Sims_Kelly_HW5_report

November 7, 2020

Q1

SVM

(a) Explain why can we set the margin c=1 to derive the SVM formulation

Answer Given that the optimization problem for SVM is

$$max_{w,b}\frac{2c}{||w||}$$

subject to

$$y^i(w^T x^i + b) \geq c, \forall i$$

We are able to apply a constant to the constraint without changing the overall problem. E.g. dividing **w** and **b** by a factor of **a**

$$y^i(\frac{w^T x^i}{a} + \frac{b}{a}) \geq c \qquad (1) = y^i(w^T x^i + b) \geq (c * a) \qquad (2)$$

Where **a** is some arbitrary constant. Since dividing **w** and **b** by constant **a** (1) is equivalent to multiplying **c** by constant **a**, we can choose **a** to be

$$\frac{1}{c}$$

Hence

$$c * \frac{1}{c} = 1$$

(b) Using Lagrangian dual formulation, show that the weight vector can be represented as

$$w = \sum_{i=1}^{n} \alpha_i y_i x_i$$

where $\alpha_i \geq 0$ are the dual variables. What does this imply in terms of how to relate data to w?

Answer Given the alternate form of the optimization problem for SVM

$$min_{w,b}\frac{1}{2}w^T w$$

s.t.

$$1 - y^i(w^T x^i + b) \leq 0, \forall i$$

The lagragian seeks to find where the gradient of the objective is not equal, but proportional to the gradient of the constraint(s). That is, where the constraint is tangent with the objective.

$$\text{objective} = \alpha * \text{constraint}$$

Where α is the gradient proportionality constant, aka the lagrangian multiplier. From this step, we can convert the objective and the constraint to the lagrangian form of

$$\mathscr{L}(w, b, \alpha) = \frac{1}{2}w^T w + \sum_{i=1}^{m} \alpha_i(1 - y_i(w^T x_i + b))$$

We have now converted our constrained optimization problem into an unconstrained quadratic form, which is solved by taking the gradient of $\mathscr{L}$ and setting equal to 0

$$\nabla\mathscr{L} = \begin{bmatrix} \frac{\partial\mathscr{L}}{\partial w} \\ \frac{\partial\mathscr{L}}{\partial b} \\ \frac{\partial\mathscr{L}}{\partial \alpha} \end{bmatrix} = \mathbf{0}$$

where

$$\frac{\partial\mathscr{L}}{\partial w} = 0 = \frac{\partial}{\partial w}\frac{1}{2}w^T w + \frac{\partial}{\partial w}\sum_{i=1}^{m} \alpha(1 - y_i(w^T x_i + b)) = 0 = w - \sum_{i=1}^{m} \alpha_i y_i x_i = 0$$

$$\boxed{w = \sum_{i=1}^{m} \alpha_i y_i x_i}$$

From this we can see the vector **w** is a linear combination of the feature vectors and the lagrangian proportionality constant. This means that w is derived entirely from the feature vectors themselves

(c) Explain why only the data points on the "margin" will contribute to the sum above, i.e., playing a role in defining w.

Answer The KKT condition which specifies the criteria for the saddle point of $\mathscr{L}$ states that

$$\alpha_i g_i(w) = 0 \qquad (1)$$

where (1) is the constraint from above

$$\alpha_i(1 - y_i(w^T x_i + b)) = 0 \qquad (2)$$

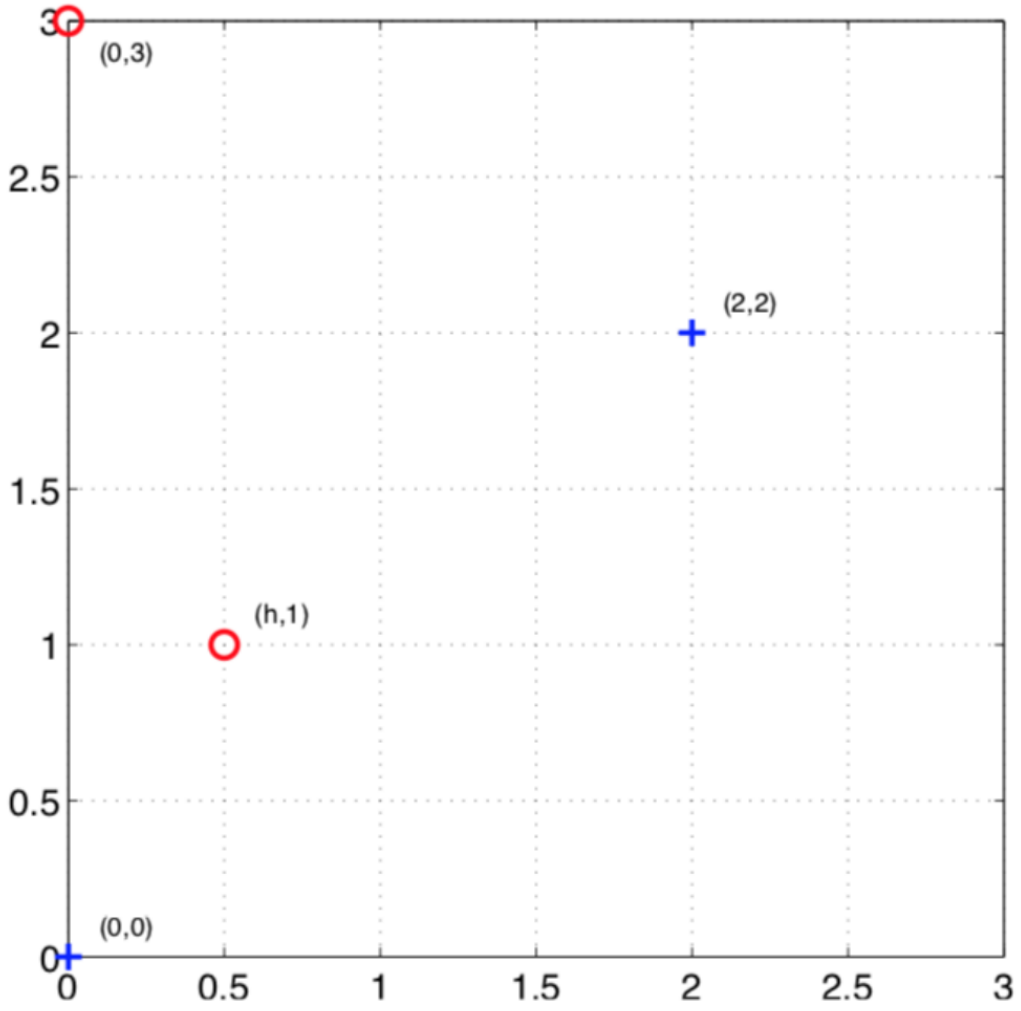
in order for (2) to be true, notice that for a given point, if 1 minus its projection onto **w** is non-zero, meaning the point is not on the margin,

$$1 - y_i(w^T x_i + b) < 0$$

then α must be equal to 0 in those instances. Conversely, if 1 minus the projection of a point onto **w** is zero,

$$1 - y_i(w^T x_i + b) = 0$$

then $\alpha_i > 0$. Therefore, only those points who have a non-zero alpha lay on the margin. When plugging all these values back into the summation above in the previous section, we can see those points which do not lay on the margin will have a multplicative factor of zero for α_i and those which do lay on the margin will have a non-zero multplicative factor. Thus, only those points that lay on the margin contribute to the summation above



(d) Suppose we only have four training examples in two dimensions as shown in Fig. The positive samples at x 1 = (0, 0), x 2 = (2, 2) and negative samples at x 3 = (h, 1) and x 4 = (0, 3).

(i.) For what range of parameter h > 0, the training points are still linearly separable?

Answer If we draw a straight line between the two blue cross points at (0,0) & (2,2), and solve for the slope (m) in $y = mx + b$

$$y = \frac{(2-0)}{(2-0)}x + 0$$

$$y = x \qquad (1)$$

we can see that the points are linearly separable up until the point in which y = x (1) to the left side of the line. Since the point **(h,1)** has a y of 1, it will be that the points remain separable for h < 1. Conversely, if we draw a line from the point in the upper left hand corner at **(0,3)** to point **(2,2)** and solve for the equation $y = mx + b$

$$y = \frac{(2-3)}{(2-0)}x + 3$$

$$y = \frac{-1}{2}x + 3 \qquad (2)$$

We plug in 1 for y into (2) and solve for x

$$1 = \frac{-1}{2}x + 3 - 2(1-3) = xx = 4$$

Hence, when h > 4, the graph is linearly seperable to the right side. Therefore, the points are still linearly seperable under the conditions of

$$\boxed{\begin{cases} 0 \leq h < 1 \\ h > 4 \end{cases}}$$

(ii.) Does the orientation of the maximum margin decision boundary chage as h changes, when the points are seperable?

Answer for the first interval $0 \leq h < 1$, we can see from equation (1) that the slope is positive 1. Therefore the margin runs in an upward trend from the bottom left to the upper right. For the second interval, we can see from equation (2) that the slope is negative $\frac{-1}{2}$, meaning the margin would run in a downward trend from the upper left corner to the lower right corner of the graph

Q2

Multi-class classification for MNIST data set, comparison

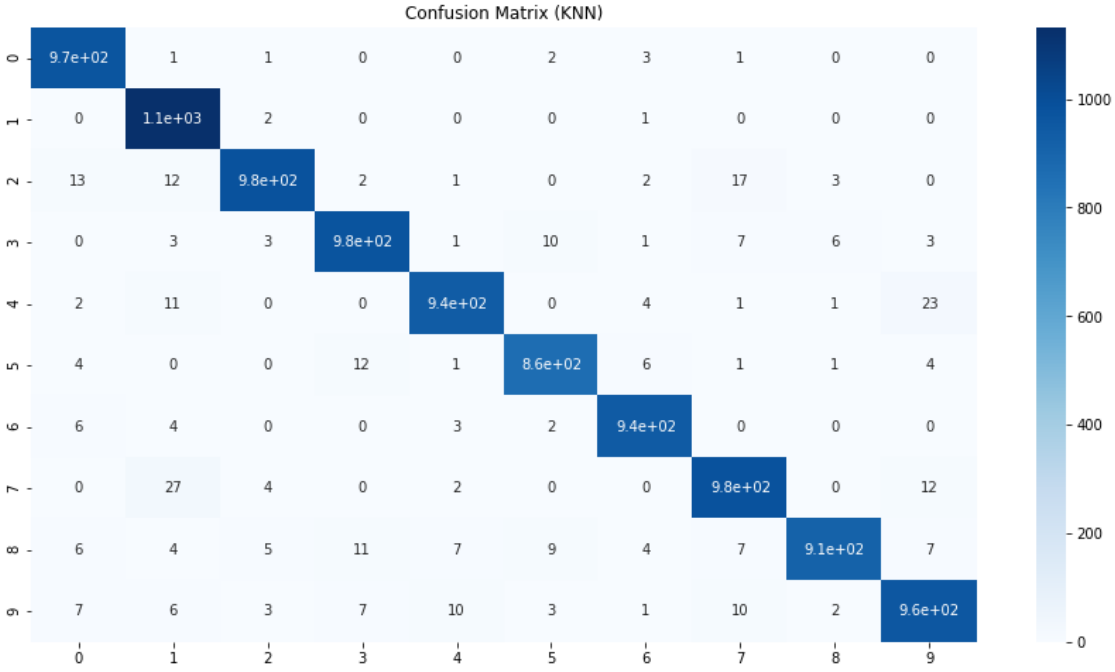
This question is to compare different classifiers and their performance for multi-class classifications on the complete MNIST dataset at http://yann.lecun.com/exdb/mnist/. You can find the data 1file mnist 10digits.mat in the homework folder. The MNIST database of handwritten digits has a training set of 60,000 examples, and a test set of 10,000 examples. We will compare KNN, logistic regression, SVM, kernel SVM, and neural networks. We suggest to use Scikit-learn, which is a commonly-used and powerful Python library with various machine learning tools. But you can also use other similar libraries in other programming languages of your choice to perform the tasks. Below are some tips. * We suggest you to "standardize" the features before training the classifiers, by dividing the values of the features by 255 (thus map the range of the features from [0, 255] to [0, 1]). * You may adjust the number of neighbors K used in KNN to have a reasonable result

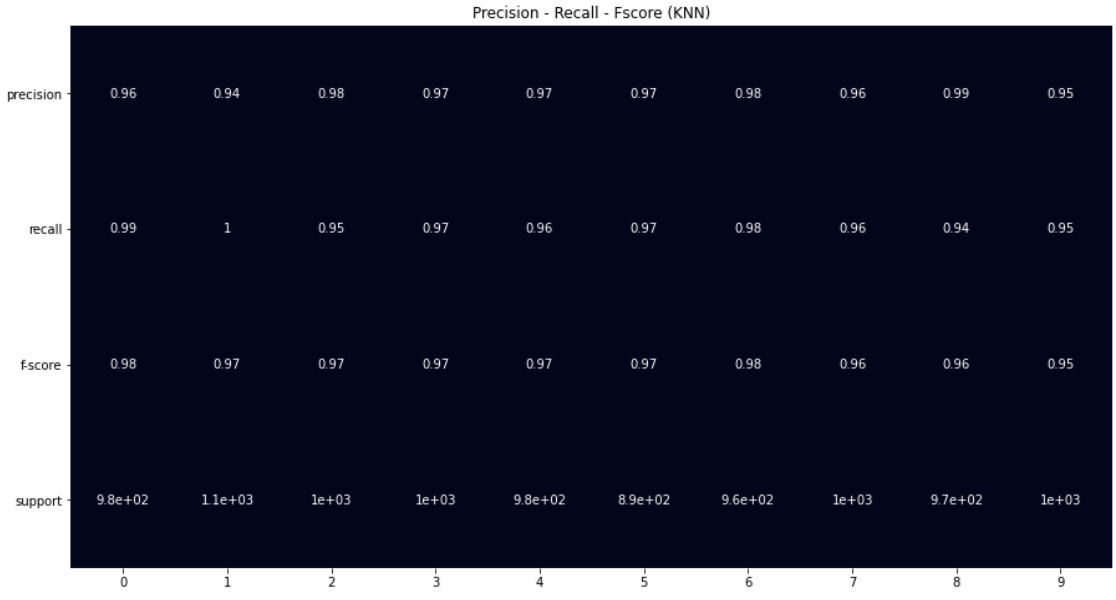
(you may use cross validation but it is not required; any reasonable tuning to get good result is acceptable). * You may use a neural networks function sklearn.neural network with hidden layer sizes = (20, 10). * For kernel SVM, you may use radial basis function kernel, and a heuristic called "median trick": choose the parameter of the kernel $K(x, x') = exp-||x - x'||^2/(2\sigma^2)$. Choose the bandwidth as $\sigma = \sqrt{M/2}$ where M = the median of $||x^i - x^j||^2, 1 \leq i, j \leq m', i \neq j$ for pairs of training samples. Here you can randomly choose m' = 1000 samples from training data to use for the "median trick" * For KNN and SVM, you can randomly downsample the training data to size m = 5000, to improve computation efficiency. Train the classifiers on training dataset and evaluate on the test dataset.

Report confusion matrix, precision, recall, and F-1 score for each of the classifiers. For precision, recall, and F-1 score of each classifier, we will need to report these for each of the digits. So you can create a table for this. For this question, each of the 5 classifier, KNN, logistic regression, SVM, kernel SVM, and neural networks

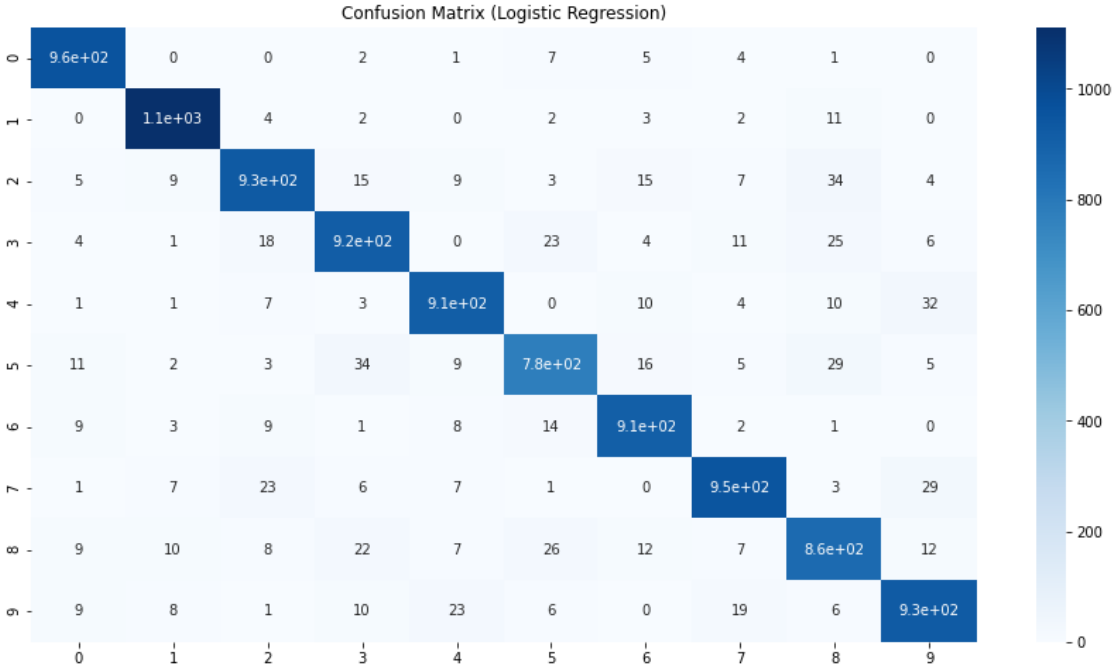
{-} Answer

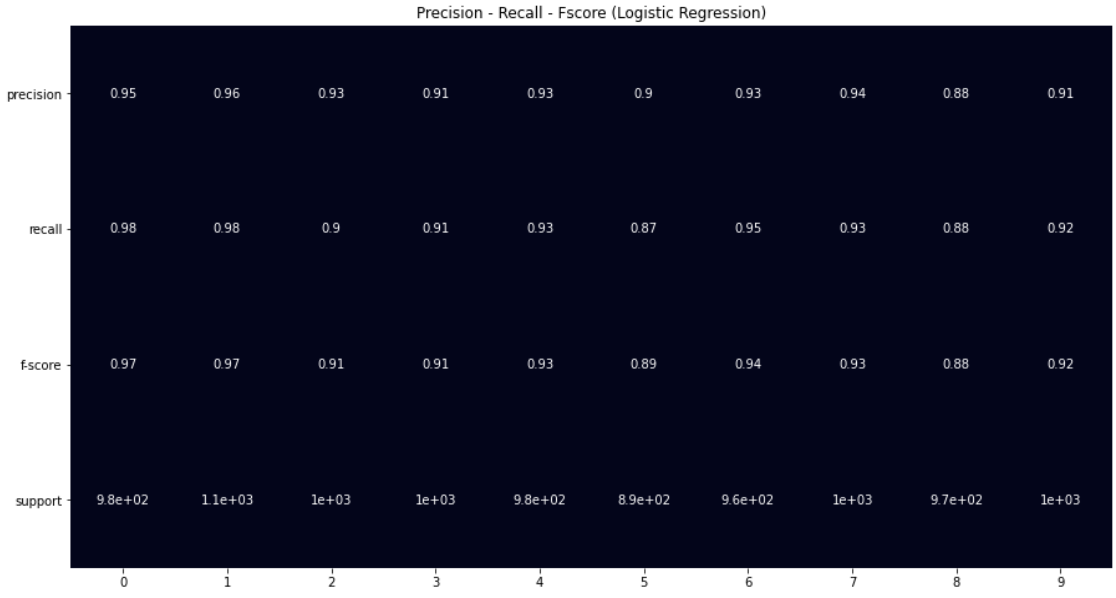
0.1 KNN


Confusion Matrix (KNN)
0 9.7e+02 1 1 0 0 2 3 1 0 0
1 0 1.1e+03 2 0 0 0 1 0 0 0
2 13 12 9.8e+02 2 1 0 2 17 3 0
3 0 3 3 9.8e+02 1 10 1 7 6 3
4 2 11 0 0 9.4e+02 0 4 1 1 23
5 4 0 0 12 1 8.6e+02 6 1 1 4
6 6 4 0 0 3 2 9.4e+02 0 0 0
7 0 27 4 0 2 0 0 9.8e+02 0 12
8 6 4 5 11 7 9 4 7 9.1e+02 7
9 7 6 3 7 10 3 1 10 2 9.6e+02
0 1 2 3 4 5 6 7 8 9
1000
800
600
400
200
0

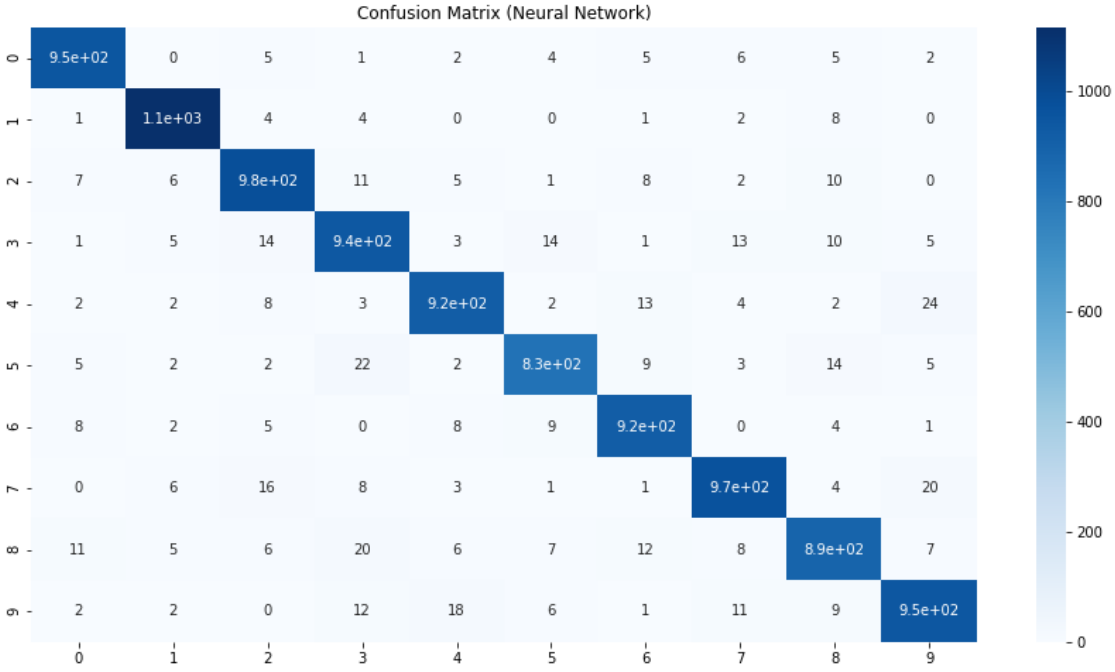

Precision - Recall - Fscore (KNN)
precision 0.96 0.94 0.98 0.97 0.97 0.97 0.98 0.96 0.99 0.95
recall 0.99 1 0.95 0.97 0.96 0.97 0.98 0.96 0.94 0.95
f-score 0.98 0.97 0.97 0.97 0.97 0.97 0.98 0.96 0.96 0.95
support 9.8e+02 1.1e+03 1e+03 1e+03 9.8e+02 8.9e+02 9.6e+02 1e+03 9.7e+02 1e+03
0 1 2 3 4 5 6 7 8 9

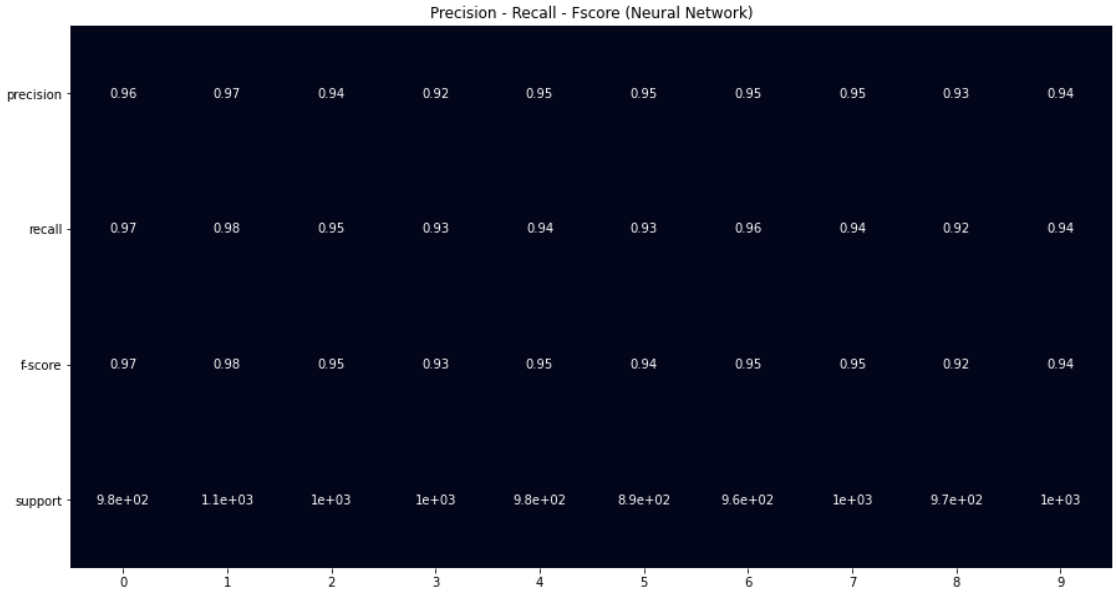
0.2 Logistic Regression



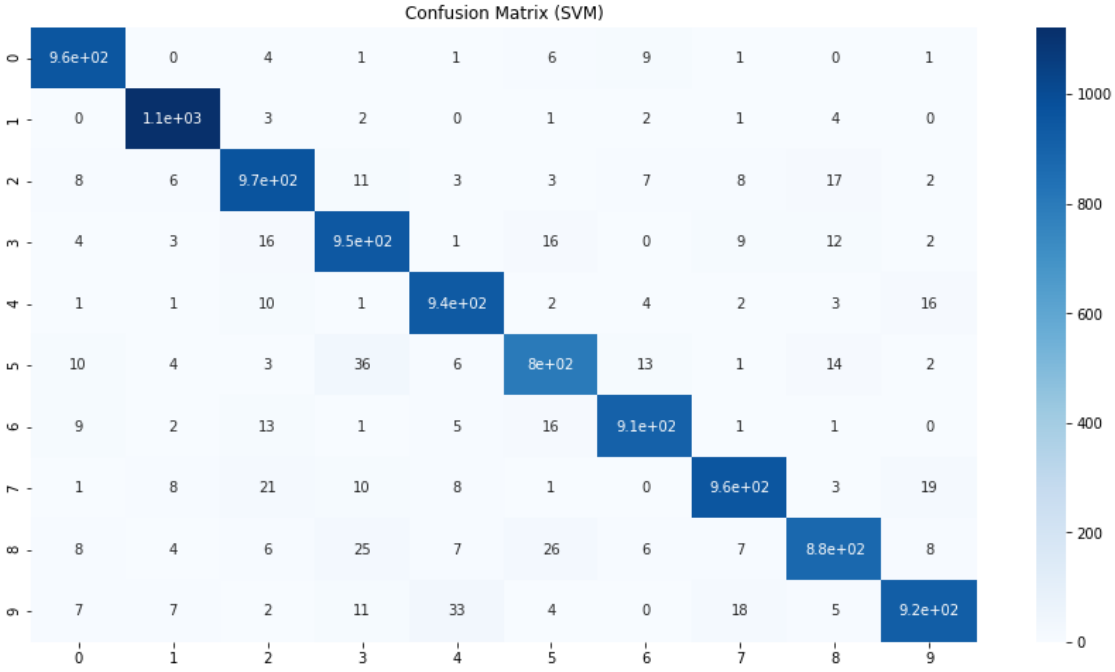


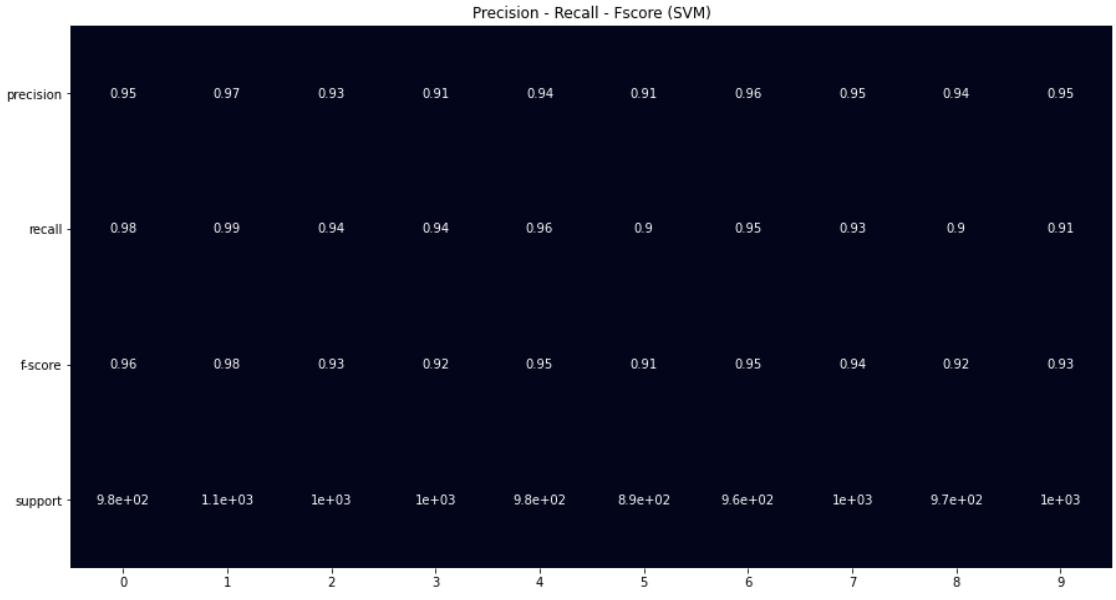
0.3 Neural Network





0.4 SVM





Comment on the performance of the classifier and give your explanation why some of them perform better than the others.

Answer All models perform exceptionally well given the high dimensionality of the input feature space. However, KNN and NN performs better than Logistic Regression and SVM. This is probably due in part that the latter models inherently are linear classifiers. Although the RBF kernel is being applied to SVM to conform to the nonlinearity of the classes, the data isn't very easily separated. KNN doesn't attempt to find a parametric function that maximizes the separation

between the classes, this is one reason why it performs better. It attempts to find the classes based on how similar they are. In this respect, similarity is deteremined from distance in euclidean space. Conversely, with respect to the NN, it has the capability to learn extremely complex non linear decision functions. Due to the high dimensionality and lack of clear separability between the classes, the NN has the ability to learn an efficient non linear function for classificaiton. Hence why it performs better than SVM and Logistic Regression.