**Supplementary Materials**

svmTemplate = templateSVM('KernelFunction', 'linear', 'BoxConstraint', 1);

SVMModel = fitcecoc(X\_train\_subset, y\_train\_subset, 'Learners', svmTemplate, 'Coding', 'onevsall');

A red and black graph with numbers and a red and black diagram

Description automatically generated with medium confidence

As you can see over here, only one class has been predicted. This is an error and we re-run the code and redefine everything to get the proper result.

**Glossary**

**SVM Model** – Support Vector Machines are a supervised machine learning algorithm which works by forming hyperplanes between data points to distinguish between different features/ classes.

**KNN Model –** KNN is a “lazy” easy to implement supervised machine learning algorithm which works by distributing all points in a space grouping them based on their features in memory. Once a new datapoint comes in “c”, the nearest neighbours of c are identified, and the data point is grouped.

**K –** The K parameter symbolises the number of K neighbours which are close to “c” to palace it within a class. This is for a KNN

**C –** A hyperparameter of the SVM which represents the marginality of the soft margin allowing for misclassifications. A higher C indicates a smaller margin and allows for less misclassifications within the data.

**Kernal Function –** This is a hyperparameter of the SVM which is used in the “Kernel Trick” that takes a data point and computes the dot product taking it to a higher dimensional space to form the hyperplane.

**Linear-** A linear boundary is an option of the hyperparameter “Kernel Function” which specifies no requirement to take the data points to a higher dimensional space; they are linearly separable.

**Polynomial –** This implicitly maps data points into a higher dimensional space, the polynomial kernel allowing SVMs to capture complex relationships which might not be linear.

**RBF (Radial Basis Function) –** It is a powerful kernel that transforms data points into an infinite-dimensional space and is used for non-linearly separable data.

**Learner Methods – ONEvsALL –** A classification strategy used to train multiple binary classifiers focusing on distinguishing one class from the rest of classes and is useful for multiclassification data.

**Cohen Kappa Coefficient –** A statistic that measures the agreement between two raters or observers who are assessing the same items or subjects. This is useful for evaluating the reliability of classifications. 1 indicated perfect agreement, 0 indicated hardly any agreement and less than 0 indicates worse than chance.

**Implementation Details**

**Other Model Architectures:**

Naïve Bayes: We did not choose this model architecture, but this was the most considered. We go with KNN and SVMs because these are very different algorithms, both suited to complex data, and so we thought it’ll be a more interesting comparison. These classifiers are designed for categorical data, and modelling continuous pixel values could be challenging.

Decision Tree: There are so many ways of exploring this machine learning algorithm with techniques such as bagging and boosting. However, we decided this may not be the best approach for higher dimensional data as the MNIST dataset consists of lots of images of handwritten digits, represented as a grid of pixels. This makes the decision tree very complex potentially resulting in overfitting. Decision Trees are more suited to datasets with a smaller number of features.

**Parameterisations:**

KNN Distance Metric Hyperparameter: There are a multiple of ways the KNN can calculate the distance between a new data point “c” and the k nearest neighbour points in the plane. Examples include Euclidean distance, Minkowski distance, Manhattan distance. We decide to stick with Euclidean distance and not perform hyperparameter tuning on this due to time constraints.

SVM Learner Method Hyperparameter: There are different learner methods that we use to train the SVM on classifying the MNIST dataset. Examples include binaryclassification and ternaryclassification. We stick with onevsall as we have 9 different classes.

**Issues of Errors:**

% Example 1: Linear SVM

svmModelLinear = fitcsvm(X, Y, 'KernelFunction', 'linear');

One error included having to changed the fitcsvm to fitcoec because there are 9 different classes, so the “Error Correcting Output Codes” variation is better for binary classifies like SVM to handle multi classifications.

SVMModel = fitcecoc(X\_train\_subset, y\_train\_subset, 'Learners', 'svm', 'Coding', 'onevsall');

Another error involved changing the dataframe loaded into a table into an array. Models cannot ready data from tables and this image data had to be changed to arrays to work.

X\_train = table2array(X\_train);

y\_train = table2array(y\_train);

X\_test = table2array(X\_test);

y\_test = table2array(y\_test);

Finally, the last error involved having to create a subset of the data for the algorithms to test on, especially for the SVM, because it took to long to train the SVM on 60 images.

num\_samples\_train = 5000; % Number of training samples

num\_samples\_test = 1000; % Number of test samples

random\_indices\_train = randperm(size(X\_train, 1), num\_samples\_train);

X\_train\_subset = X\_train(random\_indices\_train, :);

y\_train\_subset = y\_train(random\_indices\_train, :);

random\_indices\_test = randperm(size(X\_test,1), num\_samples\_test);

X\_test\_subset = X\_test(random\_indices\_test, :);

y\_test\_subset = y\_test(random\_indices\_test, :);

SVMModel = fitcecoc(X\_train\_subset, y\_train\_subset, 'Learners', 'svm', 'Coding', 'onevsall');