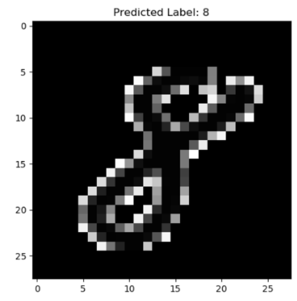
Evaluation of confusion matrices and tables

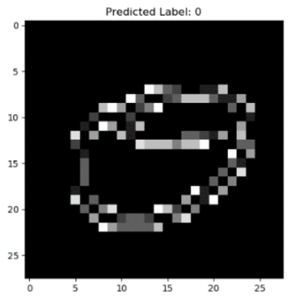
From the two tables above, it is clear that whatever the gamma and the C value are, the accuracy does not change on the whole. In both tables, the least accuracy is found in the case of having both gamma and C equal to 0.01. In both tables, the highest accuracy is over 95%, best accuracy being 95.93% in the first table and 95.56% in the second table. Once again C=10 and C=100 give the same values for all gamma variations. Just like in the linear kernel, the confusion matrices once again display diagonals of green and yellow squares which imply that most predictions were correctly done. When taking into consideration the duration displayed next to confusion matrices, it can be noted that the polynomial kernel needs more time for training than the linear kernel. Nevertheless, the SVM is still not very time consuming when using a polynomial kernel, as the highest duration is 1min.

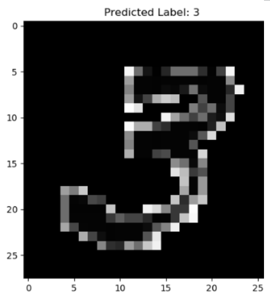
Sample images

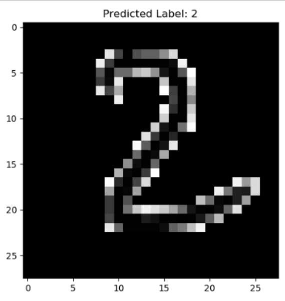
For each combination of variations, four sample images (out of the eight outputted by the program) of the classifications done on the test set are illustrated below.

Gamma = 0.1, C=0.1

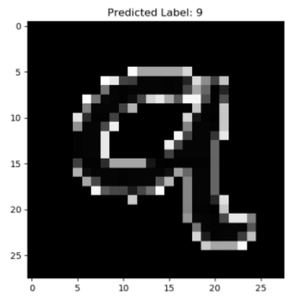


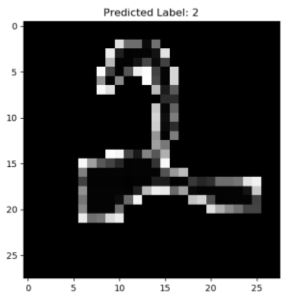


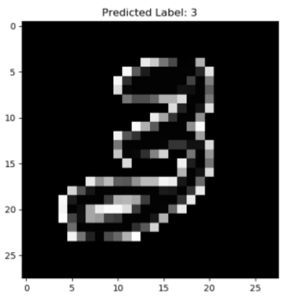


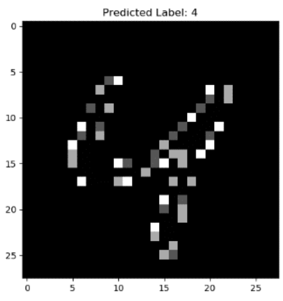


Gamma= 0.1, C = 0.01

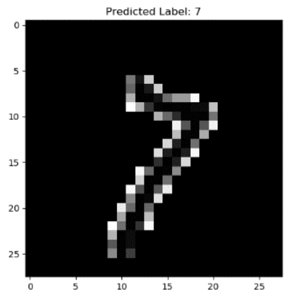


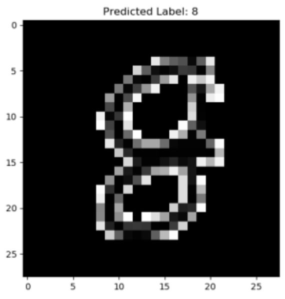


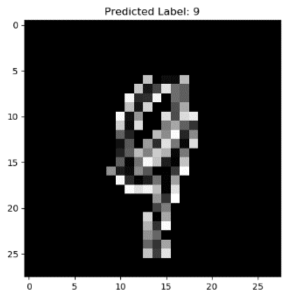


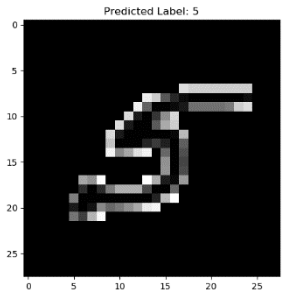


Gamma=0.1, C=10

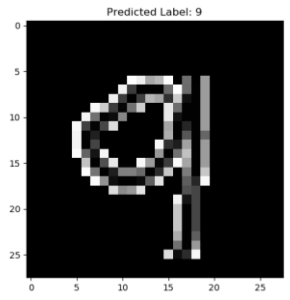


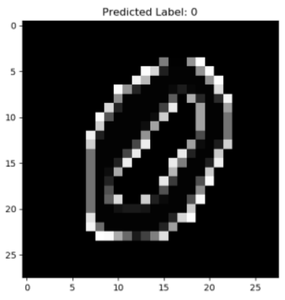


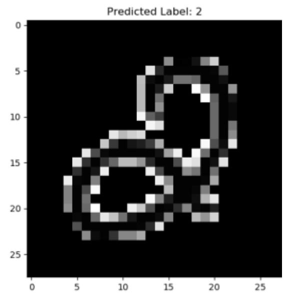


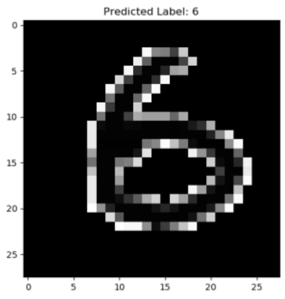


Gamma=0.1, C=100

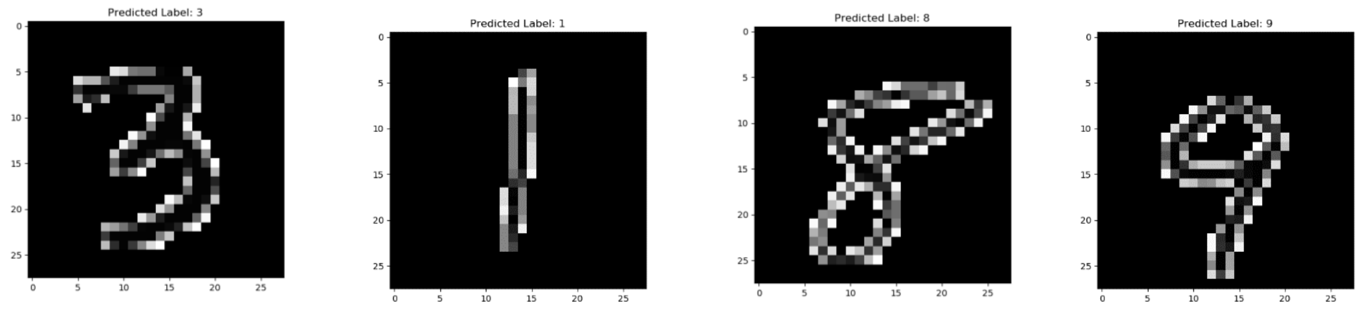




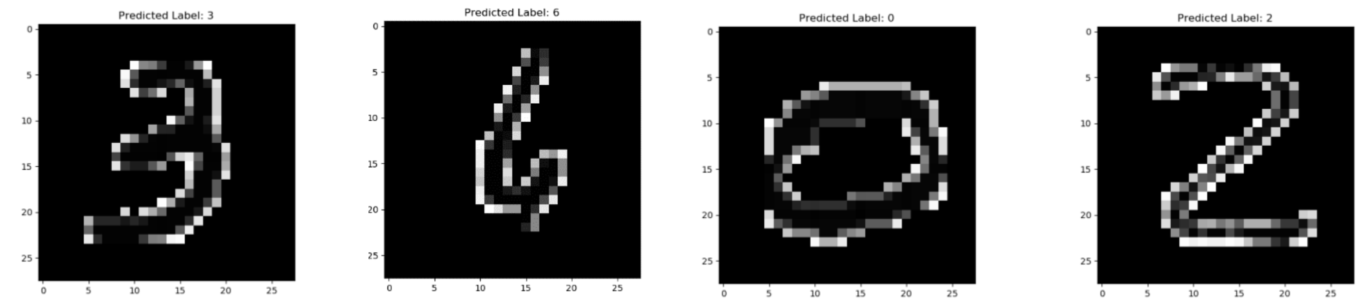




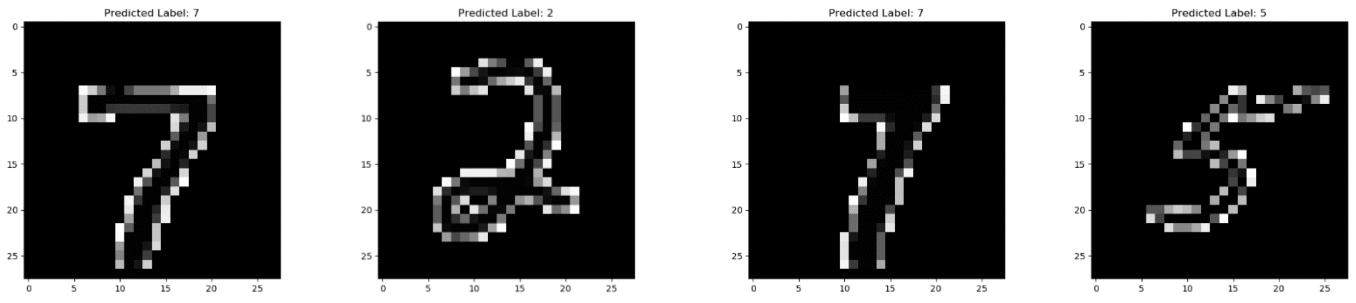
Gamma=0.01, C=0.1



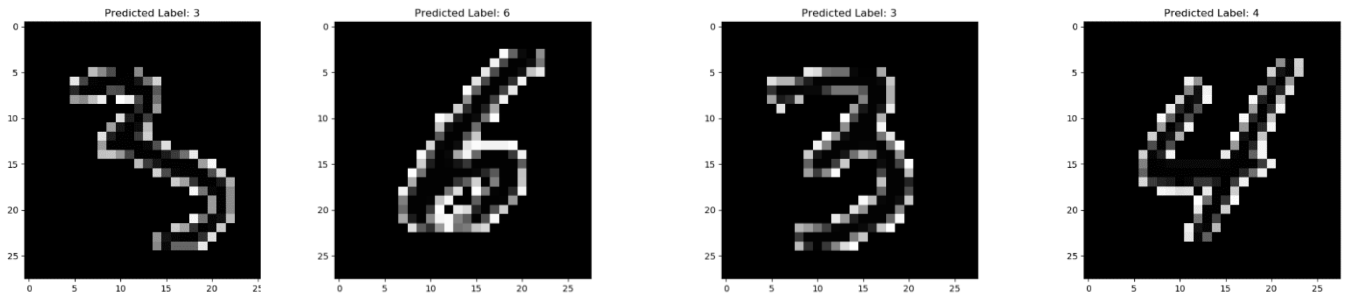
Gamma-0.01, C=0.01



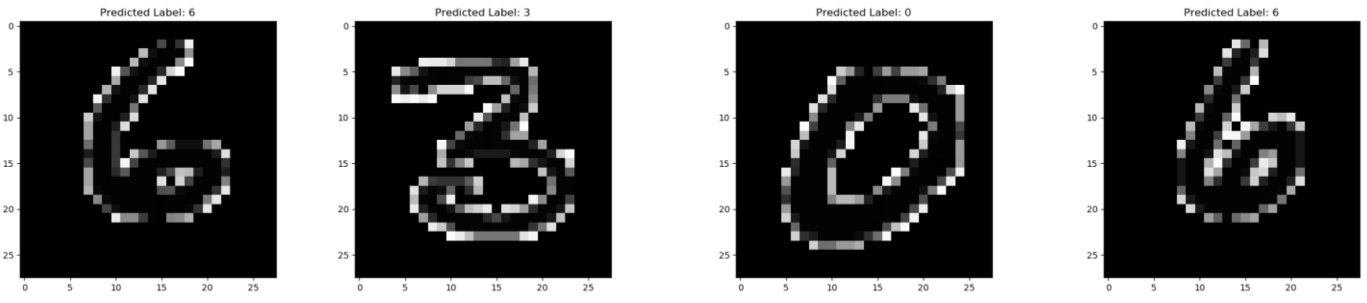
Gamma=0.01, C=10



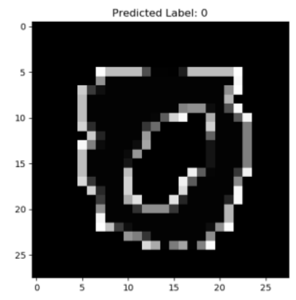
Gamma=0.01, C=100

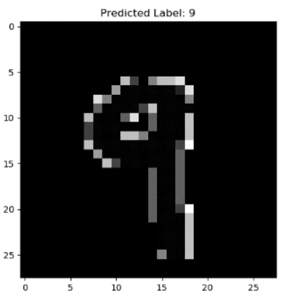


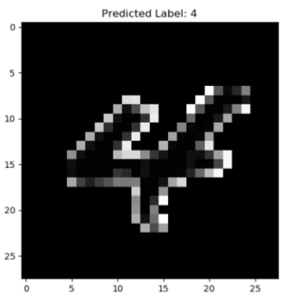
Gamma=10, C=0.1

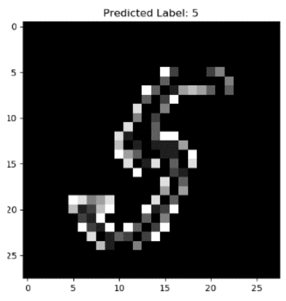


Gamma=10, C=0.01

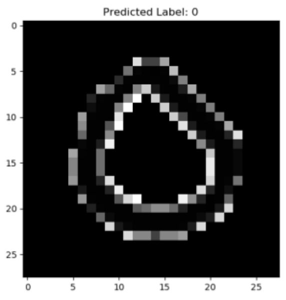


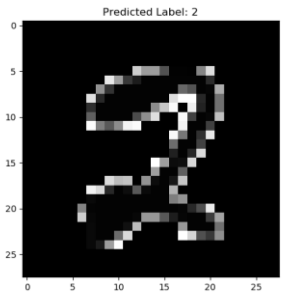


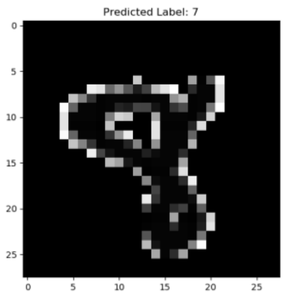


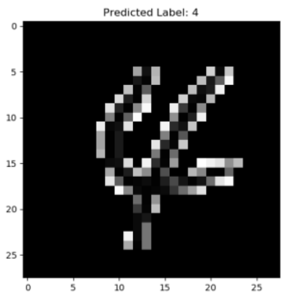


Gamma=10, C=10

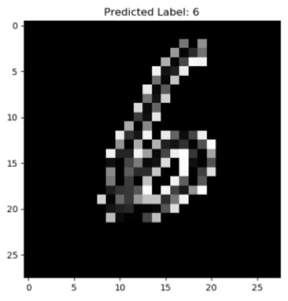


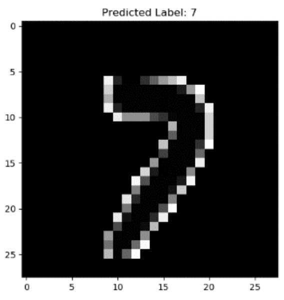


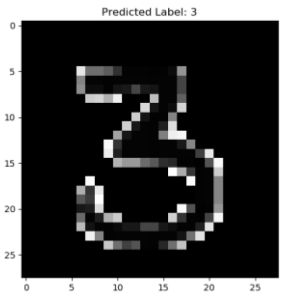


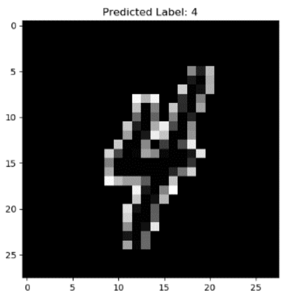


Gamma=10, C=100









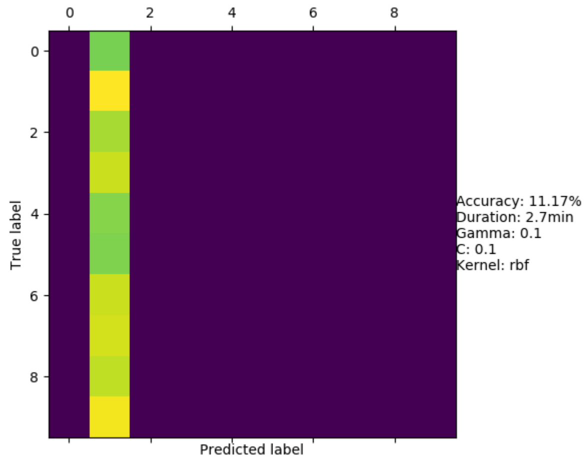
Evaluation of sample images

As discussed in the evaluation of the confusion matrices and accuracy tables, the polynomial kernel provides us with very high accuracies. In fact, all of the random sample images shown above have correct classifications.

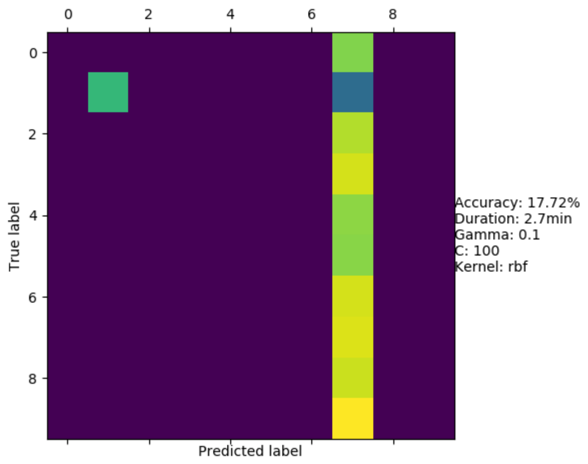
RBF Kernel

Confusion Matrices and Accuracy tables

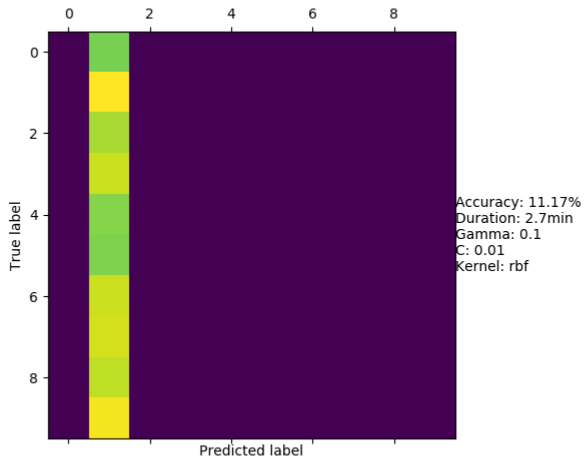
The following are the confusion matrices computed on the validation set, when using the rbf kernel.



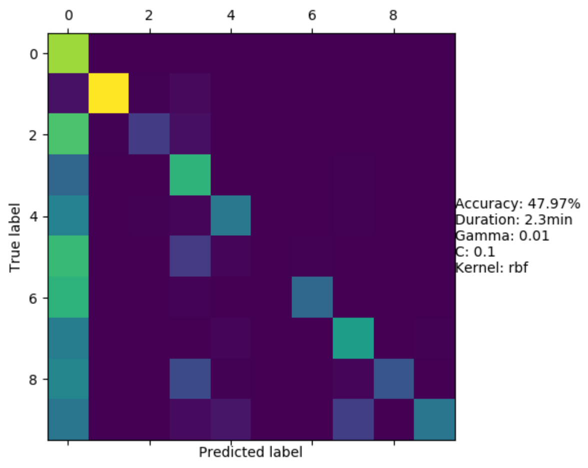
Accuracy: 11.17%
Duration: 2.7min
Gamma: 0.1
C: 0.1
Kernel: rbf



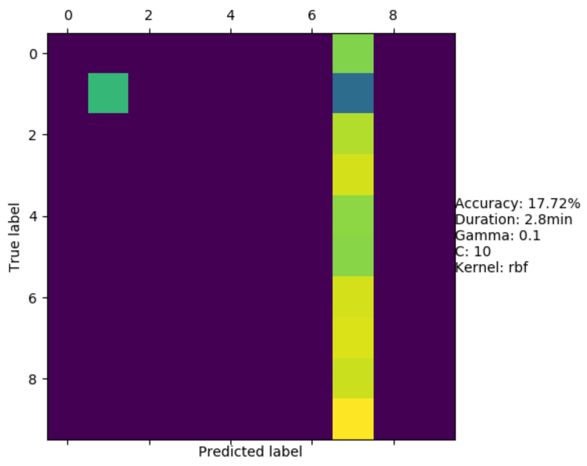
Accuracy: 17.72%
Duration: 2.7min
Gamma: 0.1
C: 100
Kernel: rbf



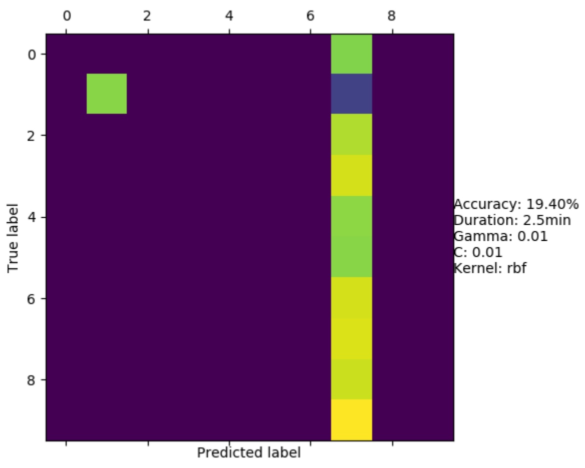
Accuracy: 11.17%
Duration: 2.7min
Gamma: 0.1
C: 0.01
Kernel: rbf



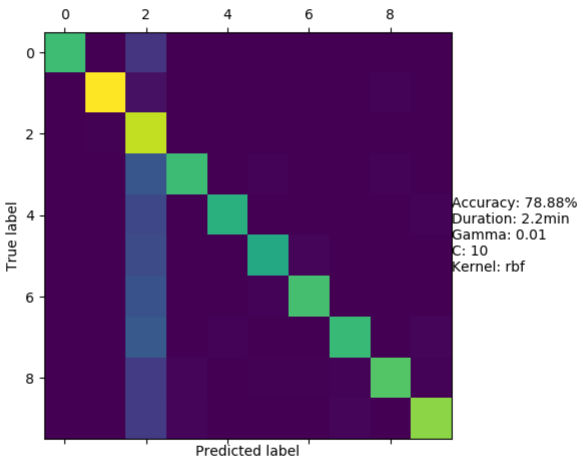
Accuracy: 47.97%
Duration: 2.3min
Gamma: 0.01
C: 0.1
Kernel: rbf

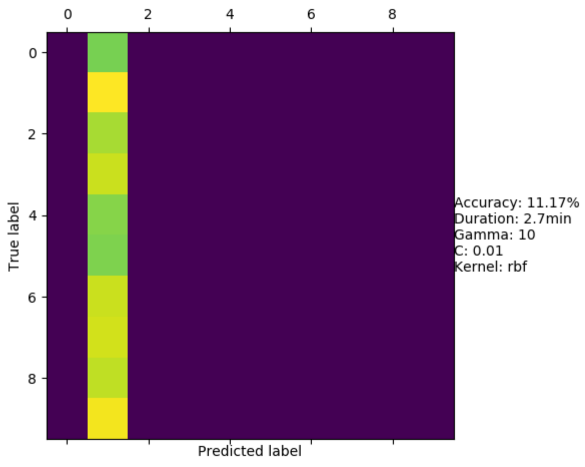


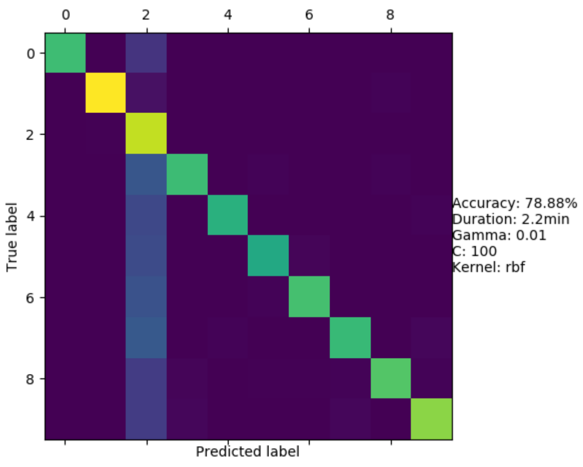
Accuracy: 17.72%
Duration: 2.8min
Gamma: 0.1
C: 10
Kernel: rbf

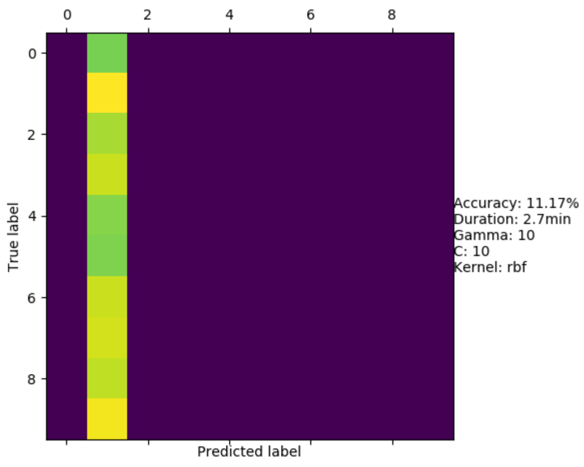


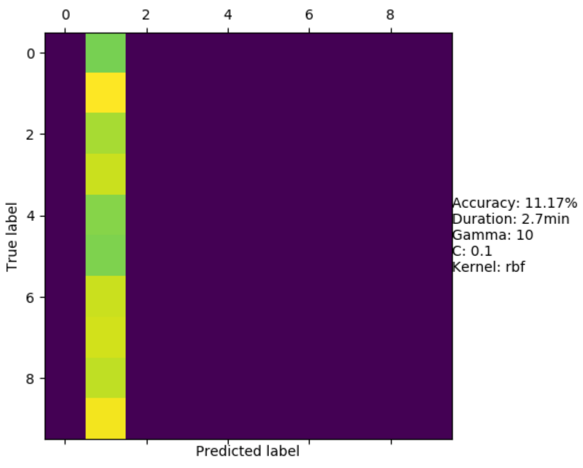
Accuracy: 19.40%
Duration: 2.5min
Gamma: 0.01
C: 0.01
Kernel: rbf

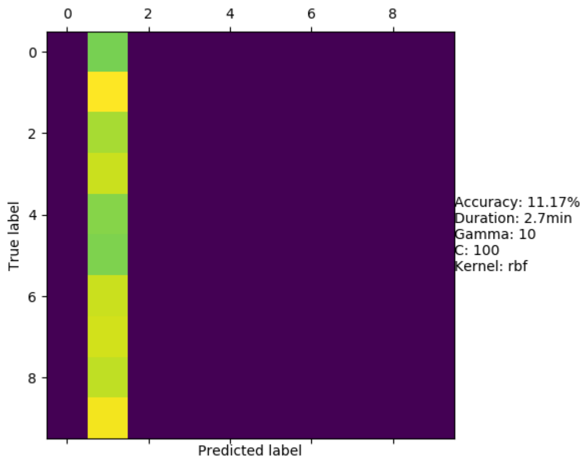












The following is a table of the accuracies (in %) of the SVM model on the validation set given different gamma and C parameters.

Gamma \ C	0.01	0.1	10	100
0.1	11.17	11.17	17.72	17.72
0.01	19.40	47.97	78.88	78.88
10	11.17	11.17	11.17	11.17

The following is a table of the accuracies (in %) of the SVM model on the test set given different gamma and C parameters.

Gamma \ C	0.01	0.1	10	100
0.1	11.35	11.35	17.96	17.96
0.01	19.5	48.18	78.17	78.17
10	11.35	11.35	11.35	11.35

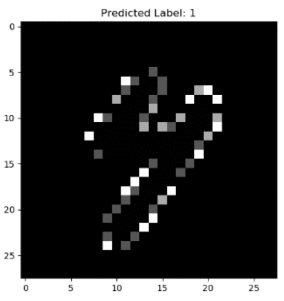
Evaluation of confusion matrices and tables

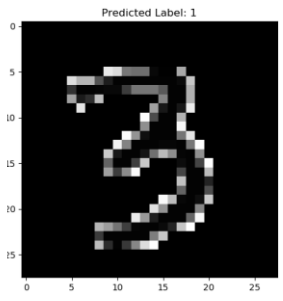
In the two tables, it can be seen that the rbf kernel gives a very low accuracy, the lowest being 11.35% and 11.17%. The gamma value that gives the highest results is 0.01, varying between 19.4% to 78.88% according to the C value inputted. The best results are given by gamma=0.01 and C=10 or C=100. Once more, the regularization values 10 and 100 give the same results for every gamma variation. From the confusion matrices, it is visible that in most cases, a lot of the data was classified to either a 1 or a 7, hence, the very low accuracy. There are only two confusion matrices which show that most of the data was classified into its correct class. As a result, those are the only two cases in which the SVM got the highest accuracy. When analysing the duration of the rbf kernel, it can be noticed that this kernel is the most time consuming of all the kernels tested. As a matter of fact, the shortest duration is 2.2min while the longest is 2.8min.

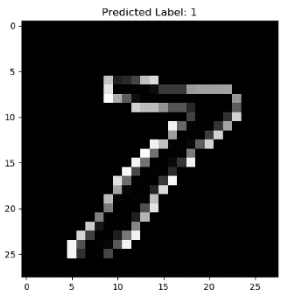
Sample images

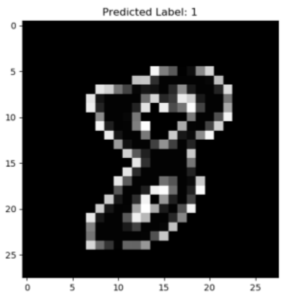
For each combination of variations, four sample images (out of the eight outputted by the program) of the classifications done on the test set are illustrated below.

Gamma = 0.1, C=0.1

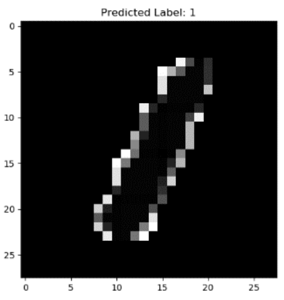


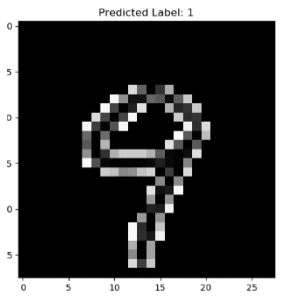


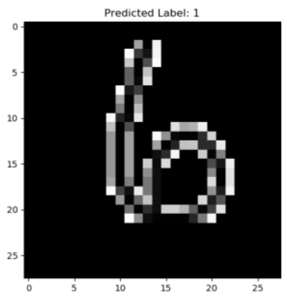


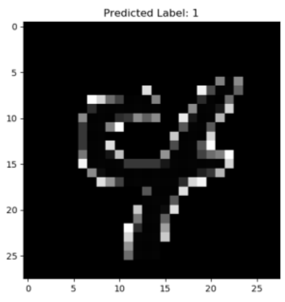


Gamma= 0.1, C = 0.01

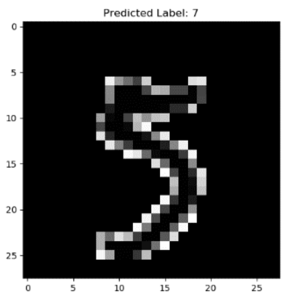


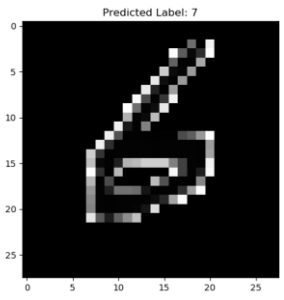


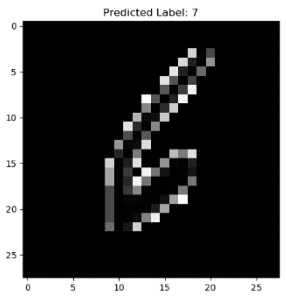


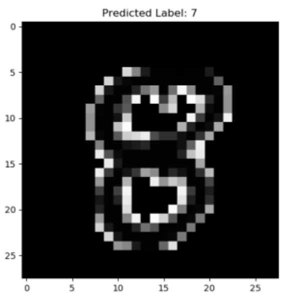


Gamma=0.1, C=10

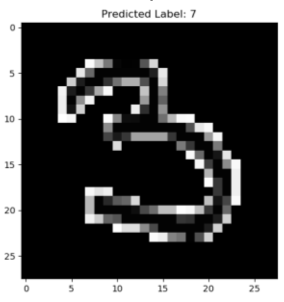


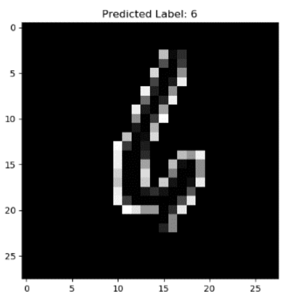


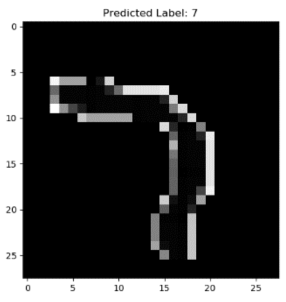


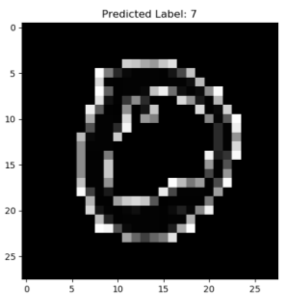


Gamma=0.1, C=100

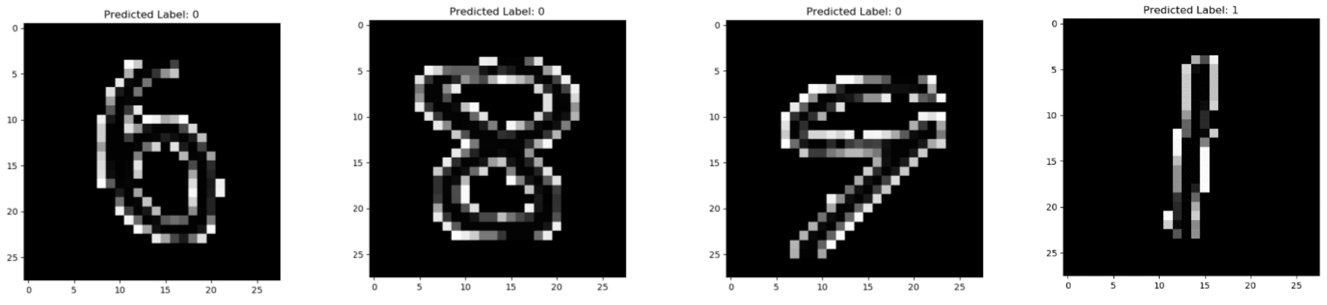




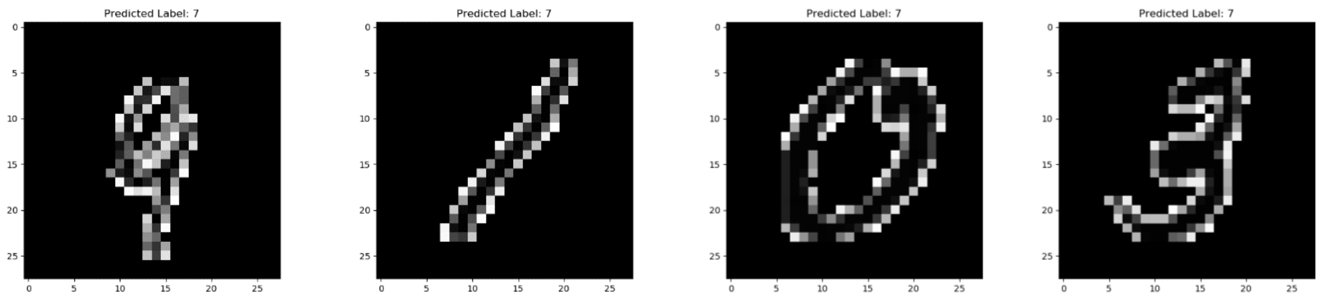




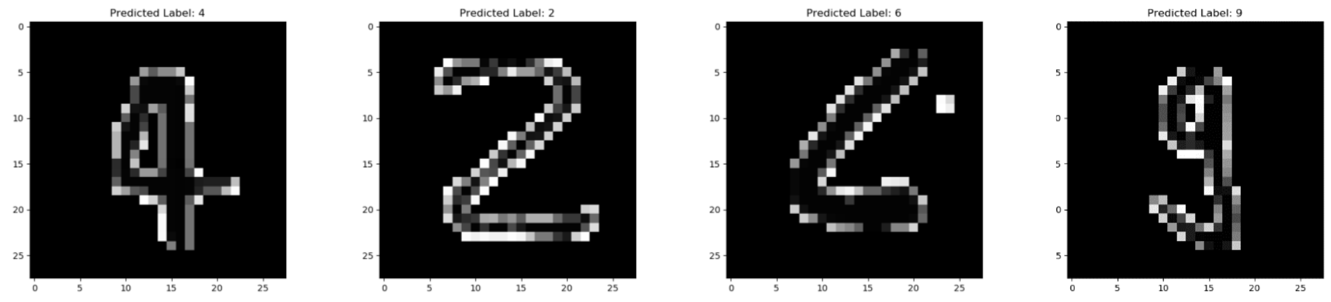
Gamma=0.01, C=0.1



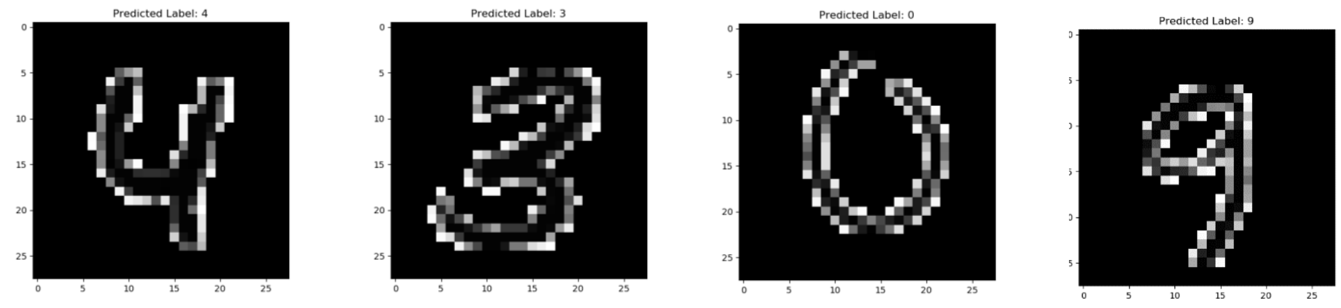
Gamma-0.01, C=0.01



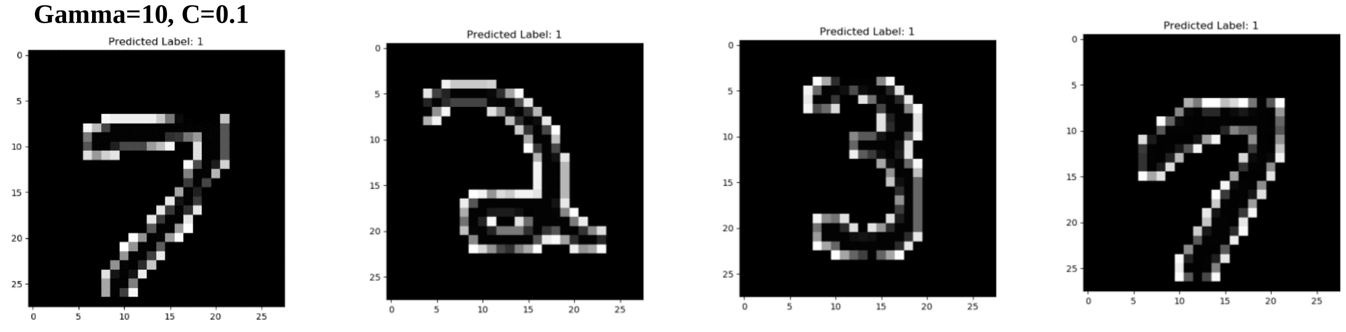
Gamma=0.01, C=10



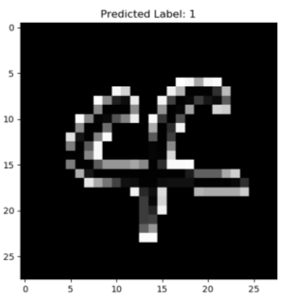
Gamma=0.01, C=100

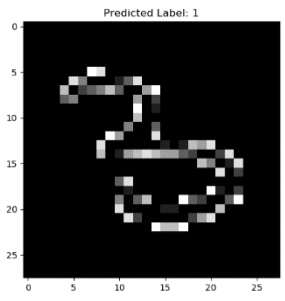


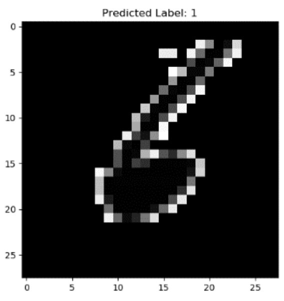
Gamma=10, C=0.1

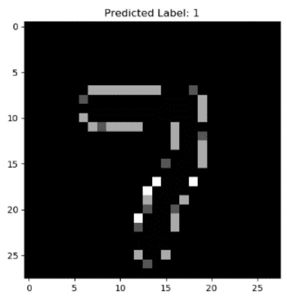


Gamma=10, C=0.01

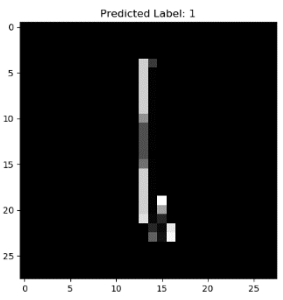


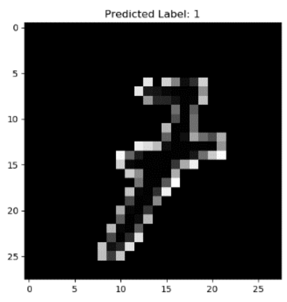


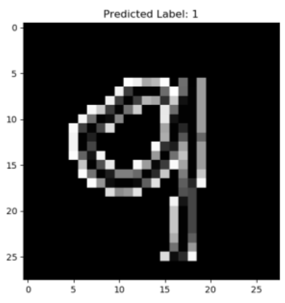


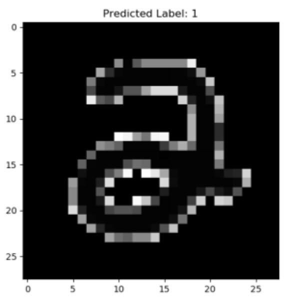


Gamma=10, C=10

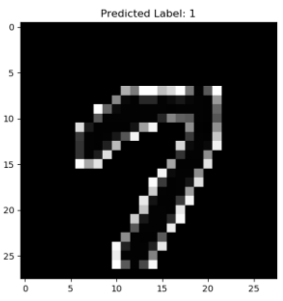


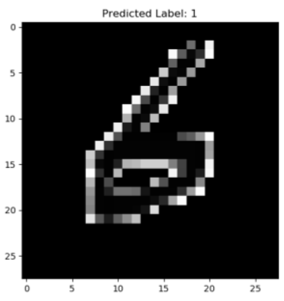


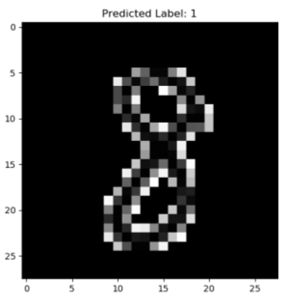


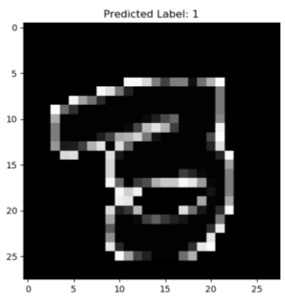


Gamma=10, C=100









Evaluation of sample images

The sample images above, mirror the results shown by the confusion matrices. This is because, in the majority of cases, the numbers are misclassified, and the main predictions are 1 and 7. Moreover, the only times in which all the four sample images have the correct prediction are when gamma=0.01 and C=10 or 100, which also reflect the results previously discussed.

Comparing results of all three kernels

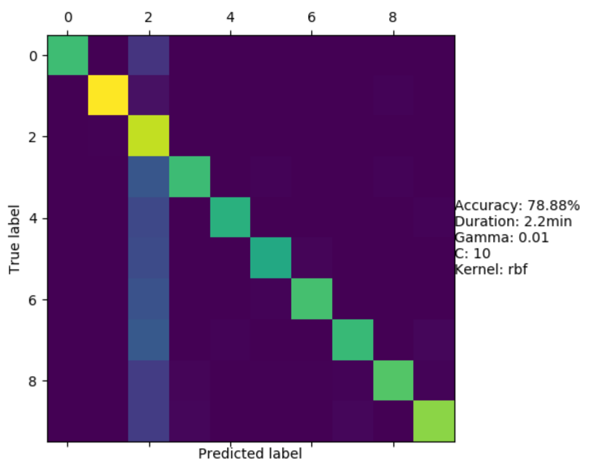
The below is a table containing the model's accuracy (in %) for each kernel it was tested on.

Linear Kernel				
Gamma C	0.01	0.1	10	100
0.1	93.52	91.87	91.44	91.44
0.01	93.52	91.87	91.44	91.44
10	93.52	91.87	91.44	91.44
Polynomial Kernel				
Gamma C	0.01	0.1	10	100
0.1	95.93	95.93	95.93	95.93
0.01	93.95	95.83	95.93	95.93
10	95.93	95.93	95.93	95.93
RBF Kernel				
Gamma C	0.01	0.1	10	100
0.1	11.17	11.17	17.72	17.72
0.01	19.40	47.97	78.88	78.88
10	11.17	11.17	11.17	11.17

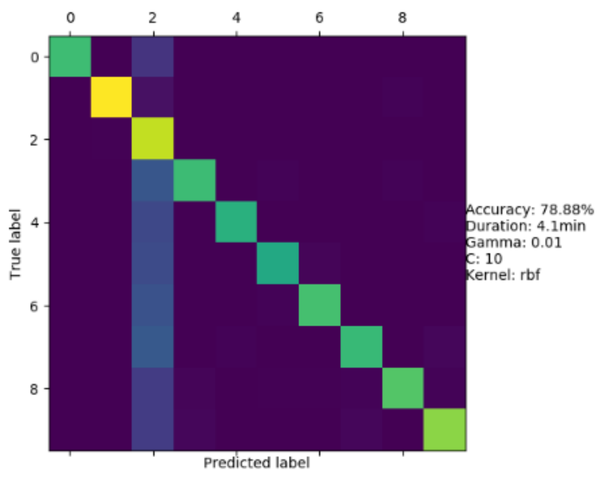
From this table it can be concluded that, the polynomial gives the best results in terms of accuracy. It is followed by the linear kernel, which provides competitive accuracies but still not as good as the polynomial kernel's. Finally, the rbf kernel gives the worst accuracies which are not in any way competitive with the other two kernels. In all cases, the accuracy does not get any different when the regularization value is more than 10. The following is a table is derived from the above table, containing the highest results for each kernel, and the parameters used to get them to give a clearer picture.

	Highest Accuracy (in %)	Gamma	C
Linear Kernel	93.52	0.1, 0.01, 10	0.01
Polynomial Kernel	95.93	0.01, 10, 10	10, 100 (all when gamma =0.1 or 10)
RBF Kernel	78.88	0.01	10, 100

When analysing the duration each variation took, it is notable that the linear kernel generally takes the shortest time, followed by the polynomial kernel which takes a bit more time, and last but not least, the rbf kernel takes the longest time of all three. One should keep in mind that, the duration of the SVM model may vary from one computer to another according to the device's speed for computation. In fact, the durations shown in this documentation were outputted by a rather powerful PC, and when tested on a slower PC the durations increased. An example of this is shown in the Figures below, having the image on the left outputted by a more powerful PC, while the image on the right outputted by a less powerful PC. The parameters are the same in both Figures, which gave the same accuracy, which means that the results are not affected by the different use of devices, but only the duration is.



Accuracy: 78.88%
Duration: 2.2min
Gamma: 0.01
C: 10
Kernel: rbf



Accuracy: 78.88%
Duration: 4.1min
Gamma: 0.01
C: 10
Kernel: rbf

Further experimentation using the best pairs of parameters

As previously stated, the training set was modified to contain 12,100 examples instead of 60,000 training examples. After all the experimentations were done and analysed, it was decided that the algorithm should be run again but on the whole training set (i.e. all 60,000 training examples), to analyse if the accuracy increases or not. For each kernel type, this experimentation is only done using one of the best pair of parameters, taken from the table displayed in the previous section. The bar charts below show the distribution of the data in the training set. The distributions do not vary much, but for obvious reasons the frequency of each class is higher in the large data set.

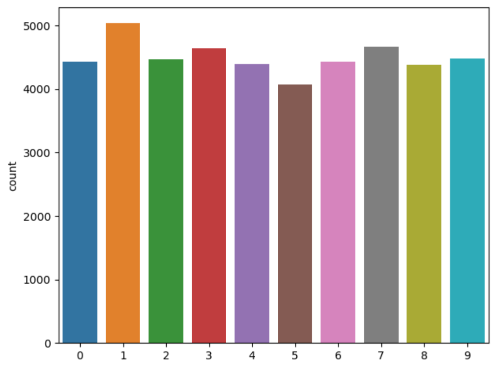


Figure 25 The bar chart of the whole data set

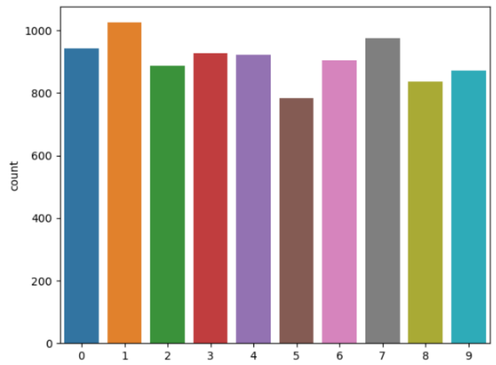
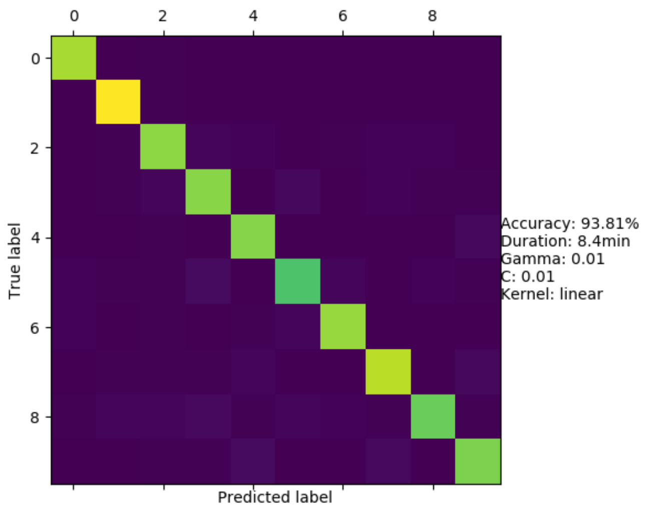


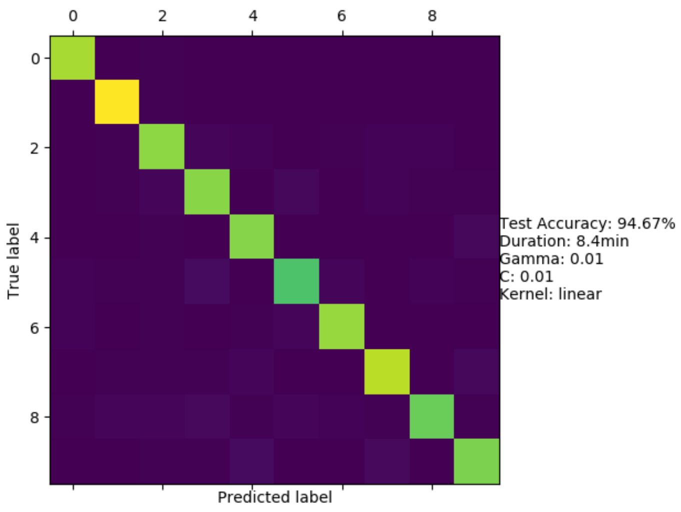
Figure 24 The bar chart of the smaller data set

Linear Kernel

For this kernel, the parameters chosen to rerun the SVM model with are: gamma=0.01 and C=0.01, as they are one of the pairs that gave the best accuracy when trained on a smaller dataset. The Figures below are the confusion matrices outputted by the program when trained and tested on the whole dataset.



Accuracy: 93.81%
Duration: 8.4min
Gamma: 0.01
C: 0.01
Kernel: linear



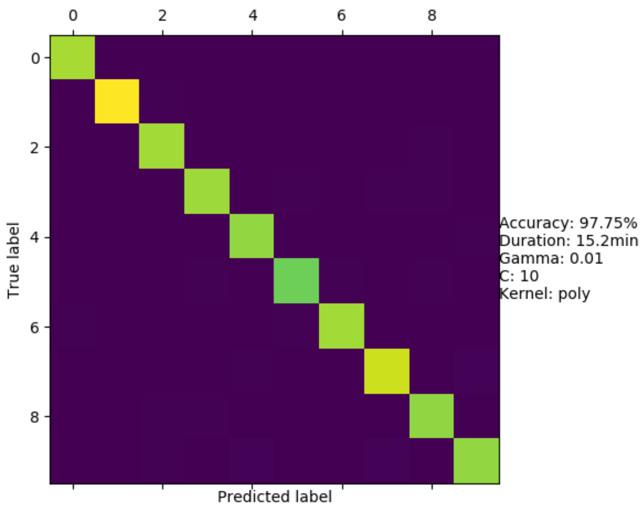
Test Accuracy: 94.67%
Duration: 8.4min
Gamma: 0.01
C: 0.01
Kernel: linear

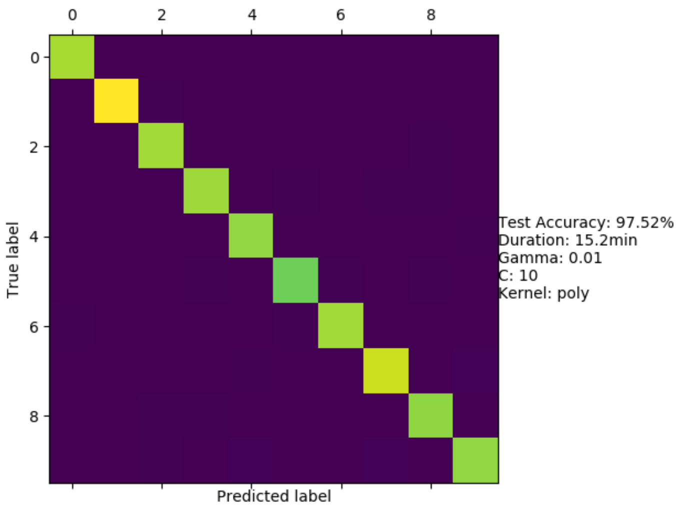
	Trained on 12,100 training examples		Trained on 60,000 training examples	
	Accuracy on Validation Set	Accuracy on Test Set	Accuracy on Validation Set	Accuracy on Test Set
Gamma =0.01 C=0.01	93.52	92.78	93.81	94.67

The table above compares the accuracies (in %) obtained when training a linear kernel on a relatively smaller dataset, with the accuracies obtained on the entire dataset. It is noticeable that the accuracies increased, mostly when testing the SVM model on the test set, as it increased by 1.8%.

Polynomial Kernel

For this kernel, the parameters chosen to rerun the SVM model with are: gamma=0.01 and C=10, as they are one of the pairs that gave the best accuracy when trained on a smaller dataset. The Figures below are the confusion matrices outputted by the program when trained and tested on the whole dataset.



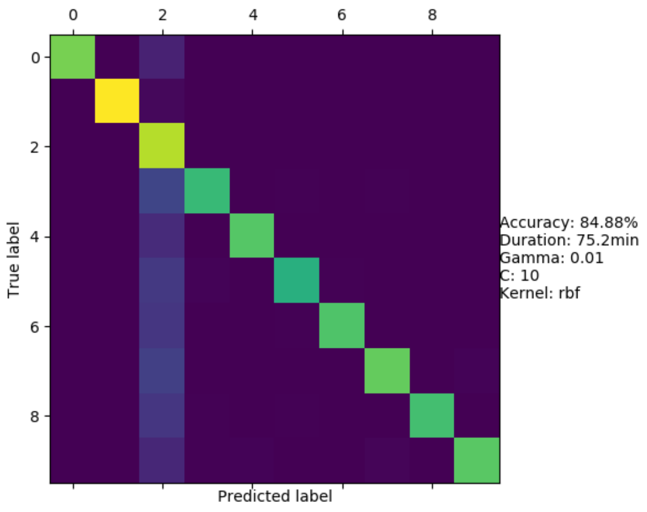


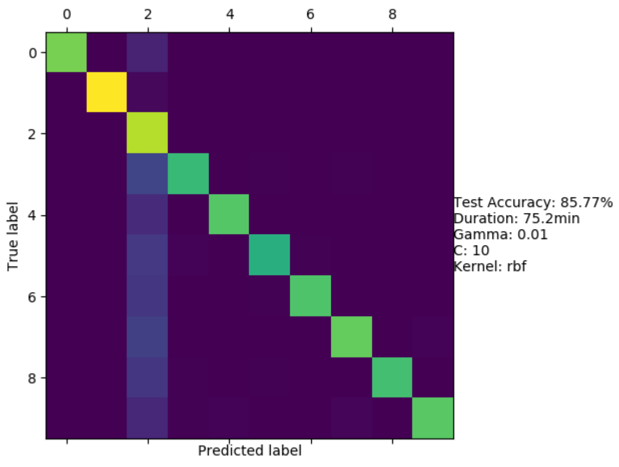
	Trained on 12,100 training examples		Trained on 60,000 training examples	
	Accuracy on Validation Set	Accuracy on Test Set	Accuracy on Validation Set	Accuracy on Test Set
Gamma =0.01 C=10	95.93	95.56	97.75	97.52

The table above compares the accuracies (in %) gained when training the SVM with a polynomial kernel on the smaller dataset, with the accuracies obtained on the entire dataset. In both cases, the accuracy increased with more than 1%, having the accuracy on the validation set increasing by 1.82% and the accuracy on the test set increasing by 1.96%.

RBF Kernel

For this kernel, the parameters chosen to rerun the SVM model with are: gamma=0.01 and C=10, as they are one of the pairs that gave the best accuracy when trained on a smaller dataset. The Figures below are the confusion matrices outputted by the program when trained and tested on the whole dataset.





	Trained on 12,100 training examples		Trained on 60,000 training examples	
	Accuracy on Validation Set	Accuracy on Test Set	Accuracy on Validation Set	Accuracy on Test Set
Gamma =0.01, C=0.01	78.88	78.17	84.88	85.77

The table above compares the accuracies (in %) gained when training the SVM with the rbf kernel on the smaller dataset, with the accuracies obtained on the entire dataset. Training on a larger dataset surely impacted the SVM's performance in this situation, as the accuracies have a very prominent increment. In fact, when compared to the other two kernels, this one has the biggest difference between the first accuracies and the last accuracies. The accuracy on the validation set is increased by 6%, while the accuracy on the test set is increased by 7.6%.

Overall Conclusions

In this assignment, the SVM model has been studied and experimented with. An SVM is a model used in machine learning used for classification of data or regression. From the research carried out it was understood that an SVM has three parameters, kernel, gamma and regularization, which should be fine-tuned to the problem to gain the best results. Each kernel represents a function which transforms the data accordingly, and handle non-linearity.

The support vector machine model was compared and contrasted to five other classification techniques which are: Logistic Regression (LR), K-Nearest Neighbour (KNN), Decision Trees, Random Forest and Convolutional Neural Networks (CNN) alongside Neural Networks in general. In every case, one algorithm should be preferred over the other based on the problem given as each technique has its own pros and cons. For example, one technique might be faster than the SVM, but at the same time the SVM model can gain a higher accuracy than that same technique.

This documentation contains also an explanation of the artefact's code and datasets used, together with one of the outputs as an example. This is then followed by the analysis of the experiments done on the SVM using different variations of parameters, firstly using 12,100 training examples, and then using all the 60,000 training examples on the best parameters. From these experiments, it could be concluded that the polynomial kernel gave the best performance on the MNIST dataset, while the rbf kernel performed the worst. This was consistent even when the SVM was trained and tested on the whole dataset, having the polynomial kernel reaching 97.75% accuracy and the rbf kernel reaching 85.77%. For all kernels, the percentage of the accuracy increased after training the SVM model on the whole dataset. It should also be kept in mind that the durations shown in this documentation may vary according to the device being used.

References

[1] S. Patel, “Chapter 2: Svm (support vector machine) — theory.” https://medium.com/machine-learning-101/chapter-2-svm-support-vector-machine-theory-f0812effc72, accessed on 02-01-2020.
[2] R. Gandhi, “Support vector machine — introduction to machine learning algorithms.” https://towardsdatascience.com/support-vector-machine-introduction-to-machine-learning-algorithms-934a444fca47, Last accessed on 02-01-2020.
[3] C.-F. Lin and S.-D. Wang, “Fuzzy support vector machines,” IEEE transactions on neural networks, vol. 13, no. 2, pp. 464–471, 2002.
[4] R. Berwick, “An idiot’s guide to support vector machines (svms),”
[5] S. Ray, “Understanding support vector machine algorithm from examples (along with code).” https://www.analyticsvidhya.com/blog/2017/09/understaing-support-vector-machine-example-code/, Last accessed on 02-01-2020.
[6] G. Drakos, “Support vector machine vs logistic regression.” https://medium.com/@george.drakos62/support-vector-machine-vs-logistic-regression-94cc2975433f?, Last accessed on 02-01-2020.
[7] M.-W. Chang, “Introduction to logistic regression and support vector machine,” 2009.
[8] R. Kompella, “Support vector machines (intuitive understanding) — part1.” https://towardsdatascience.com/support-vector-machines-intuitive-understanding-part-1-3fb049df4ba1, Last accessed on 03-01-2020.
[9] “Logistic regression: Loss and regularization.” https://developers.google.com/machine-learning/crash-course/logistic-regression/model-training, Last accessed on 03-01-2020.
[10] “Confusion on hinge loss and svm.” https://stats.stackexchange.com/questions/372999/confusion-on-hinge-loss-and-svm, Last accessed on 03-01-2020.
[11] J. Kim1, B. Kim, and S. Savarese, “Comparing image classification methods: K-nearest-neighbor and support-vector-machines,” in Proceedings of the 6th WSEAS international conference on Computer Engineering and Applications, and Proceedings of the 2012 American conference on Applied Mathematics, vol. 1001, pp. 48109–2122, 2012.
[12] A. Goel and S. Mahajan, “Comparison: Knn & svm algorithm,” International Journal for Research in Applied Science & Engineering Technology (IJRASET), vol. 5.
[13] “Comparative study on classic machine learning algorithms.” https://towardsdatascience.com/comparative-study-on-classic-machine-learning-algorithms-24f9ff6ab222, Last accessed on 04-01-2020.
[14] H. H. Nguyen, “A complete view of decision trees and svm in machine learning.” https://towardsdatascience.com/a-complete-view-of-decision-trees-and-svm-in-machine-learning-f9f3d19a337b, Last accessed on 06-01-2020.
[15] H. Shafri and F. Ramle, “A comparison of support vector machine and decision tree classifications using satellite data of langkawi island,” Information Technology Journal, vol. 8, no. 1, pp. 64–70, 2009.
[16] T. Yiu, “Understanding random forest.” https://towardsdatascience.com/understanding-random-forest-58381e0602d2, Last accessed on 06-01-2020.
[17] “When to use random forest over svm and vice versa?.” https://datascience.stackexchange.com/questions/6838/when-to-use-random-forest-over-svm-and-vice-versa, Last accessed on 06-01-2020.
[18] A. F. Agarap, “An architecture combining convolutional neural network (cnn) and support vector machine (svm) for image classification,” arXiv preprint arXiv:1712.03541, 2017.
[19] M. Jeeva, “The scuffle between two algorithms -neural network vs. support vector machine.” https://medium.com/analytics-vidhya/the-scuffle-between-two-algorithms-neural-network-vs-support-vector-machine-16abe0eb4181, Last accessed on 08-01-2020.