CS57800: Project Report

Shraddha Parimita Sahoo & Vinoth Venkatesan
(sahoo0, venkat26)@purdue.edu

Contents

Introduction

The aim of the project is to study the performance of various classification algorithms, namely, Support Vector Machines (SVM), K-nearest neighbors (KNN), and neutral networks. We evaluate these algorithms on the task of identifying handwritten digits. The details of the data set and the pre-processing that was done is described next.

Data set Preprocessing

The MNIST data set, which is a subset of a bigger data set from National Institute of Standards and Technology (NIST), is a collection of handwritten digits, each between 0 to 9, and is commonly used as a benchmark data set for evaluating various image processing systems. The data set is preprocessed and has a training set of 60,000 examples, and a test set of 10,000 examples. The pre-processing details are described next.

The original NIST data set which was black and white (i.e. had only two levels) was first processed to fit into a 20×20 pixel image. Each image was then smoothed (anti-aliased) to obtain a grey-scale image which was then centered in a 28×28 pixel box. The centering was performed by computing the center of mass of the pixels, and translating the image so as to position that point at the center of the 28×28 field. The data set was downloaded using the *scikit-learn* python library (`fetch_mldata` method in package `sklearn.datasets`).

Next, we describe the classification algorithms that were used in this project.

Classification Algorithms

In this section, we describe three methods, namely, SVM, KNN and Neural network, for the task of recognizing handwritten digits. The classification task is to learn a function that maps images of handwritten digits $\mathcal{X}$ to digits $\mathcal{Y} = \{0, 1, \ldots 9\}$, from a data set of labeled examples (images of handwritten digits). In our case, $\mathcal{X} = [0, 255]^{784}$ corresponds to the set of flattened 28×28 grey-scale images. Let d (784 in this case) denote the dimension of the feature space and k (10 in this case) denote the number of classes. We have a data set $S = \{(\mathbf{x}_i, y_i)\}_{i=1}^{n}$ of n labeled images.

Support Vector Machines

Support Vector Machines (SVM) is one of the most widely used classification algorithm for both binary and multi-task problems. SVM is a maximum margin classifier. To solve the multi-class problem of classifying handwritten digits, we use the one-vs-rest classifier approach where each class gets a classifier bringing the total number of classifiers to 10 in our case. We train each binary classifier using by solving the following optimization problem:

$$\hat{\boldsymbol{\alpha}} = \min_{\boldsymbol{\alpha} \in \mathbb{R}^d} \quad \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{n} \alpha_i \alpha_j y_i y_j K(\mathbf{x}_i, \mathbf{x}_j)$$

$$\text{subject to} \quad 0 \leq \alpha_i \leq C, i = 1, \ldots, n, \; \sum_{i=1}^{n} \alpha_i y_i = 0,$$

where α_i and α_j are non-negative Lagrange multipliers used to enforce the classification constraints, y_i andy_j are predicted labels for data points $\mathbf{x}_i$ and $\mathbf{x}_j$ and K is the kernel function.

The prediction y for a test data point $\mathbf{x} \in \mathcal{X}$, for each binary classifier, is then obtained as follows:

$$y = \text{sign}\left(\sum_{i=1}^{n} \hat{\alpha}_i y_i K(\mathbf{x}, \mathbf{x}_i)\right)$$

Finally, the class which received the most votes is selected as the predicted label. We have evaluated the performance of the **linear kernel** in our experiments. We used `scikit-learn` python package for an implementation of the above.

K-Nearest Neighbors

K-nearest neighbors (KNN) is a popular non-parameteric classification algorithm which predicts the label of data point by computing the majority label over K nearest neighbors to the data point. In the training phase, a data structure (KD Tree) is constructed over all training data points for efficiently performing nearest neighbor queries during prediction. During prediction, the constructed tree is queried to find the K nearest neighbors, in Euclidean distance, and then the majority label is predicted as the class label of the test data point. We use uniform weighting for computing the votes.

Neural networks

Neural Network / Multi-Layer Perceptron (MLP) is a supervised learning algorithm that uses multiple layers of neurons (perceptrons) to learn a non-linear function $f(\cdot) : \mathbb{R}^d \rightarrow \mathbb{R}^k$. It differs from logistic regression by having multiple non-linear layers between the input layer and the output layer. In our case, the input layer has a dimension of 784(d) and the output layer has a dimension of 10(k - classes). Given training data, we use backpropagation to learn the weights and biases of the perceptrons in the neural network by optimizing a cost function.

When a new test point comes in, we predict the class it belongs based on the output array (k-dimensional) from the output layer by looking at the class which has the highest vote. Various parameters like the learning rate, the activation function for the perceptrons and regularization affect the performance of the neural networks and we use a 10-fold cross-validation approach to tune these parameters.

Experimental Set-up

In this section we describe our experimental setup in detail. We used 10-fold cross validation (CV) with stratified sampling to pick the optimal hyper-parameters for each algorithm, namely, the regularization parameter for SVM, the value of K for KNN, the best activation function, gradient descent learning rate and the regularization parameter for Neural networks. We then plot the mean cross-validation accuracy for each hyper-parameter value used for different algorithms.

After picking the best hyper-parameters for each algorithm from the above step, we ran experiments to find the effect of number of training data samples on accuracy. For different training data set sizes we computed the mean accuracy across 10 randomly sampled (stratified) data sets while keeping test samples fixed at 10,000. We then plot the test set accuracy as the number of samples is varied.

In another set of experiments, we explored the effect of using PCA (Principal Component Analysis) features on the accuracy. We ran experiments to pick the top-K PCA features which explained 90% of the variance in the data. We plot the variance explained versus the number of PCA features. Apart from these, we also performed hypothesis testing and studied a few other parameters, all of which have been explained in the following sections.

We used the implementation provided by the `scikit-learn` python library for the classification algorithms.

Stratified sampling

We used stratified sampling to ensure that the relative proportion of various classes in the full data set is maintained in each of the sampled data set. In stratified sampling the whole dataset is first partitioned into 10 groups (one group for each class 0-9). Then data points are sampled from each group and combined to form samples.

The experimental set-up for 10 fold cross validation, effect of varying number of training data on accuracy and PCA feature generations are explained next.

Implementation details

Receiver Operating Characteristic (ROC) curve

The ROC curve is an useful visual metric to study the classification potential of an algorithm. It is obtained by varying the threshold that is used to classify the data points in the test data set. After running the classification algorithms that we used on the test data, we obtained predicted labels and scores from the corresponding `decision_function()` for the algorithm from the `scikit-learn` python library.

input : *true_labels*, *predicted_scores*, *pos_label*
output: *fpr*, *tpr*

```
1  true_labels ← (true_labels == pos_label)
2  // Sort scores and corresponding truth values
3  desc_score_indices ← Sort(predicted_scores)
4  predicted_scores ← predicted_scores [predicted_scores ]
5  true_labels ← true_labels [predicted_scores ] // Find distinct scores
6  distinct ← DistinctValues(predicted_scores)
7  // Accumulate the true positives with decreasing threshold
8  tps ← CumulativeSum(true_labels)
9  fps ← 1 + distinct- tps
10 // Normalizing the scores to get TPR and FPR
11 fpr ← fps/ fps [-1]
12 tpr ← tps/ tps [-1]
```

Algorithm 1: ROC curve parameters (TPR/FPR)

This information is used to plot the ROC curve. The implementation for plotting the curve which relates the True Positive Rate (TPR) and the False Positive Rate (FPR) is shown in Algorithm 1.

The algorithm shown here returns the FPR and TPR values taking the score and the true labels as the arguments. It should be noted that this implementation is for a binary classification problem (two-labels). Since our problem is a multi-class classification problem, we extended this implementation by performing a "macro-averaging" approach which interpolates all the FPR and TPR scores of all the classes. In this regard, the `roc_curve_params()` method takes in a *pos_label* argument which is the current label (digits 0-9) that is under consideration.

An example of an ROC curve that is generated based on the above implementation is shown in Fig.

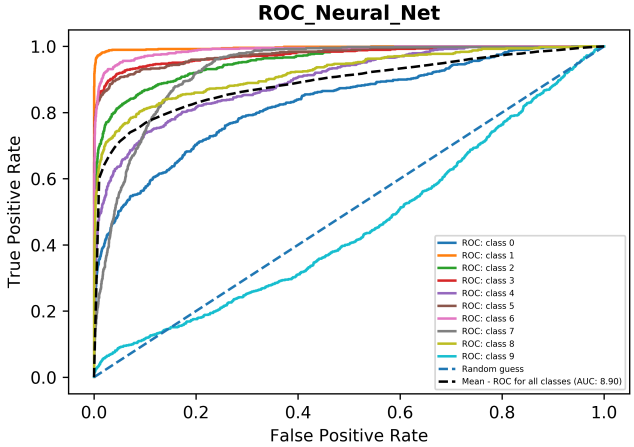


Figure 1: ROC curve example

1. The ROC curve of each of the 10 classes is highlighted along with the *mean_curve* and the Area Under Curve (AUC) metric for the mean curve which is a measure of the accuracy of the classification algorithm. A perfect classification algorithm would have an AUC score of 1. The mean curve is found using the macro-averaging approach shown in the following algorithm.

```
def plot_ROC_curve(y_true, y_score):
    mean_tpr = 0.0
    mean_fpr = np.linspace(0,1,100)
    for i in xrange(10):
        fpr[i], tpr[i] = roc_curve_params(y_true, y_score, pos_label = i)
        mean_tpr += interp(mean_fpr, fpr[i], tpr[i])
        mean_tpr[0] = 0

    mean_tpr /= 10
    mean_tpr[-1] = 1
```

Confusion matrices

Confusion matrices are another way to study the performance of a classification algorithm. While the configuration of a confusion matrix is straightforward for a binary classification, for multi-class classification it can be represented as a matrix of values of size $(n \times n)$, where n is the number of classes.

An example of a confusion matrix for the MNIST dataset is shown in Fig. 2. The confusion matrix is being plotted as a colormap using the in-built capabilities of `matplotlib` Python package and each row represents the proportion of classification of each class being classified as different classes by a classifier. For a perfect classifier, the confusion matrix would be an identity matrix.

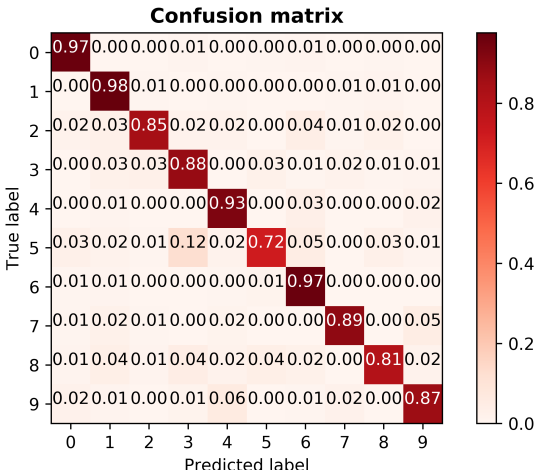


Figure 2: Confusion matrix example

Accuracy curve

As part of the experiments, we perform a 10-fold cross validation for parameter tuning, for hypothesis testing, etc. and to study the performance of algorithms in these tests, we study the accuracy score in the test fold. Since this is a multi-class classification, the accuracy of the model is represented as:

$$Accuracy = \frac{True\_Positives}{Total\_number\_of\_samples}$$

Once we have this information, the plot showing the variation of accuracy with varying parameters were plotted as error bars (containing both mean and variance information) and these have been included in the results section.

Hypothesis testing

Once we had all these performance metrics, we compared the performance of algorithms using hypothesis testing. The data from the 10-fold CV steps were used for comparing the algorithms. Given a pair of means (μ_1, μ_2) and variances (σ_1, σ_2), we determine:

$$x = \frac{(\hat{\mu_1} - \hat{\mu_2})\sqrt{n}}{\sqrt{\hat{\sigma_1}^2 + \hat{\sigma_2}^2}}$$

$$\nu = \left\lceil \frac{\hat{\sigma_1}^2 + \hat{\sigma_2}^{2^2}(n-1)}{\hat{\sigma_1}^4 + \hat{\sigma_2}^4} \right\rceil$$

From these values, we reject the null hypothesis $\mu_1 = \mu_2$, in favor of $\mu_1 > \mu_2$, if $x > x_{1-\alpha,\nu}$. We fixed the confidence level $\alpha = 0.95$. In our case, we performed hypothesis testing for comparing the following pairs of algorithms:

- SVM vs. Neural Net (using PCA data)

- SVM vs. KNN (using PCA data)
- Neural Net vs. KNN (using PCA data)
- SVM vs. Neural Net (using actual data)
- SVM vs. KNN (using actual data)
- Neural Net vs. KNN (using actual data)
- SVM (with PCA) vs. SVM (without PCA)
- Neural Net (with PCA) vs. Neural Net (without PCA)
- KNN (with PCA) vs. KNN (without PCA)

All of these hypothesis tests were done with the results from the 10-fold cross validation (which use the tuned parameters that were found earlier using the parameter tuning experiments explained earlier). So, in a sense each algorithm is performing at it's best for the MNIST dataset. Of course, considering more parameters and tuning within a finer range of these parameters might result in better performance of the respective algorithms.

Principal Component Analysis (PCA)

Principal Component Analysis (PCA) is a common dimensionality reduction procedure that is used in statistics / machine learning for high dimensional data. After finding the principal components, we can select a few components in the order of maximum variance along these components and project the training data into this subspace.

When a test data point comes in, we project it to the previously mentioned subspace and use the dimensionally reduced form of the data for prediction. We used this procedure for a few experiments as explained in the upcoming sections, albeit with a few considerations to make sure that we don't use any information from the test data:

- **10-fold cross validation.** Whenever we perform cross-validation and we use PCA for dimensionality reduction, we make sure that we perform PCA to only the `9-folds` of data that is being considered as the `training set`. So the PCA matrix and mean values that are obtained for these data are then used to: **(i) center** the test data (the remaining 1-fold) and **(ii) project** that centered data to the subspace.
- **Centering the data.** The centering of the data is done only for the training set and the same mean is used for the test set to ensure that the data is being processed with respect to the same origin in the subspace

Other Miscellaneous Metrics

Apart from these parameters, we also report Precision and Recall scores for the final test runs (10-fold CV using the tuned parameters). While precision and recall are not as efficient as accuracy in gauging the performance of an algorithm, they have been included to provide a sense of completion and in understanding the performance from different perspectives.

Since we use a multi-class dataset, the precision and recall scores have been calculated using a "*weighted-averaging*" technique. Here, the metrics are calculated for each label and a weighted average of them is taken based on the support (the number of samples for each label in the dataset).

Experiments/Results

10-Fold Cross Validation For Parameter Tuning

For 10-fold cross validation we used stratified sampling using all 70,000 MNIST data. We first divided the whole data into 10 groups (data belonging to each digit from 0-9). Then for each fold in 10-fold CV, we picked one fold from each group and combined them to form our test data set (approximately 7,000 data points) and rest of the data i.e. the other 9-folds were our training data set (approximately 63,000 data points).

Support Vector Machines

The results for SVM is shown in Fig. 3. The figure shows the trend for "*Accuracy*" as we vary the regularization parameter (C) for SVM. The plot shows both the mean and the variance of the accuracy values (from the cross-validation for each value of 'C').

Observations. The C parameter is a measure of how much we want to avoid mis-classifying each training example. Large values of C will result in the hyperplane having a smaller margin and in cases where the data is separable, this would usually results in a better performing SVM with lesser misclassifications. Conversely, a a small value of C will cause the optimizer to look for a larger-margin separating hyperplane, even if that hyperplane misclassifies more points.

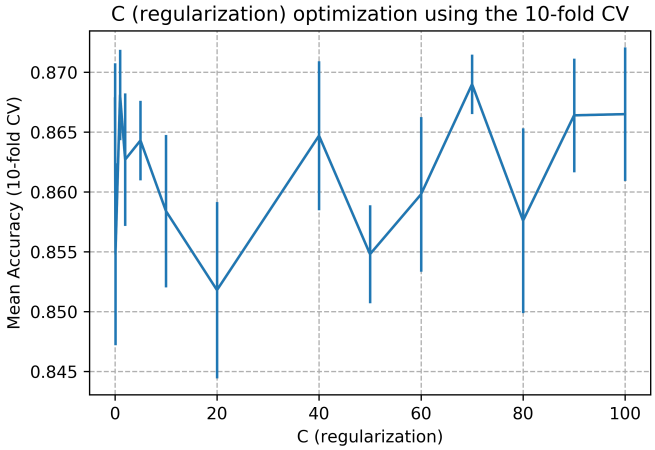


Figure 3: 10-fold CV (Parameter tuning) for SVM

An interesting thing that we observed while tuning the parameter 'C' for our dataset is the *absence* of a clear trend while changing the parameter value. As can be seen from the plot, we had a higher density of values for 'C' at the lower range in order to study the effect of the parameter better. However, the trend of the variance and mean in accuracy suggests that the data (MNIST) is ***not linearly separable*** causing the hyperplane to misclassify points even at high values of 'C' and hence the effect of C is very minimal in this dataset.

After performing these experiments, the hyper-parameter C for SVM was selected to be **70**. This was the highest value in the plot and hence was chosen even though the trend does not suggest an obvious answer. Apart from the accuracy curve, the performance was also studied using ROC curves while changing the

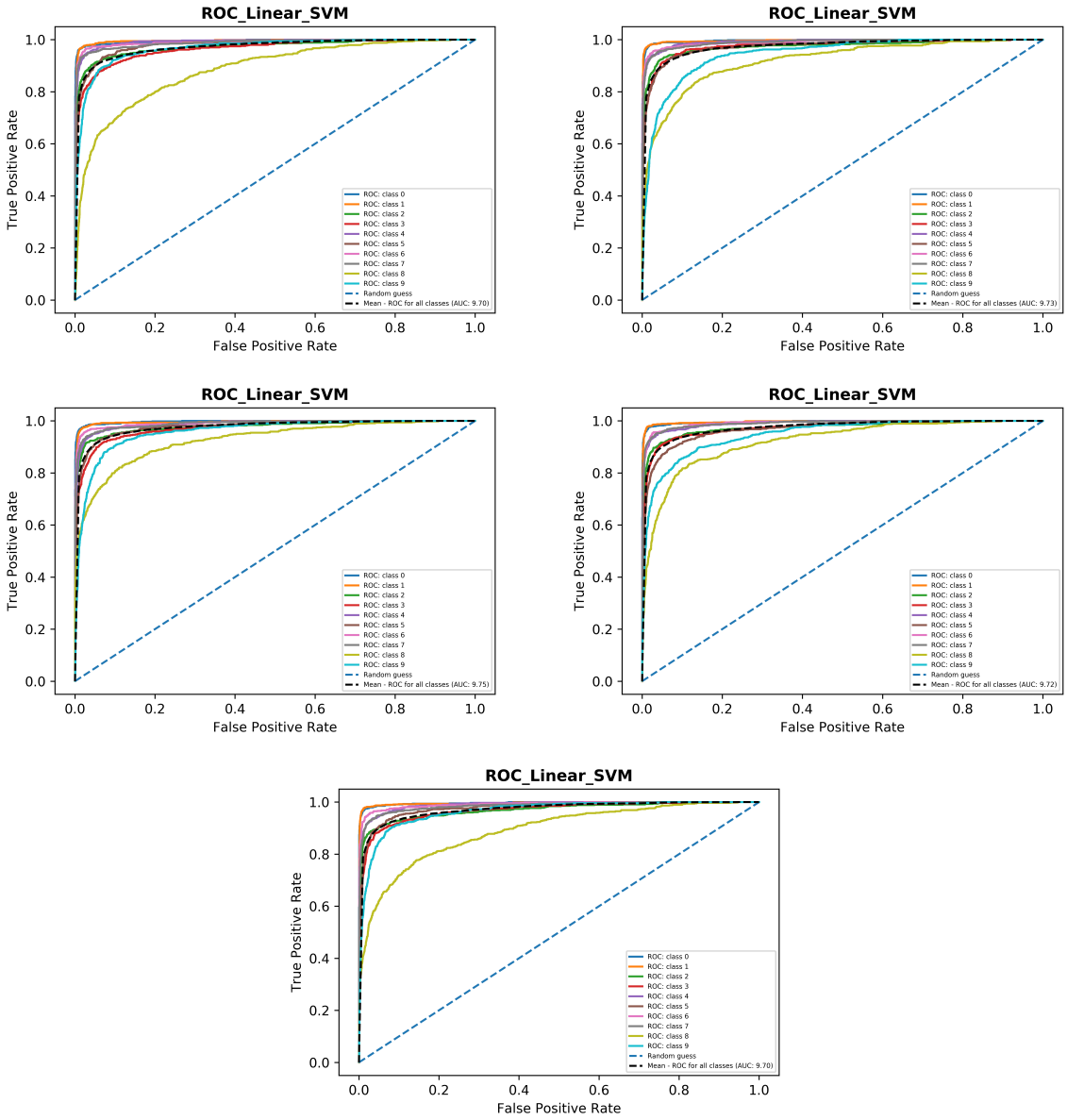


Figure 4: ROC curves with AUC for the mean curve for $C = 0.01, 2, 40, 70, 100$ (Going from Left to Right and Top to Bottom)

values of 'C'. The results are shown in Fig. 4 for a few values of C. Along with the ROC curves, the Area Under Curve (AUC) metric is also included for the mean curve (which was found using the "macro-averaging" approach). The plots shown are all for the 1^{st} fold of the cross validation experiment. The plots for all the folds and for the entire range of the parameter is included in the submission folder.

Neural Networks

The results for Neural Networks is shown in Fig. 5. Unlike SVM, where we just looked at one parameter to optimize, we optimized the neural net that we chose using three different parameters namely: the `activation function`, the `learning rate` for the backpropagation algorithm, and the regularization parameter α.

Observations.

- **Activation function.** We looked at three options for the activation function for the neurons in the NN: The logistic sigmoid function (**logistic**), the hyperbolic tan function (**tanh**) and the rectified linear unit function (**relu**). While the **relu** function just a max classifier ($f(x) = max(0, x)$), the other two are more involved and thus we expect better accuracy, which is the case as can be seen from the Figure. After running the CV, we chose the **logistic** activation function based on the results.

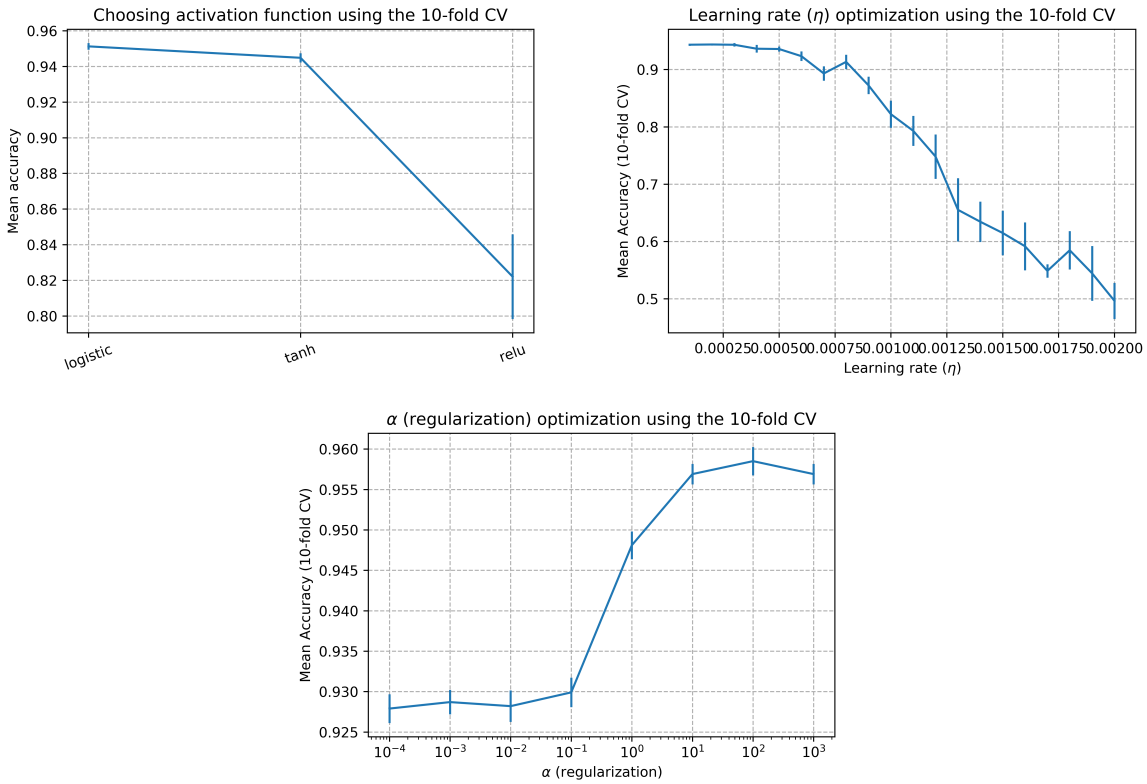


Figure 5: CV for Neural networks.

- **Learning rate.** Learning rate is NN is used to control the convergence to an optimal solution of the cost function. A larger learning rate might converge faster but might overshoot the optimal value, while a lower learning rate might take a long time to converge. Hence, we experimented

with a range of values for the tuning, expecting a better accuracy for lower values. And the results met our expectations, giving us the best accuracy for a value of (η): 0.0003. While this comes at a price of higher training times, the compromise was not significant and in order to demonstrate this, a plot of the training times for the dataset for different values of the learning rate is shown in Fig. 6.

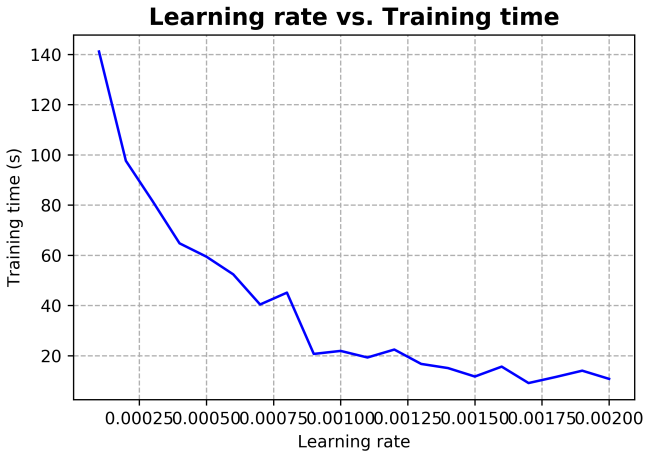


Figure 6: Training time vs. Learn rate

- **Regularization parameter:** Like any other learning algorithm, the regularization parameter here helps in reducing an **overfit** of the training data and having the trained algorithm to have a robust classifying behavior when it sees the test data. We used a range of values, and in order to have a bigger picture of the effect of the parameter, a **logscale** was used as shown in Fig. 5. From these observations, a value of (α): 100 is picked as the best tuned parameter.

- **Miscellaneous notes.** Before optimizing these parameters, the size of the neural networks was also decided upon. This was done based on a combination of a review of existing literature for dealing with dataset (and similar datasets) and a set of experiments (refer to the code). After exhaustive experimentation, the size of the neural net was chosen to be $(784, 100, 10)$, a model which has a single hidden layer with 100 neural units.

K-Nearest Neighbors

The results are shown in Fig. 10.

Observations. The K parameter in KNN is basically the number of neighbor data points to look at before deciding which class the test point belong to.

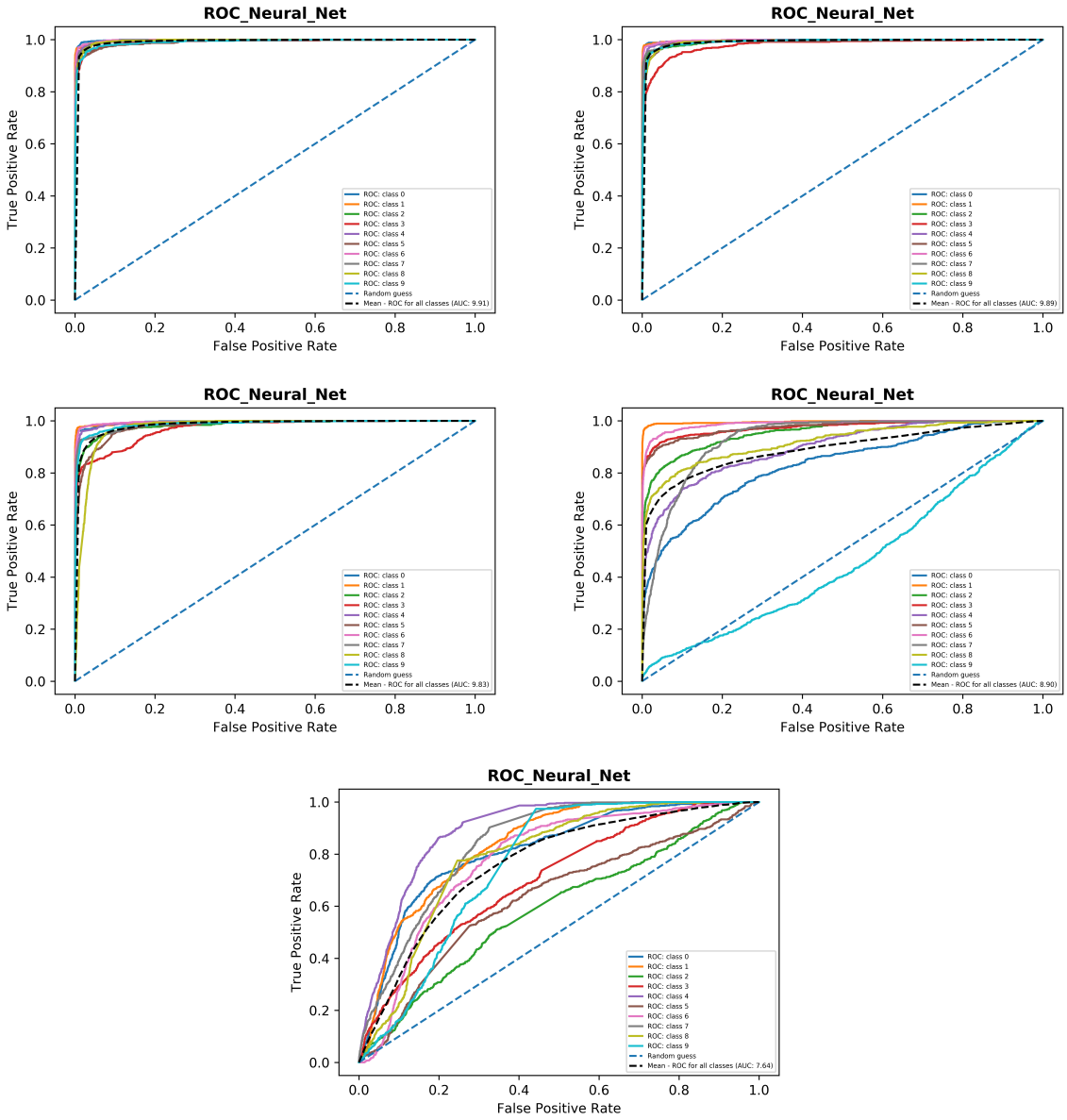


Figure 7: ROC curves with AUC for the mean curve for $\alpha = 10^{-5}, 10^{-4}, 10^{-1}, 10^{2}, 10^{3}$ (Going from Left to Right and Top to Bottom)

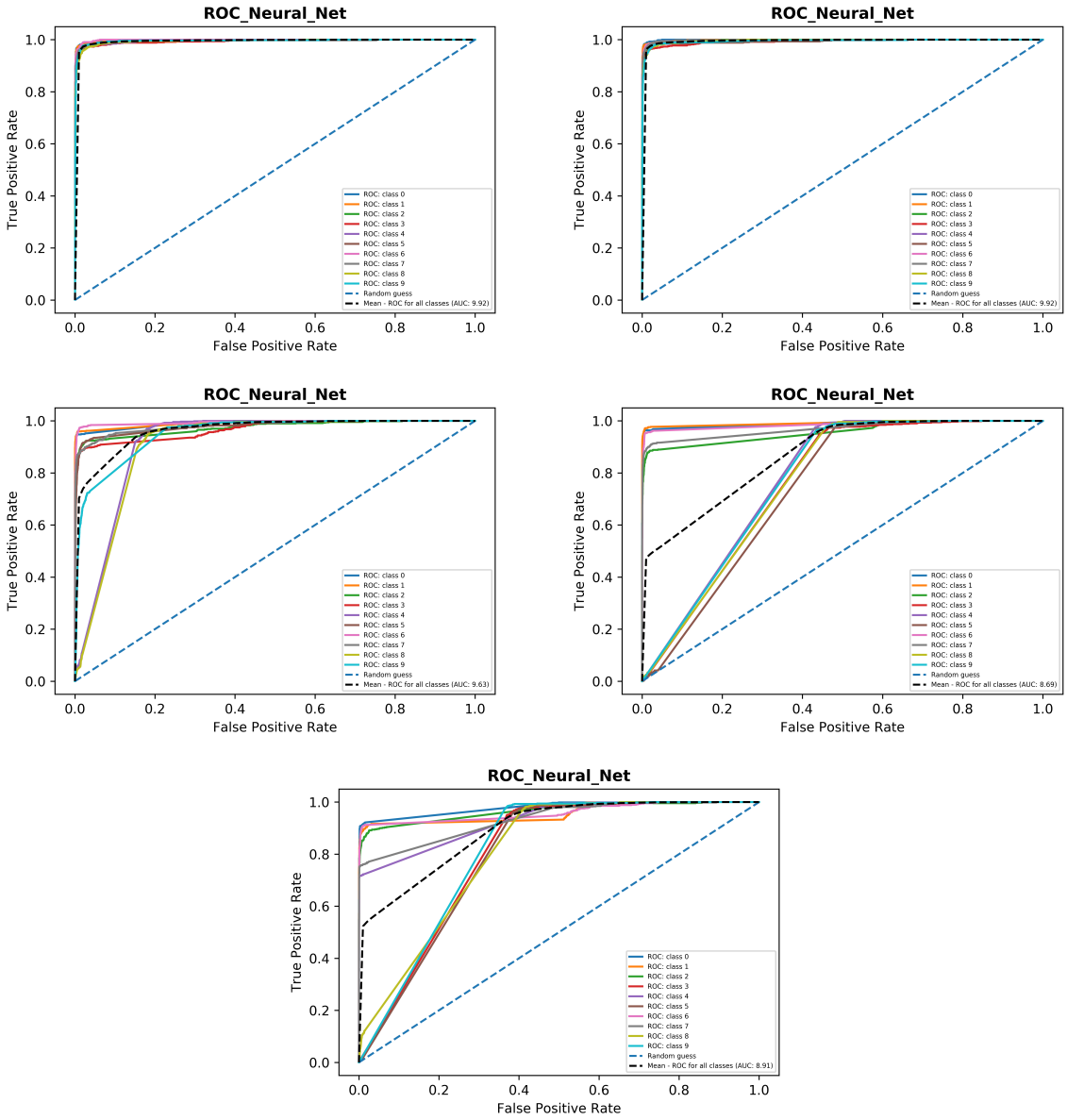


Figure 8: ROC curves with AUC for the mean curve for $Learn\_rate = 10^{-5}, 5^{-5}, 11^{-4}, 15^{-4}, 19^{-4}$ (Going from Left to Right and Top to Bottom)

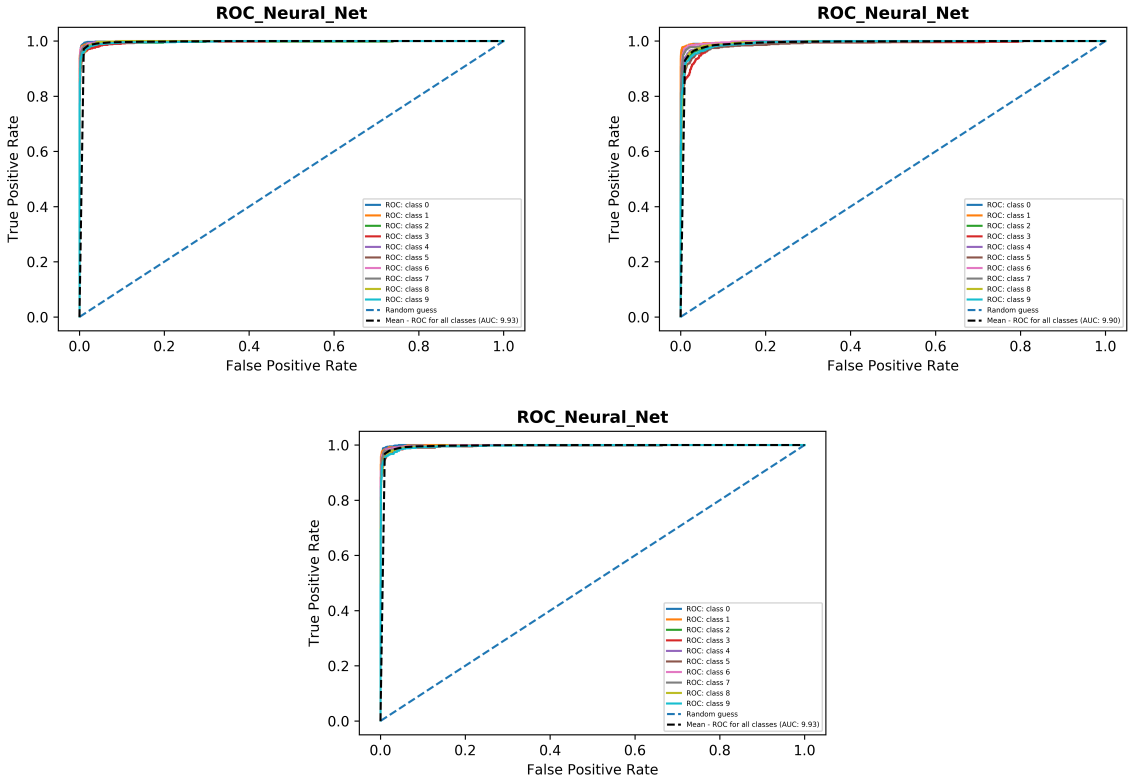


Figure 9: ROC curves with AUC for the mean curve for Activation_funciton = `logistic`,`relu` and `tanh` (Going from Left to Right and Top to Bottom)

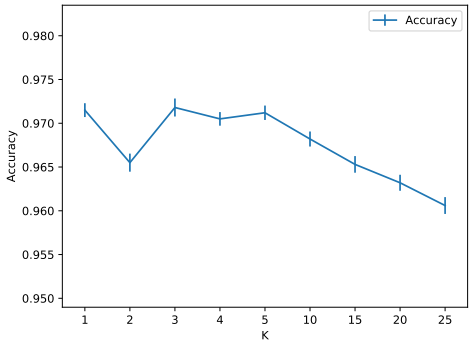


Figure 10: 10-fold CV for KNN.

From the figure we observe that the mean accuracy is high at both K=1 and K=3. But since the mean accuracy was little higher at K=3, we chose the value of K to be 3 for KNN. We also observe that as we increase the value of K from 3 to to higher number, the accuracy of the test data monotonically decreases. So effect of K is evident from the plot for the MNIST dataset.

Apart from the accuracy curve, the performance was also studied using ROC curves while changing the values of 'K'. The results are shown in Fig. 11 for a few values of K.

Effect of number of samples on accuracy

For this experiment, we progressively varied the fraction of samples used for training from 1% to 100% out of the entire 60,000 samples using randomized stratified sampling, taking the fraction of data from each class (digits 0-9) and combining them to form the training data and computed the accuracy on the test data (fixed size 10,000) for each case. Here we fixed the hyper-parameters for respective algorithm found using CV. The results are shown in Figure 12.

TODO: Maybe include a paragraph about the implementation / pseudocode for randomized stratified sampling

From the plots we see that, approximately 20% (12,000) of the training data points (60,000) is required for high accuracy. After 20% data points, the accuracy almost remains in the same range even if we increase the number of training data points.

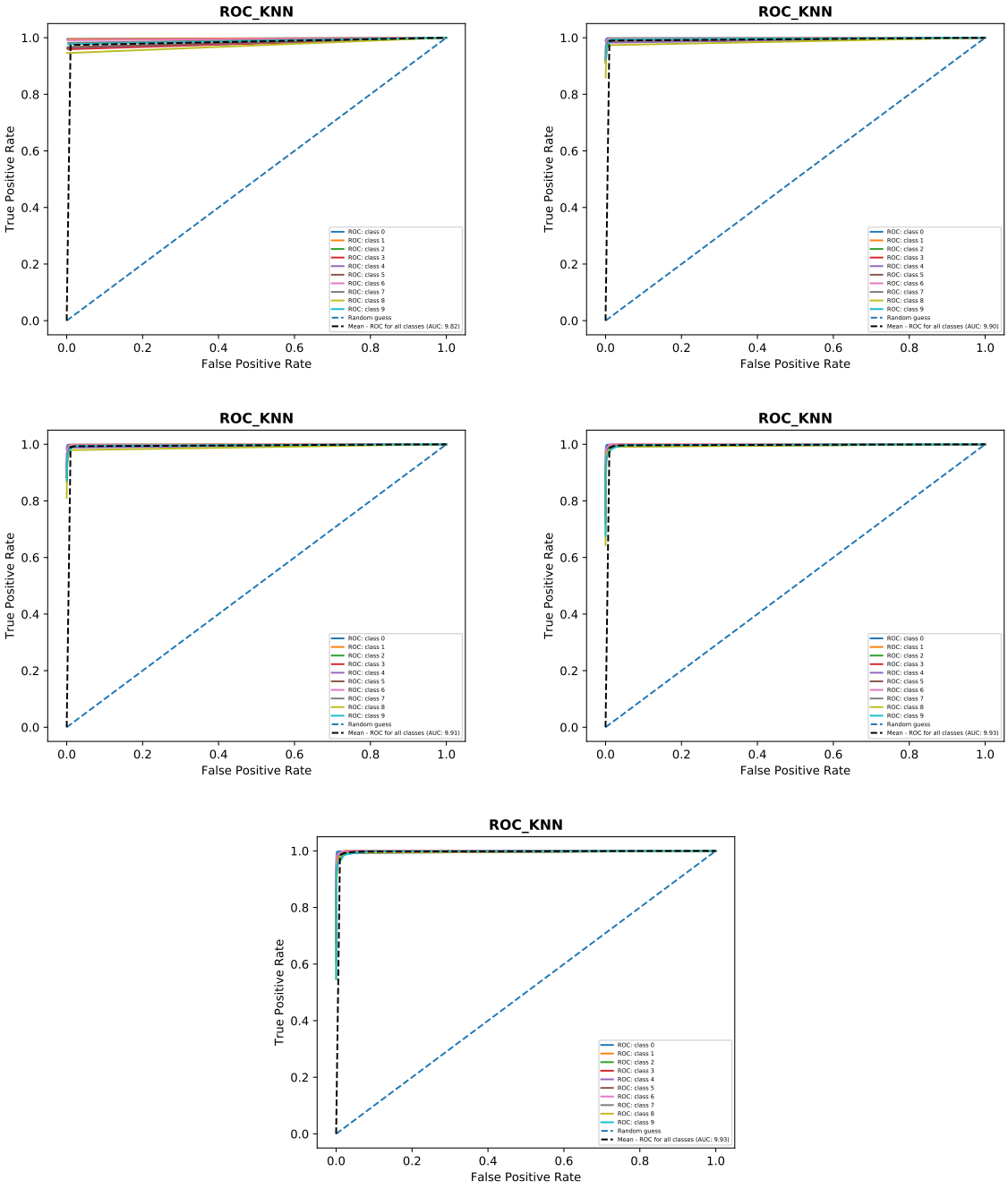
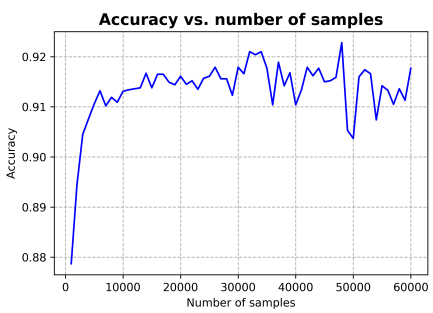
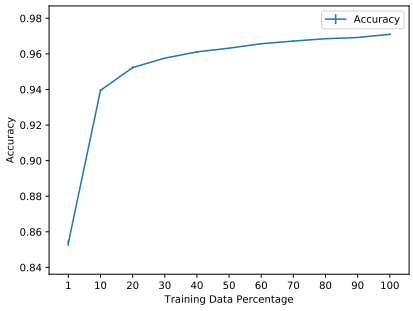


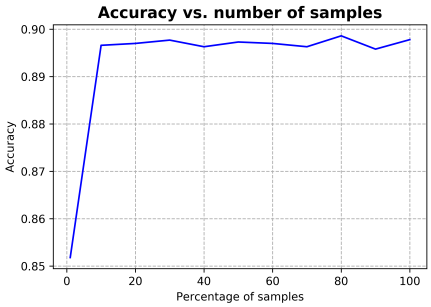
Figure 11: ROC curves with AUC for the mean curve for $K = 1, 3, 5, 15, 25$ (Going from Left to Right and Top to Bottom)



(a) SVM



(b) KNN



(c) Neural Network

Figure 12: Training set size vs. accuracy

PCA features

For this experiment, we computed the top-K principal component features such that the top-K principal components explained 90% of the variance in the data. The results is shown in Figure 13.

From the plot we can see that the number PCA features are 87 out of 784 features which explains 90% variance in the data.

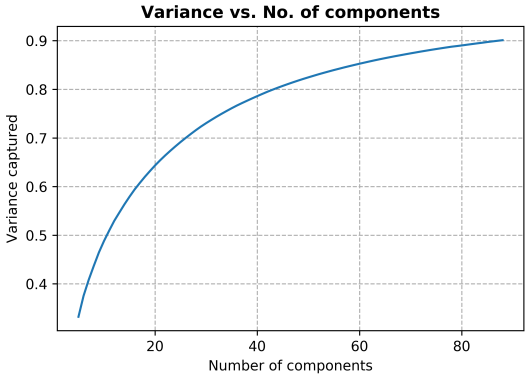


Figure 13: PCA features.

Hypothesis testing

The following sections contain the results of the hypothesis testing under various conditions and for different combinations. The results shown are basically the rejection or the acceptance of the null hypothesis (accepting $\mu_1 = \mu_2$) when comparing between different algorithms under different settings.

The parameters that have been used for each algorithm are the values that were found during the parameter tuning stage of the project. So, for all of these experiments, the parameter setting remain the same and the difference is mostly in the pre-processing of the data (performing PCA or not performing PCA). The aim of this was to study was two-fold:

- To understand the effect of performing PCA on the data and to see if the results are consistent with using the actual data. This should help us in reducing the training time overall since the PCA data would have low dimensionality.
- To compare the algorithms that were used to classify the dataset.

Comparison using PCA data

Here, all the results shown are experiments that were run using PCA data. The way in which the PCA data is being generated was described in a previous section and this method ensures that we don't use any information from the test data for training.

Experiment	**Result**	**Inference**
Neural Net vs. SVM	$x > x_\alpha$	Neural Net ($\mu_{NN} > \mu_{SVM}$) Reject Null Hypothesis
KNN vs. SVM	$x > x_\alpha$	KNN ($\mu_{KNN} > \mu_{SVM}$) Reject Null Hypothesis
KNN vs. Neural Net	$x > x_\alpha$	KNN ($\mu_{KNN} > \mu_{NN}$) Reject Null Hypothesis

Table 1: Hypothesis testing results (with PCA)

Observations.

- **Support Vector Machine.** The results of the hypothesis testing test run for SVM is shown in Figs. 15 and 14. The first figure shows the accuracy variation over the 10-folds of CV experiment. The second figure shows the confusion matrix and ROC curve for the run for the 1^{st} fold.

  These can be thought of to represent the best run of the algorithm since we use the optimized parameters. To get better perspectives of the experiment, the precision and recall curves have also been included in Fig. 16.

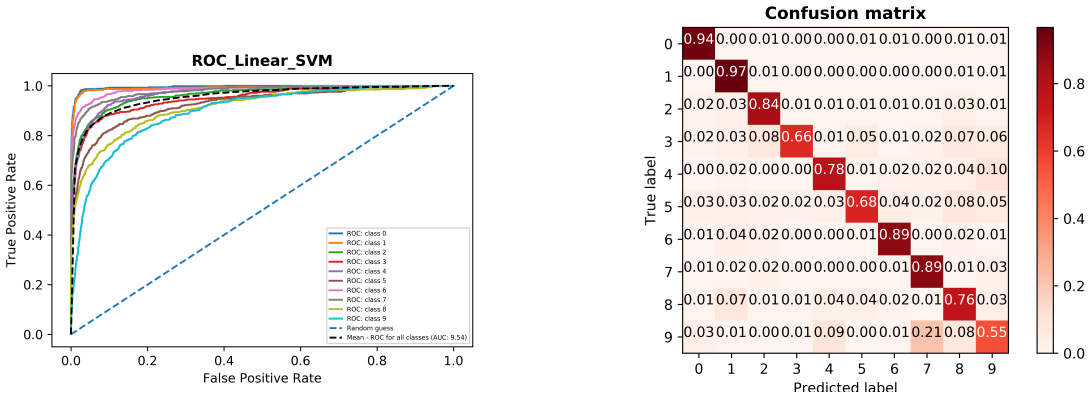


Figure 14: (Left) ROC curve. (Right) Confusion matrix [Hypothesis test SVM with PCA]

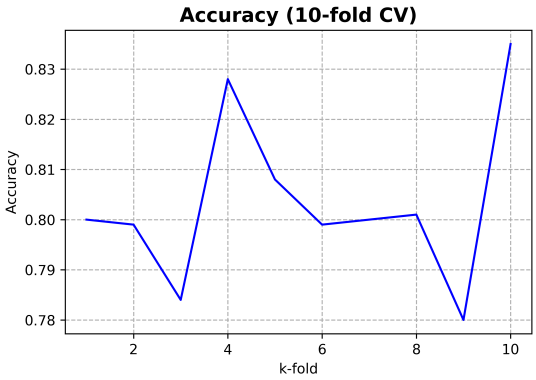


Figure 15: Accurcay (SVM) over the 10-fold CV (Hypothesis testing - with PCA).

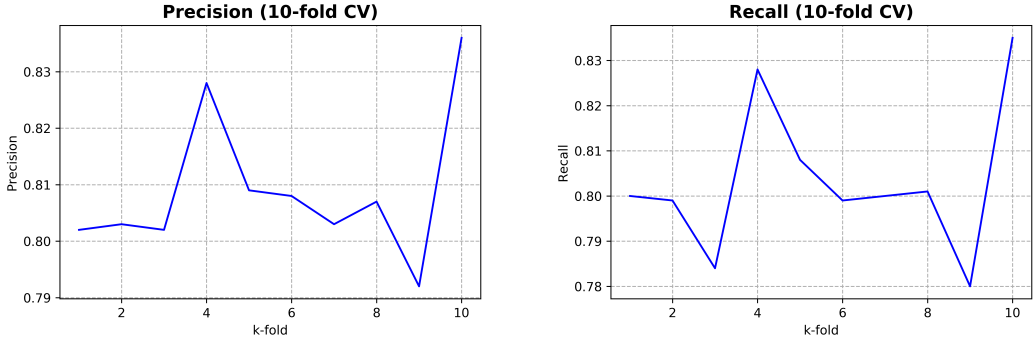


Figure 16: (Left) Precision curve. (Right) Recall curve. [Hypothesis test SVM with PCA]

- **Neural Network.** The results for the Hypothesis run for Neural Network is shown in Figs 18, 19 and 17. The results can be compared to the SVM runs and it is obvious that the Neural Net performs better than the SVM from a look at the plots for both. This is reflected in the hypothesis result table 1.

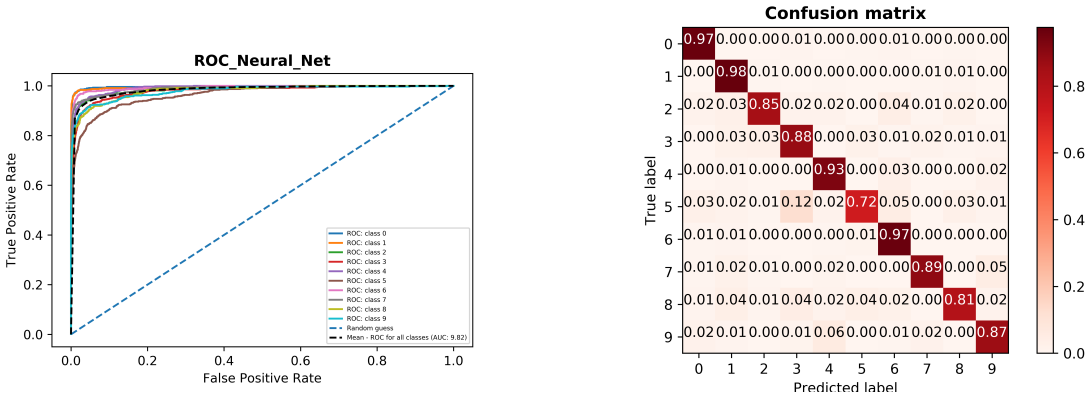


Figure 17: (Left) ROC curve. (Right) Confusion matrix [Hypothesis test NN with PCA]

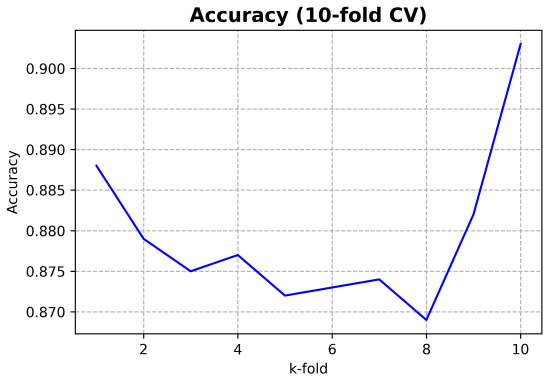


Figure 18: Accurcay (NN) over the 10-fold CV (Hypothesis testing - with PCA).

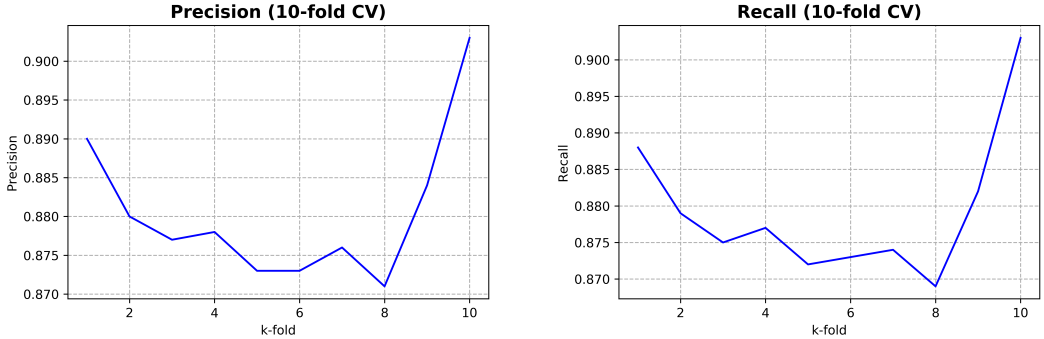


Figure 19: (Left) Precision curve. (Right) Recall curve. [Hypothesis test NN with PCA]

- **K-Nearest Neighbors.** The results for the Hypothesis run for KNN is shown in Figs . The results can be compared to the SVM and NN runs and it is obvious that the KNN performs better than both the SVM and NN from a look at the plots for both. This is reflected in the hypothesis result table 1.

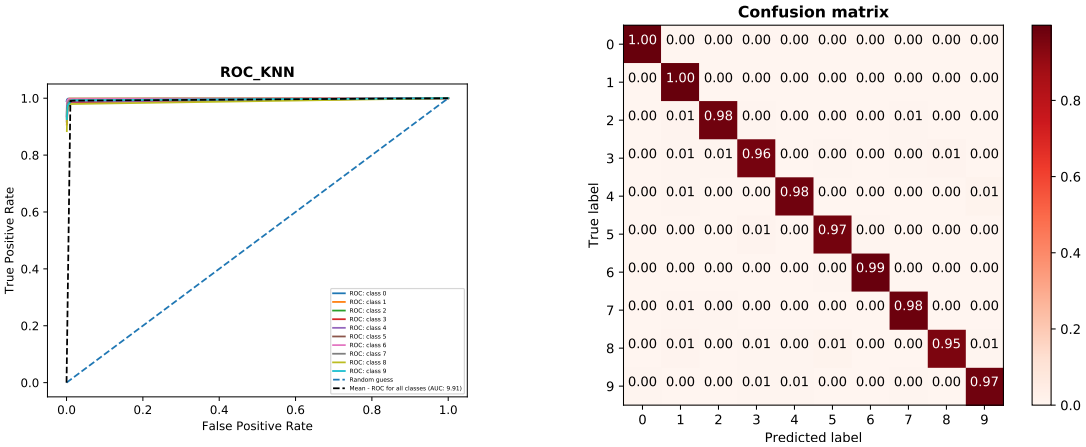


Figure 20: (Left) ROC curve. (Right) Confusion matrix [Hypothesis test KNN with PCA]

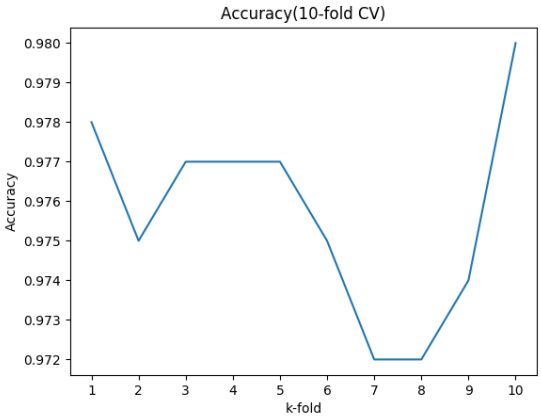


Figure 21: Accurcay (KNN) over the 10-fold CV (Hypothesis testing - with PCA).

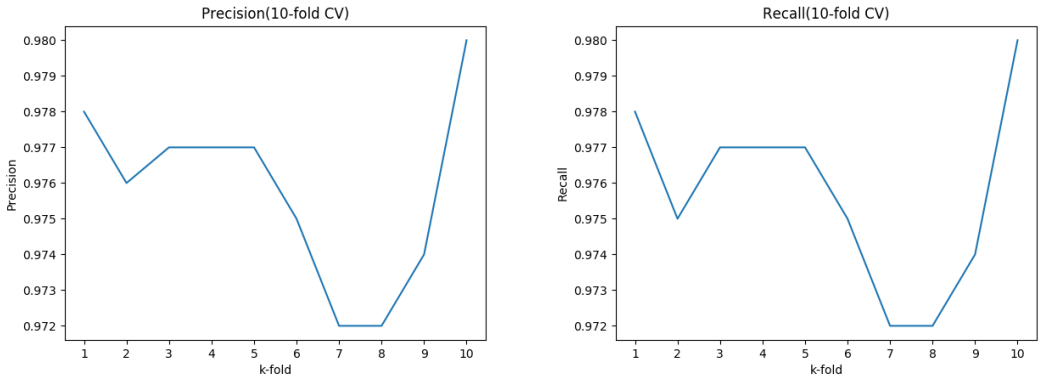


Figure 22: (Left) Precision curve. (Right) Recall curve. [Hypothesis test KNN with PCA]

Comparison using actual data

The same set of experiments mentioned above were performed using the actual data without any pre-processing and are shown in the following sections for all the three algorithms. These data were used to perform Hypothesis testing for all combinations of algorithms and the results can be seen in the table below:

Experiment	Result	Inference
Neural Net vs. SVM	$x < x_\alpha$	Equal ($\mu_{NN} = \mu_{SVM}$) Accept Null Hypothesis
KNN vs. SVM	$x > x_\alpha$	KNN ($\mu_{KNN} > \mu_{SVM}$) Reject Null Hypothesis
KNN vs. Neural Net	$x > x_\alpha$	KNN ($\mu_{KNN} > \mu_{NN}$) Reject Null Hypothesis

Table 2: Hypothesis testing results (without PCA)

Observations.

- **Support Vector Machine.** The results for the Hypothesis run for SVM with the actual data is shown in Fig 23.

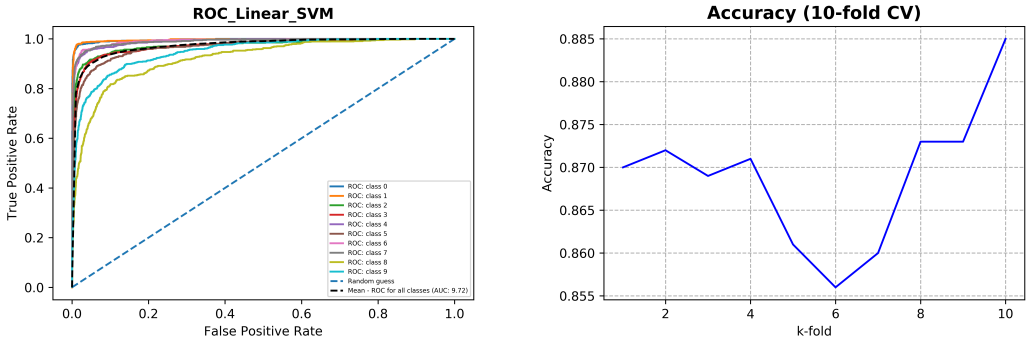


Figure 23: (Left) ROC curve. (Right) Accuracy curve. [Hypothesis test SVM without PCA]

- **Neural Network.** The results for the Hypothesis run for NN with the actual data is shown in Fig 24.

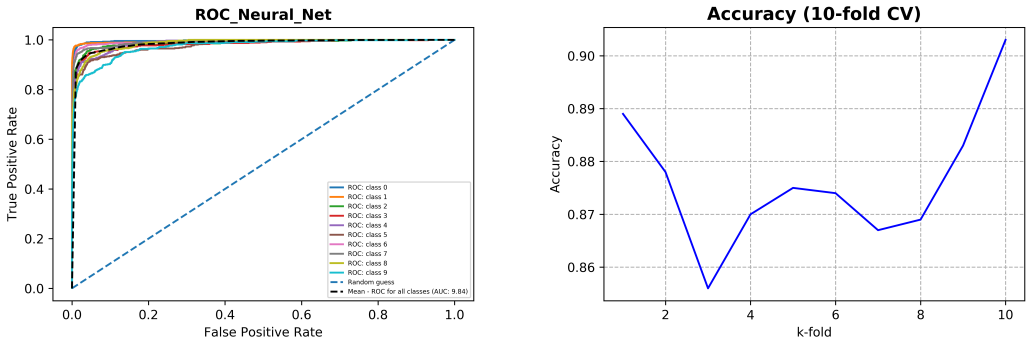


Figure 24: (Left) ROC curve. (Right) Accuracy curve. [Hypothesis test NN without PCA]

- **K-Nearest Neighbor.** The results for the Hypothesis run for KNN with the actual data is shown in Fig 25.

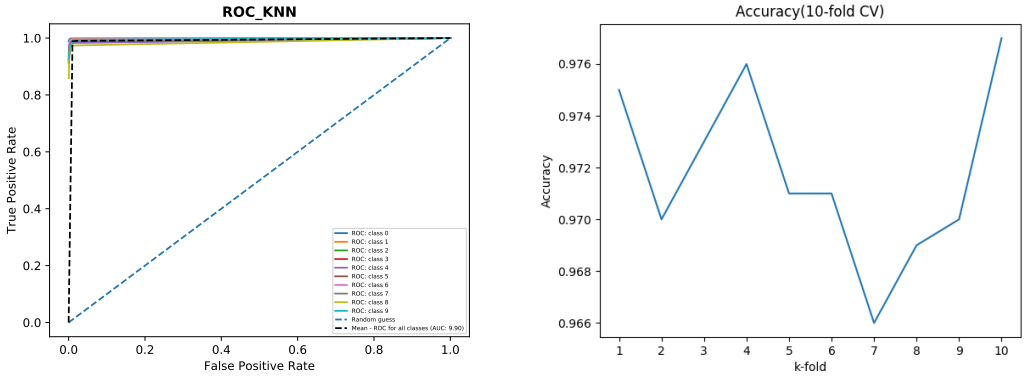


Figure 25: (Left) ROC curve. (Right) Accuracy curve. [Hypothesis test KNN without PCA]

Within algorithm comparison

Apart from the above comparisons, we performed hypothesis tests to compare the performance of these algorithms using PCA data vs. using the actual data. The results from these experiments are shown below.

Experiment	Result	Inference
SVM (actual) vs. SVM (with PCA)	$x > x_\alpha$	SVM (actual data) ($\mu_{SVM\_actual} > \mu_{SVM\_pca}$) Reject Null Hypothesis
Neural Net (actual) vs. Neural Net (with PCA)	$x < x_\alpha$	Equal ($\mu_{NN\_pca} = \mu_{NN\_actual}$) Accept Null Hypothesis
KNN (actual) vs. KNN (with PCA)	$x < x_\alpha$	Equal ($\mu_{KNN\_pca} = \mu_{KNN\_actual}$) Accept Null Hypothesis

Table 3: Hypothesis testing with PCA vs. without PCA

The results above show that while SVM performs better using the actual data, using Neural Network and KNN with PCA data produces similar performance as using the actual data. This provides a lot of flexibility in selecting the algorithm because using PCA data reduces the training time a lot.