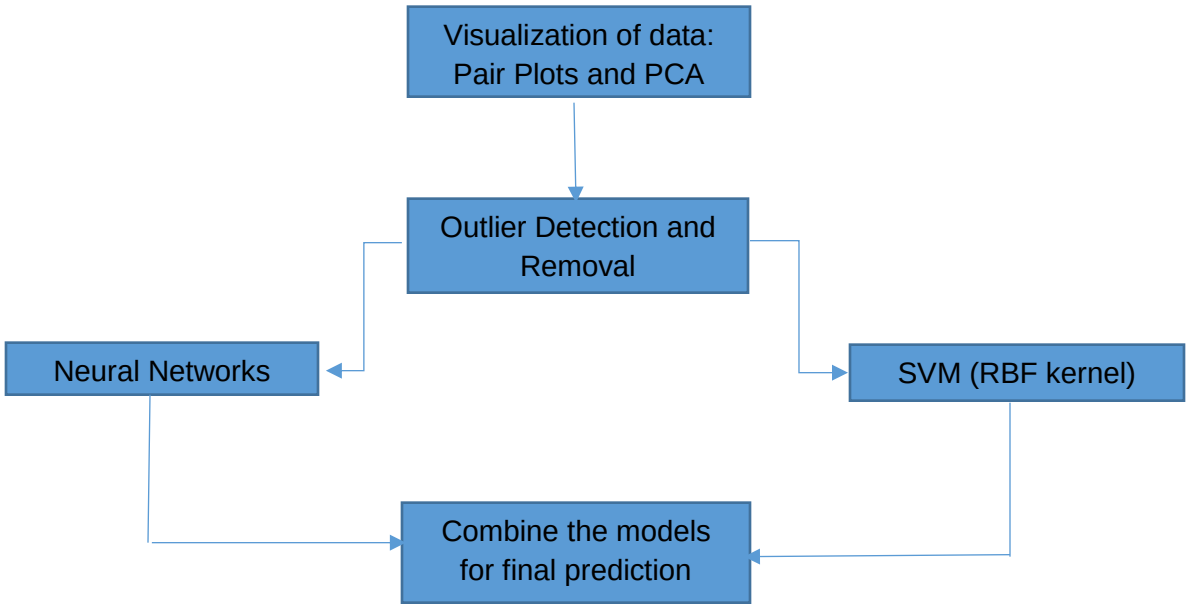
CS 273A Project Report: Rainfall Prediction

Anurag Mishra, 82482058, anuragm[at]uci.edu
Sudeep Meduri, 51884271, smeduri[at]uci.edu
KAGGLE submission name: anuragmds

Introduction:

The project that we have chosen is the in-class Kaggle competition wherein we predict whether there is rainfall at a certain location based on infrared satellite information. We have explored different techniques to predict the output. We have used neural networks and Support Vector Machine models, and then later used an ensemble learner to further optimize the performance so that the model doesn't underfit or overfit.

Flowchart showing the steps:



Data Visualization:

The input data consists of 13 different features. Since it is difficult to visualize the entire input data set, we have used a pair plot to visualize the correlation between the different input features. The pair plots can be seen in Fig.1 below.

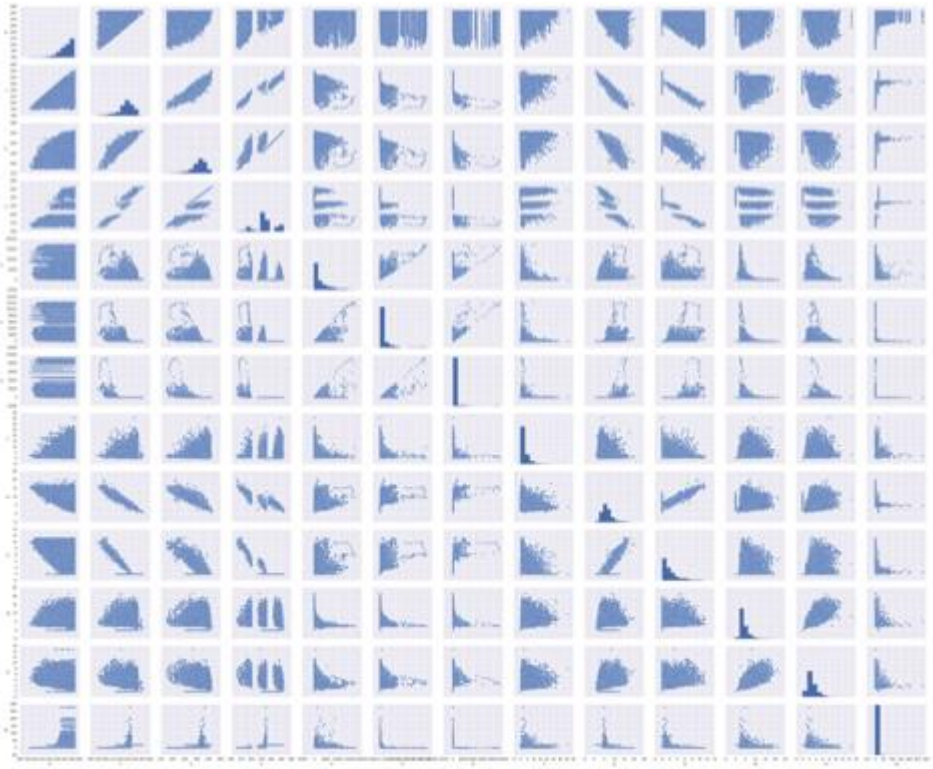


Fig.1: Pair plot of the input data set

Then we have done Principal Component Analysis(PCA) on the input data. PCA is often used as dimensionality reduction technique, and explains the variance-covariance structure of a set of variables through linear combinations. It helps to condense the information contained in a large number of input features, with minimum loss of information. We used PCA to reduce the dimensionality of the input data set to 3 principal dimensions, and plotted the input and output data set in the transformed 3-dimensional feature space (Fig.2).

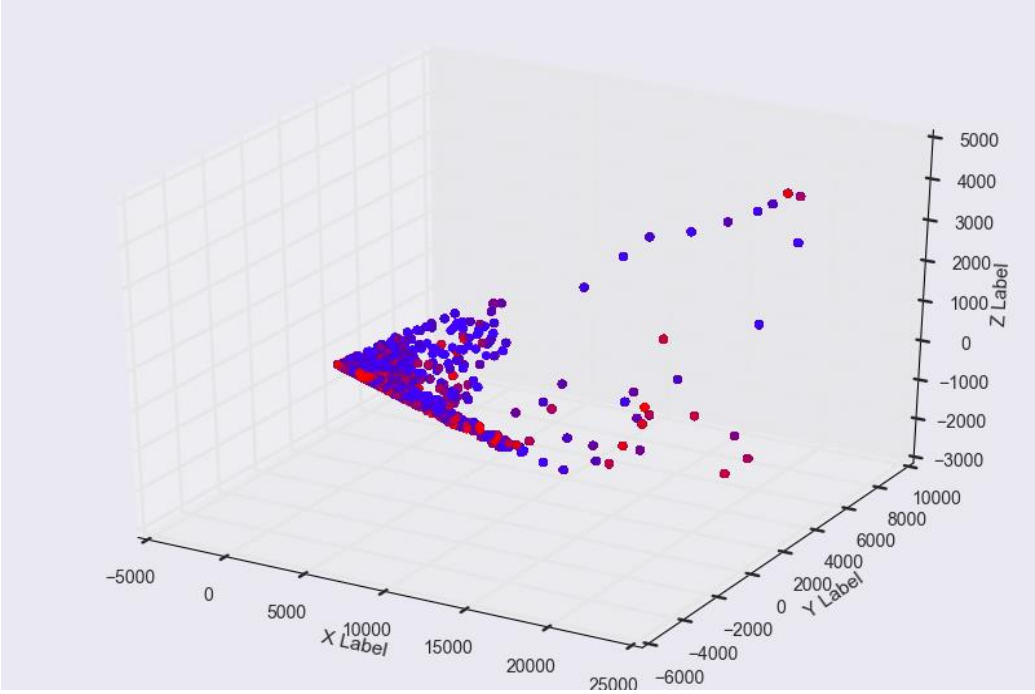


Fig.2: Visualizing the data in 3-dimensions using Principal Component Analysis

Outlier Detection and Removal:

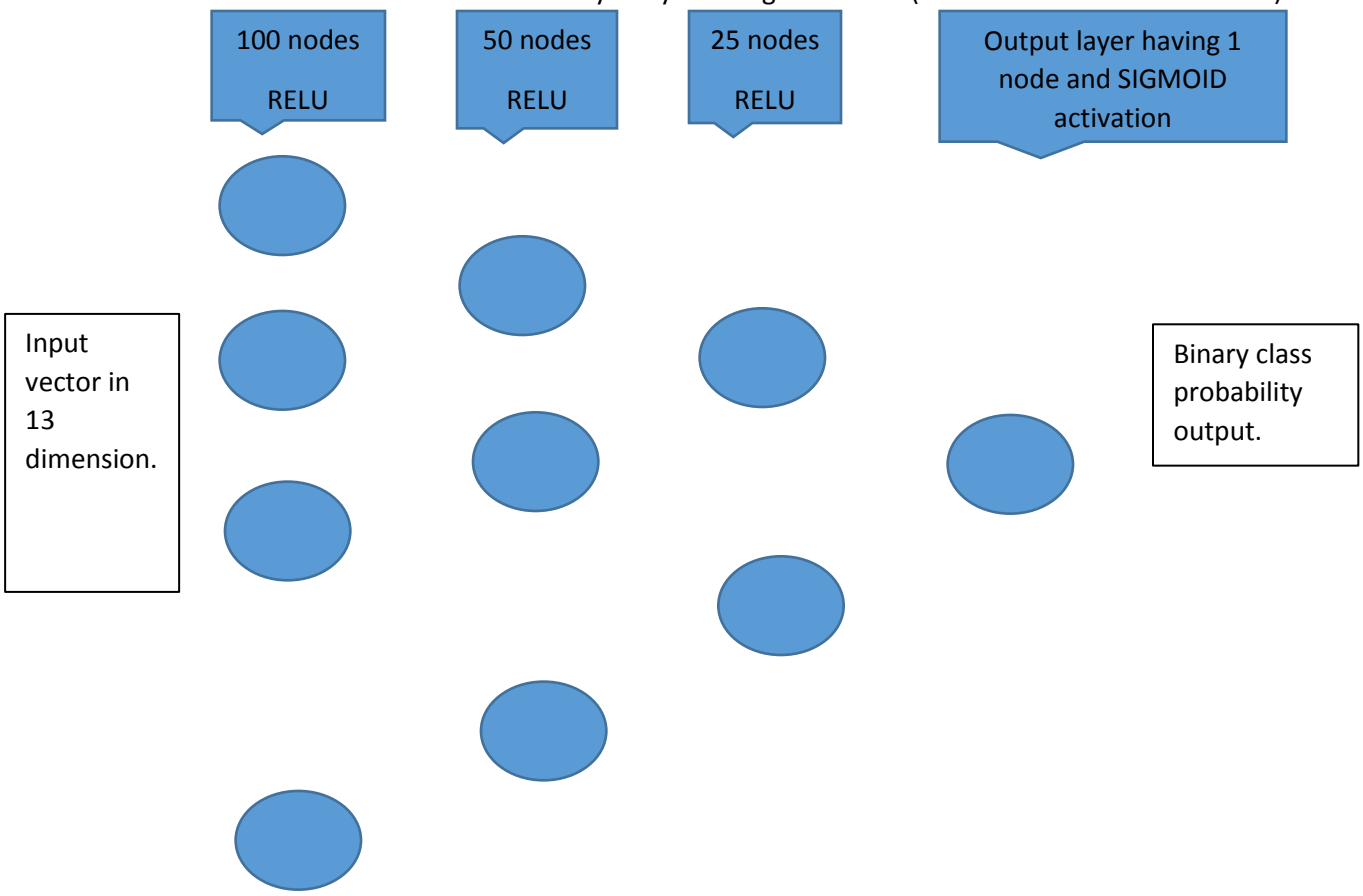
We wanted to remove outliers from the input data set, so that the model we train using the training data can be better tuned. In order to do this, we used k-means clustering to initially find the cluster centers. We set the value of k = 2 so that we segregate the input data into two clusters. Once we found the cluster centers, we calculated the minimum distance of each sample from each of the cluster centers. We then used this distance values to remove the outliers. The top 5 percent of the data which is farthest from the cluster centers is ignored. We have used the new input data set and applied neural networks and support vector machine models, which are described in detail in the next sections.

Neural Networks:

Today we are witnessing a renaissance in the world of neural networks. This has been primarily due to 2 reasons. The first is advancement in the field of big data technologies where we can capture, store and process effectively petabytes of data. The second factor is that of manufacture of very powerful GPU which can effectively crunch numbers in parallel. Thus, we huge amount of data and very powerful hardware to analyze the data. But there are challenges to be overcome when we select neural network as our model. The primary challenge is that the neural network is susceptible to overfitting very easily and the second challenge is that the neural networks have a lot of hyper parameters which need to be tuned. To help us in this we surveyed papers written by some of the pioneers (like hinton, lecun et.al) in this field.

For designing the network model and training we used (Keras + Theano) along with the standard Scikit-learn library. The network was trained on a NVidia GeForce GT 540 GPU.

The basic model which we used is best conveyed by the diagram below (nodes and activation function).



For fine tuning and implementation details we referred to the below 2 papers.

1. Efficient BackProp by LeCun et.al [http://yann.lecun.com/exdb/publis/pdf/lecun-98b.pdf]
2. Dropout: A Simple Way to Prevent Neural Networks from Overfitting by Srivastava, Hinton et.al. [http://www.jmlr.org/papers/volume15/srivastava14a.old/source/srivastava14a.pdf]

Some of the hyper-parameters that we used based on the above recommendation and for the rest we did a grid search to find the optimum values.

We have a binary classification problem. So, we need 1 output layer. For the number of input layer, we tried with 100, 50, and 20 layers and we got best results with 100 nodes in the input layer. We used the below heuristic to select the node of other layers. If layer i has n nodes and output layer has m nodes, then the layer i+1 should have (n+m)/2 nodes. This makes sure that the model is complex enough to recognize complex features in the data. But this complex network might lead to overfitting, which we will handle later.

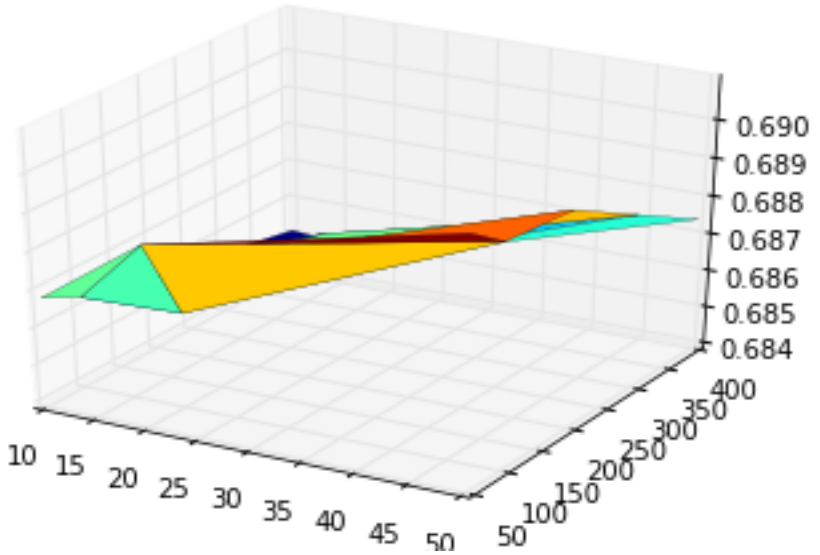
As far activation function is concerned, we needed binary probabilistic output. So, the activation for the output layer was intuitively selected as sigmoid. For the other layers, we had to choose between relu(rectifier linear unit), tanh and sigmoid. To effectively select the better activation function, we used the grid search technique. The results are shown as plots later. We found that the sigmoid function to have more error. While tanh and relu have comparable result. After consulting the paper 'Deep Sparse

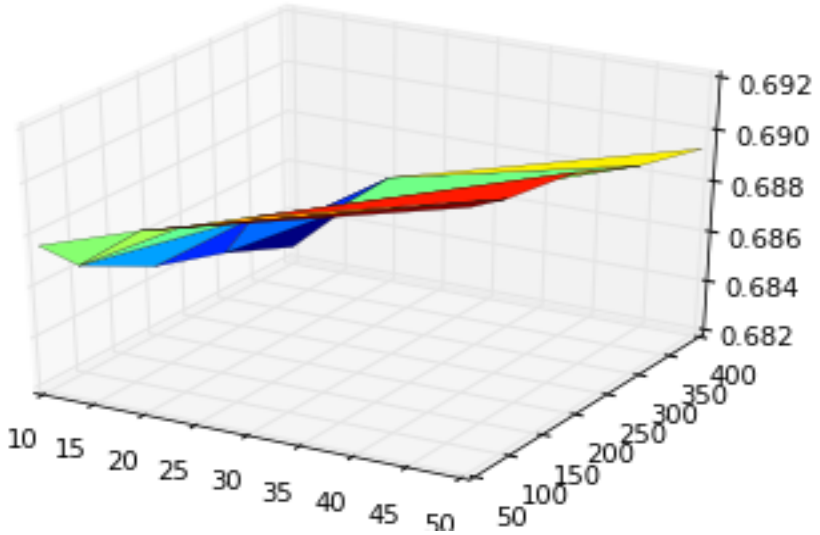
Rectifer Neural Networks' (by Bengio, Glorot et.al) we finalized on relu as the activation function for all but the last layer.

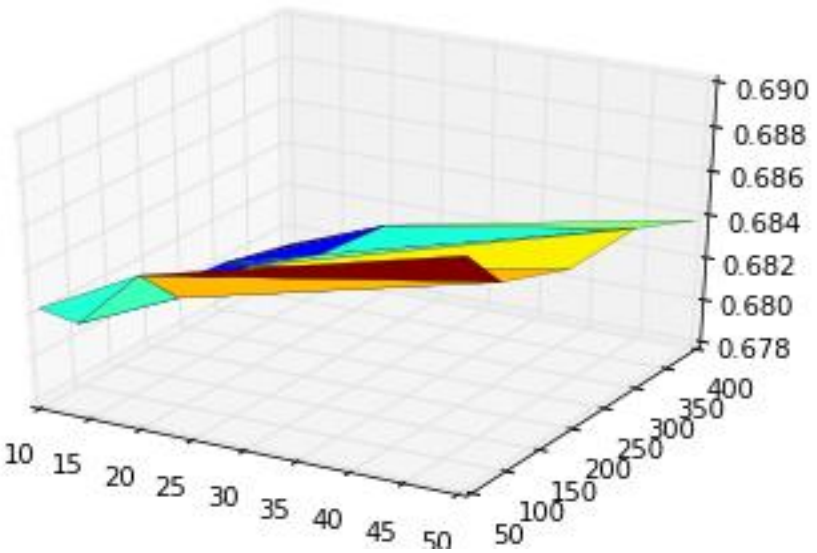
The weights were initialized with lecun uniform, which basically a Uniform initialization scaled by the square root of the number of inputs, as mentioned in the paper effective back prop. For optimization we compared stochastic gradient descent and adam(Adam: A Method for Stochastic Optimization, https://arxiv.org/abs/1412.6980v8) . Our analysis on a small sample of data showed that adam converged quicker. To compensate for overfitting, we used the technique of drop outs. The paper by Srivastav, Hinton et.al also advises to constrain the maximum norm of the weights to be 3 and the dropout rate set at 20%.

After setting up all this we were left with finalizing batch size and number of epochs. We again used grid search with batch size as [50, 100, 200, 300, 400] and number of epochs as [10, 20, 50]. While doing, this grid search we also varied the activation functions as [sigmoid, relu, tanh]. For validation during the grid search we randomly took out 10% of the training data for validation purpose. The results are presented below.

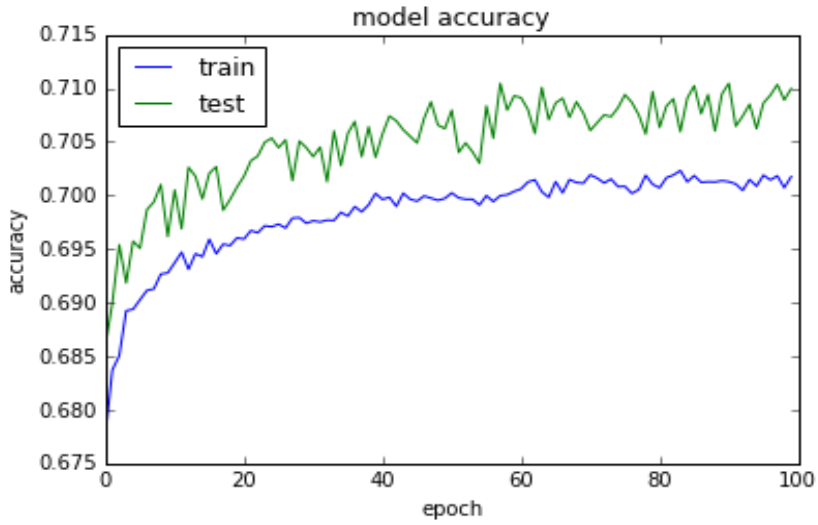
RELU			TANH			SIGMOID		
Acc.	Epochs	Batch #	Acc.	Epochs	Batch #	Acc.	Epochs	Batch #
0.6868	10	50	0.6876	10	50	0.6823	10	50
0.6887	20	50	0.6889	20	50	0.6847	20	50
0.6905	50	50	0.6919	50	50	0.6882	50	50
0.6863	10	100	0.6861	10	100	0.6807	10	100
0.6864	20	100	0.6883	20	100	0.6828	20	100
0.6898	50	100	0.6915	50	100	0.6862	50	100
0.6859	10	200	0.6846	10	200	0.6803	10	200
0.6870	20	200	0.6873	20	200	0.6825	20	200
0.6896	50	200	0.6911	50	200	0.6850	50	200
0.6855	10	300	0.6837	10	300	0.6800	10	300
0.6865	20	300	0.6859	20	300	0.6813	20	300
0.6885	50	300	0.6900	50	300	0.6851	50	300
0.6851	10	400	0.6825	10	400	0.6792	10	400
0.6849	20	400	0.6861	20	400	0.6810	20	400
0.6874	50	400	0.6893	50	400	0.6838	50	400


0.690
0.689
0.688
0.687
0.686
0.685
0.684
10 15 20 25 30 35 40 45 50
50 100 150 200 250 300 350 400


0.692
0.690
0.688
0.686
0.684
0.682
10 15 20 25 30 35 40 45 50
50 100 150 200 250 300 350 400



As we can see, the best results were found for smallest batch size (50) and the most number of epochs (50). The activation function relu and tanh gave almost comparable results. The results show that by having a smaller batch size, we are doing more updates and this we are getting better results. For the final training, we set the max epoch to 100. And while we were training the model we check pointed the weights whenever the accuracy on the validation data set was more than the checkpointed accuracy. Below is the plot of the result and we can see that accuracy on the validation data set almost stabilizes near the end of the plot.



We reload the weights of our model and predict the values with the test data set from kaggle.

Support Vector Machines:

A single learner is susceptible to overfitting. So, it is a good idea to test the data set with another model. Prior to the recent popularity of neural network, SVM was one of the better statistical models. Thus, we choose to explore SVM. SVM is a supervised learning model wherein given labelled training data, the model outputs an optimal hyperplane which categorizes new input data. SVM does this by creating a maximum margin partition between the feature space.

A standard linear SVM classifier finds a margin that separates positive and negative examples. The given input data set, however is not linearly separable. So, a linear SVM classifier would not work well. Hence for the given data set, a non-linear decision boundary is required. SVMs can produce a non-linear decision boundary using different kernel functions.

The kernel function we have used is the Gaussian radial basis function. The parameters that are relevant are C and gamma. The parameter C is the soft margin cost function which controls the influence of each individual support vector. A low C value makes the decision boundary smooth, which can lead to underfitting. While a high value of C, aims to classify the training data perfectly which can lead to overfitting.

The parameter gamma decides how much influence each training sample has on the model. It can be seen as the inverse of the radius of influence of samples selected by the model as support vectors. If gamma is too large, the radius of the area of influence of the support vectors only includes the support vector itself and this can lead to overfitting. If gamma is small, the model is too constrained and cannot capture the complexity or “shape” of the data. This can lead to underfitting.

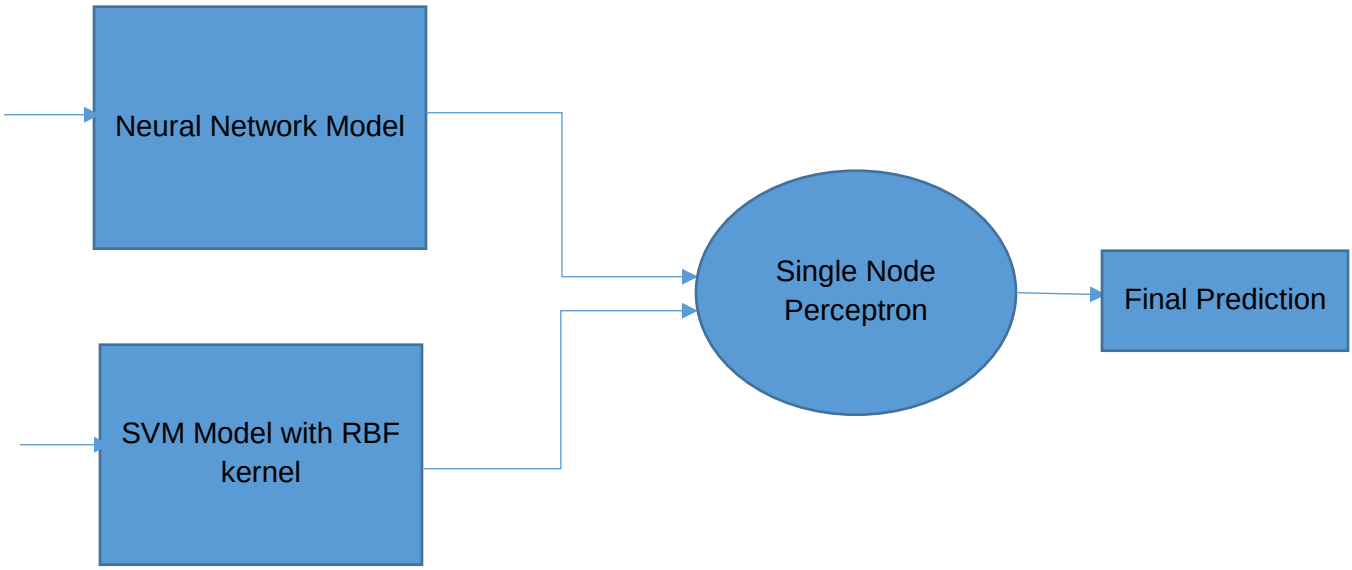
We have tried training the SVM model by varying the hyper parameters using grid search on 90pc train data (180k samples), but the process take a long time and times out. So, we had to reduce the training data set to 30k samples. We then ran the grid search by varying parameters C and gamma. The results are tabulated below.

Kernel Function	C value	Gamma	Accuracy
rbf	1	0.001	0.316
rbf	1	0.0001	0.316
rbf	10	0.001	0.670
rbf	10	0.0001	0.316
rbf	100	0.001	0.681
rbf	100	0.0001	0.677
rbf	1000	0.001	0.687
rbf	1000	0.0001	0.671

As can be seen from the table, the accuracy is highest (0.687) for the parameter values C = 1000 and Gamma = 0.001. We used these parameter values and used the model on the kaggle test input data to get the predicted output values. The AUC score obtained on kaggle was 0.63747.

Ensemble of Neural Network And SVM:

After we developed both the models, we linearly combined the output of both the models. To learn the weights of this combination we used a single node perceptron. The architecture of this approach is as below.



We used the sigmoid activation function as we wanted probabilities. Instead of again doing a grid search we used check pointing where we store the weights whenever we get a higher accuracy. The accuracy on our test data was found to be 0.689954. Thus, there was no improvement over all. The AUC score in KAGGLE was 0.66057.

Conclusion:

We have learnt a lot of machine learning algorithms in class. This project helped us to implement a few of them like neural networks, SVM, K-means clustering, PCA and learn how to apply them in real datasets. We also saw the effects of underfitting and overfitting of the models by varying different parameters.