Applied Machine Learning – Assignment 3

INTRODUCTION

Classification is a technique to categorize the data into a desired and distinct number of classes where we can assign label to each class. We find applications of classification in almost every industry such as speech recognition, biometric identification or in various important decisions of our life such as whether to set up a business in a location or not, to apply to a college or not, to buy a house in a neighborhood or not. The Classifiers can be binary classifier (with 2 distinct classes) or multi-class classifier (with more than 2 distinct classes). For this assignment I have experimented with Artificial Neural Network and K nearest Neighbors on two datasets.

• **Dataset1 –** The first dataset used in this assignment is the Facebook comment prediction dataset. As suggested, I have converted it to a binary classification problem by thresholding the output to a class label.

New_Target – This is the class label that is initialized as follows:

If Target Variable > 0, New_Target = 1 (If a post gets a comment it is labeled as 1)

If Target Variable <= 0, New_Target = 0 (If a post does not get a comment it is labeled as 0)

This dataset is interesting as it can help people and businesses understand what features are important for increasing the popularity of a product or a brand by measuring the response to advertisement posted on Facebook in terms of comments received from the people.

• **Dataset2 –** The second dataset used is the Admission Prediction dataset. Education is one of the biggest investments of a person's life, so every student should do proper research on various universities based on his/her skills and academic standing before applying for a particular university. The application process is tedious and requires lots of hard work, time and money, so proper research can help them save lots of time and money. In the dataset there are several qualitative and quantitative variables:

Admit – This is the class label that is initialized as follows:

If Chance of Admit > 0.6, Admit = 1 (If the chance of admit is greater than 0.6, it is labeled as 1)

If Chance of Admit <= 0.6, Admit = 0 (If the chance of admit is less than 0.6, it is labeled as 0)

I have taken 0.6 as the threshold here because most of the good colleges wants students that are far above average, and 60% is a good chance for a student to get admit in that college.

This dataset is interesting as it can help students to take the most crucial decision of their life and will prevent them from unnecessary hassle and expenditure during the application process as they can take more sound decisions.

Dataset 1(Facebook Comment Prediction Dataset)

Performed exploratory data analysis to find the relationship between different features and to understand the distribution of features. We found that range of values for different features differ a lot, so I performed feature scaling on these features. Performed outlier's detection and imputation. ***(*Detailed analysis done in the previous assignments)***

Artificial Neural Networks Implementation- Used keras to implement neural networks for classification using different activation functions and different hyperparameters

- Model 1: Used ReLu activation function in the hidden layers and Sigmoid function in the output layer.
  loss: 0.1035 - acc: 0.9648 - val_loss: 0.0901 - val_acc: 0.9774

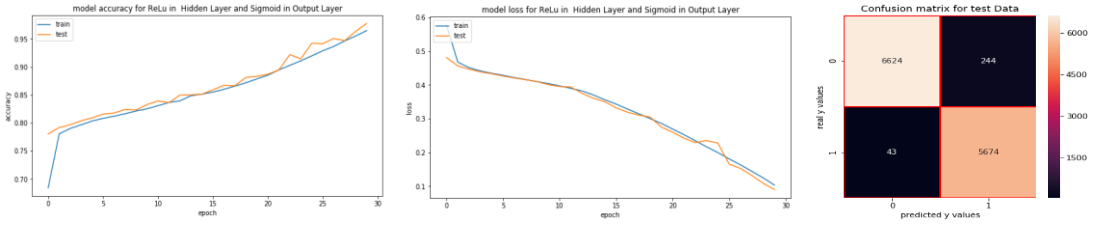


Fig.1

- Model 2: Used sigmoid activation function in the hidden and the output layers. loss: 0.3839 - acc: 0.8354 – val_loss: 0.3818 - val_acc: 0.8374

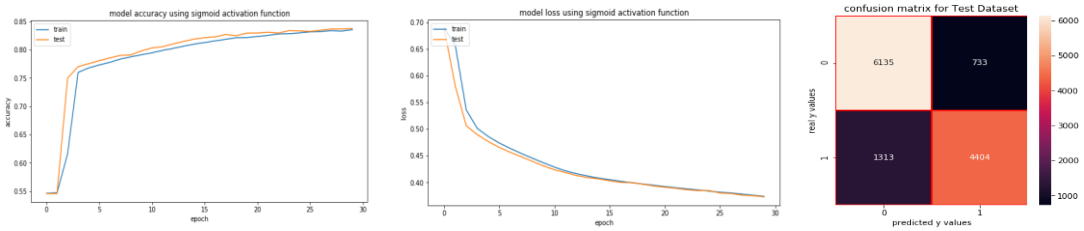


Fig. 2

- Model 3: Used sigmoid activation function in the hidden and the output layers. loss: 0.0144 - acc: 0.9959 - val_loss: 0.0216 - val_acc: 0.9940

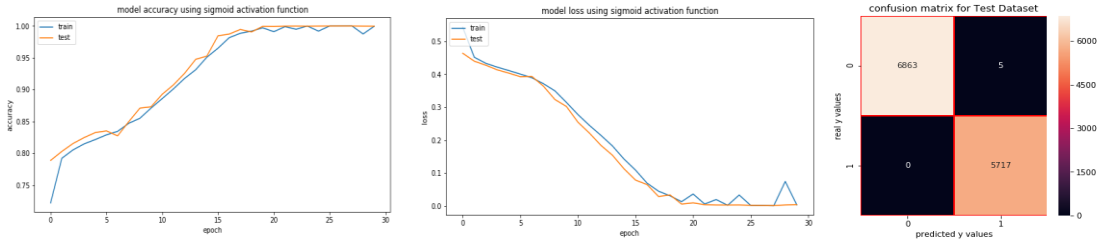


Fig. 3

- Model 4: Used sigmoid activation function in the hidden and the output layers. loss: 8.7237 - acc: 0.4528 - val_loss: 8.7002 - val_acc: 0.4543

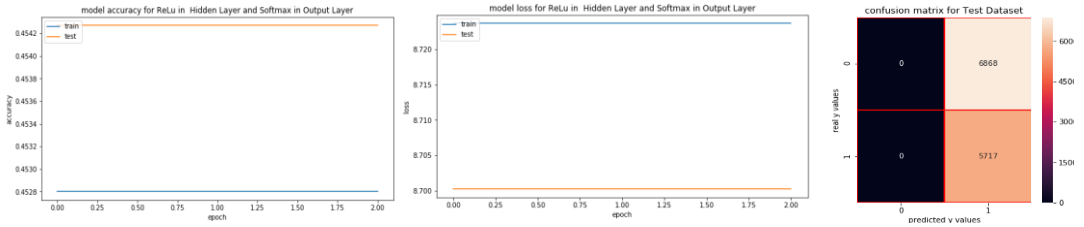


Fig. 4

We can experiment with other activation functions as well. From the above experiments we can conclude that Model 3 performs best on this dataset. So, I further experimented using this model by changing the parameters such as **number of layers, number of nodes in each layer, number of epochs and batch size and type of optimizer.**

Experiment with the number of layers: Increasing the number of hidden layers might improve the accuracy or might not, it really depends on the complexity of the problem that you are trying to solve. Increasing the number of hidden layers much more than the enough layers will cause accuracy in the test set to decrease, yes. It will cause your network to overfit to the training set, that is, it will learn the training data, but it won't be able to generalize to new unseen data.

As we can see that there is not much difference in the accuracy of the models having 2 and 3 hidden layers, so I will choose the model with 2 hidden layers.

---

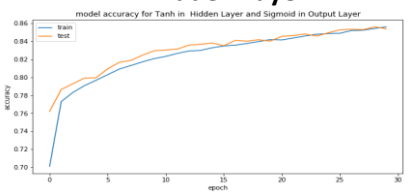
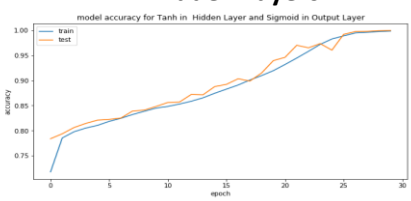
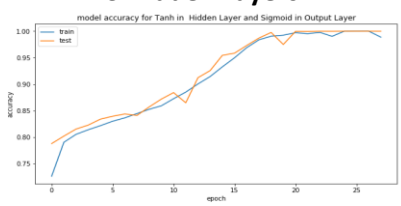
1 Hidden Layer	2 Hidden Layers	3 Hidden Layers
		
loss: 0.3358, acc: 0.8562 **val_loss: 0.3366, val_acc: 0.8540**	**loss: 0.0138, acc: 0.9983** **val_loss: 0.0099, val_acc: 0.9994**	**loss: 0.0476, acc: 0.9886** **val_loss: 0.0021, val_acc: 0.9998**

Fig. 5

Experiment with the number of nodes: We see the number of hidden neurons does affect the model performance. When a neural network has too few hidden neurons (< 16), it does not have the capacity to learn enough of the underlying patterns to distinguish between 0 – 9 effectively. When the neural network has >= 16 neurons, the neural network starts to do better. At increasing number of hidden neurons (>= 128), the number of hidden neurons does not help too much for this problem.

As we gradually reduce the number of nodes in the hidden layers the number of epochs required to converge increases and the accuracy increases. I will choose a model with a smaller number of nodes per layer for this problem

---

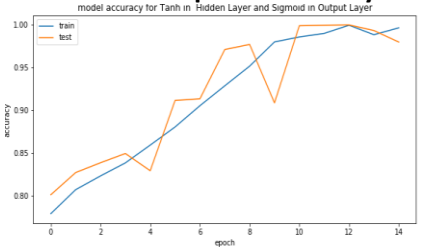
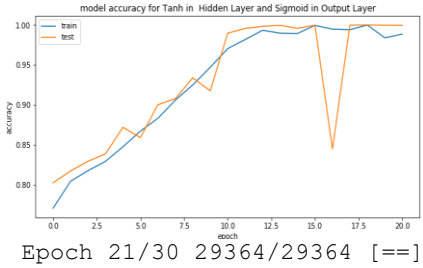
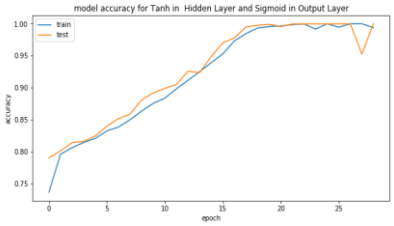
100 Nodes per Hidden Layer	**50 Nodes per Hidden Layer**	**10 Nodes per Hidden Layer**
		
`Epoch 15/30 29364/29364 [==] - 1s 48us/step - loss: 0.0166 - acc: 0.9964 - val_loss: 0.0557 - val_acc: 0.9797`	`Epoch 21/30 29364/29364 [==] - 1s 49us/step - loss: 0.0446 - acc: 0.9886 - val_loss: 0.0062 - val_acc: 0.9997`	`Epoch 29/30 29364/29364 [==] - 1s 46us/step - loss: 0.0274 - acc: 0.9938 - val_loss: 0.0018 - val_acc: 0.9999`

Fig. 6

Experiment with the batch size: Advantages of using a batch size < number of all samples: It requires less memory. Since you train the network using fewer samples, the overall training procedure requires less memory. That's especially important if you are not able to fit the whole dataset in your machine's memory. Typically networks train faster with mini-batches. That's because we update the weights after each propagation.

Disadvantages of using a batch size < number of all samples: The smaller the batch the less accurate the estimate of the gradient will be.

On increasing the batch size, the model is trained faster but the accuracy of the model decreases. There is a tradeoff between computational speed and accuracy. For this dataset I have chosen the batch size of 32 because the model is having the highest accuracy and it does not require much time to train the model

---

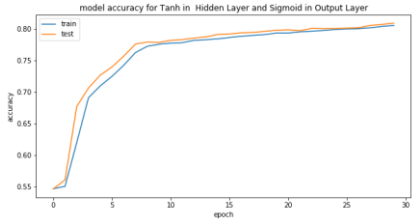
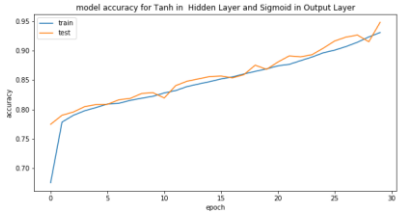
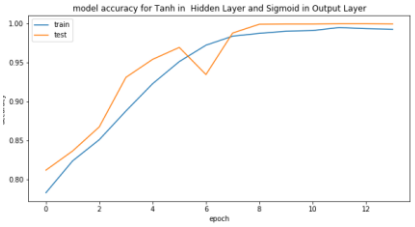
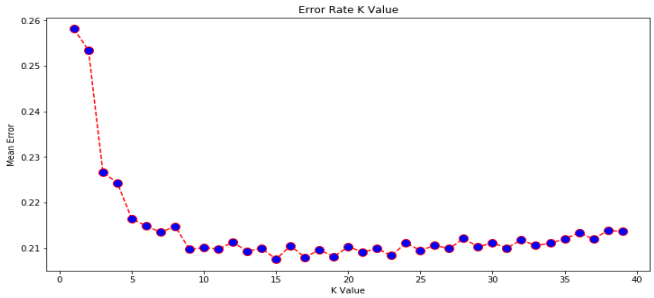
Batch Size of 1000	Batch Size of 100	Batch size of 4
		
Epoch 30/30 29364/29364 [==============================] - 0s 4us/step - loss: 0.4327 - acc: 0.8054 - val_loss: 0.4339 - val_acc: 0.8091	Epoch 30/30 29364/29364 [==============================] - 0s 16us/step - loss: 0.1813 - acc: 0.9308 - val_loss: 0.1705 - val_acc: 0.9482	Epoch 14/30 29364/29364 [==============================] - 10s 354us/step - loss: 0.0277 - acc: 0.9928 - val_loss: 0.0050 - val_acc: 0.9996

Fig. 7

Experimented by using **K fold cross validation with 5 folds**, for the model with 2 hidden layers, using tanh and sigmoid activation function and batch size of 32 got the cross_val_score value equal to 0.548. I have decided to use Train/Test Split method for this dataset because the amount of time required to train dataset is lower than cross validation and there is no significant improvement in the model performance if I use K- fold cross validation for this dataset (the model accuracy has decreased). Generally, cross validation performs much better on the unseen data as compared to Train/Test split method. We can also experiment with different number of folds (k values) and compare the model performance.

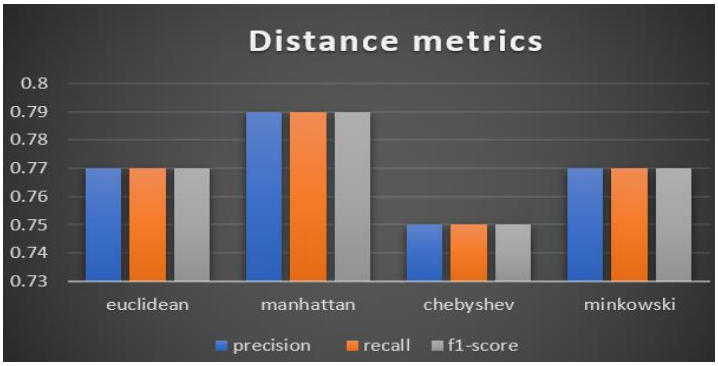
So, the final model that I am considering for this dataset is the one with 2 hidden layers,4 neurons in each layer, with batch size of 32 and tanh and sigmoid activation functions for hidden layers and output layer respectively as I got the maximum accuracy and least loss for this combination. We can get better model by further experimentation and by trying various permutations of model parameters in terms of computational speed and complexity.

K Nearest Neighbor Implementation- Experimented with different values for the number of neighbors.



The figure shows that the mean error rate is minimum if we consider 9 number of neighbors

Fig. 8



The figure shows that Manhattan distance metrics perform better for this dataset. The model accuracy using k fold cross validation method is 0.658. So, I will use train/test split method for further experiments

Fig. 9

The best model for this dataset using knn will be one having 9 neighbors and Manhattan distance

Comparison of Algorithms:

Algorithm	Accuracy
SVM	0.8
Decision Tree	0.97
Boosting	0.998
ANN	0.999
KNN	0.79

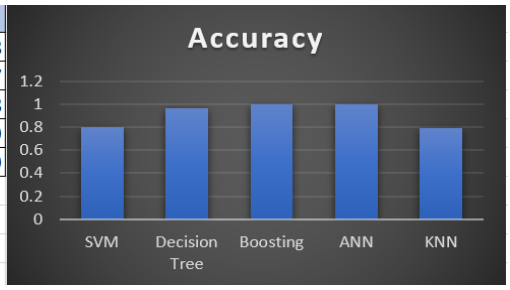


Fig. shows the comparison between different algorithms that we have experimented for this dataset. We can see that ANN performs the best in terms of accuracy as well as other performance metrics (precision, recall, f1 score).

Fig. 10

Dataset 2 (Admission Prediction dataset)

Performed exploratory data analysis to find correlation among features. Performed feature scaling and outlier detection.
(*Detailed analysis done in the previous assignments)

Artificial Neural Networks Implementation- Used keras to implement neural networks for classification using different activation functions and different hyperparameters

- Model 1: Used ReLu activation function in the hidden layers and Sigmoid function in the output layer. loss: 0.087 9 - acc: 0.9898 - val_loss: 0.0867 - val_acc: 0.9891

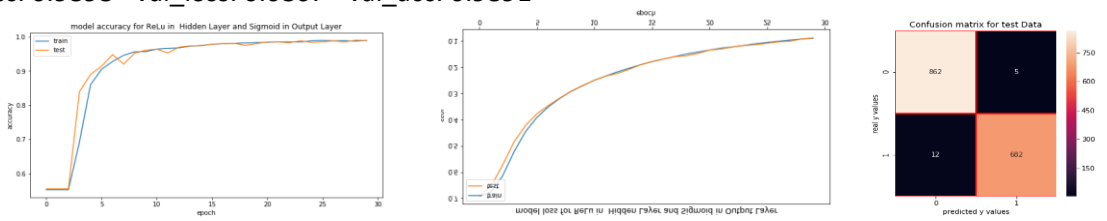


Fig. 11

- Model 2: Used ReLu activation function in the hidden layers and SoftMax function in the output layer. loss: 8.80 73 - acc: 0.4476 - val_loss: 8.8546 - val_acc: 0.4446

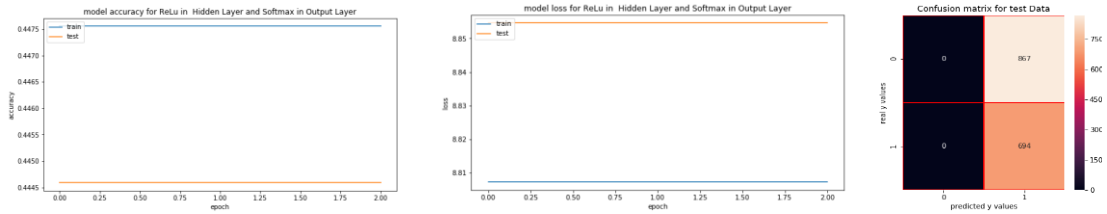


Fig. 12

- Model 3: Used Sigmoid activation function. loss: 0.1747 - acc: 0.9703 - val_loss: 0.1819 - val_acc: 0.9641

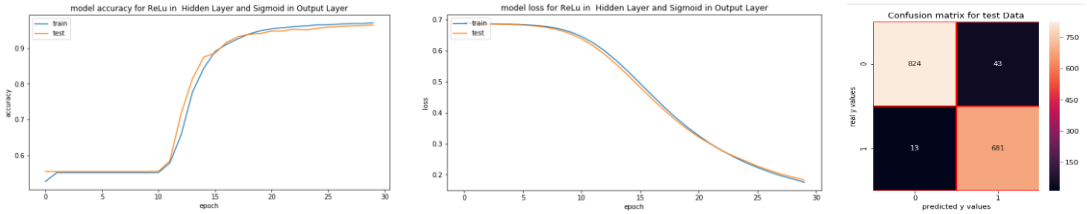


Fig. 13

- Model 4: Used Tanh activation function in the hidden layers and Sigmoid function in the output layer. loss: 0.033 2 - acc: 0.9920 - val_loss: 0.0408 - val_acc: 0.9833

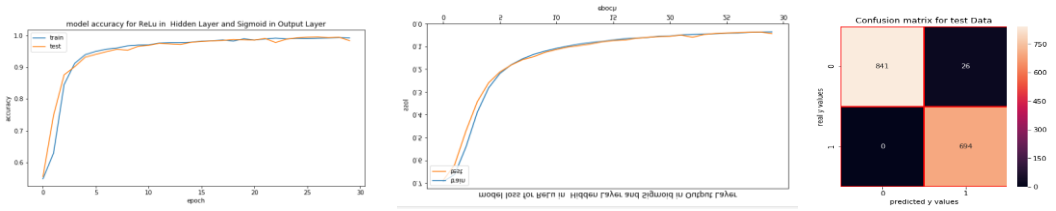


Fig. 14

We can experiment with other activation functions as well. From the above experiments we can conclude that Model 1 and Model 4 performs best on this dataset. So, I further experimented using these models by changing the parameters such as **number of layers, number of nodes in each layer, number of epochs and batch size and type of optimizer.**

As we can see the model with 2 hidden layers has the highest accuracy, so I will choose the model with 2 hidden layers

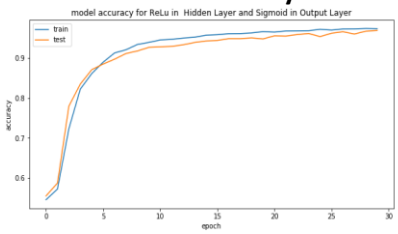
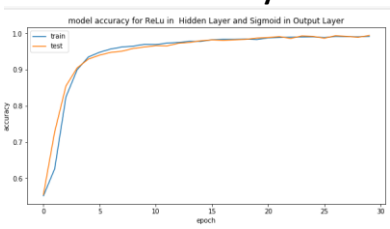
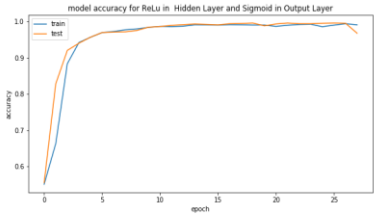
1 Hidden Layer	2 Hidden Layers	3 Hidden Layers
		
loss: 0.1022, acc: 0.9731 **val_loss: 0.1121, val_acc: 0.9686**	**loss: 0.0331, acc: 0.9923** **val_loss: 0.0339, val_acc: 0.9949**	**loss: 0.0270, acc: 0.9904** **val_loss: 0.0330, val_acc: 0.9865**

Fig. 15

As we gradually reduce the number of nodes in the hidden layers the number of epochs required to converge increases and the accuracy increases. I will choose a model with a smaller number of nodes per layer for this problem

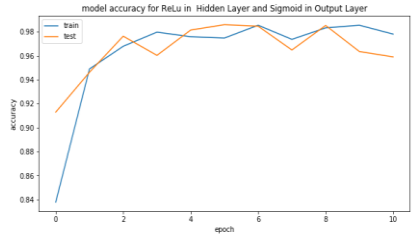
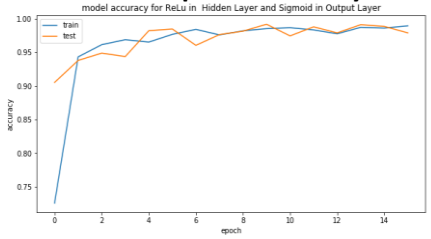
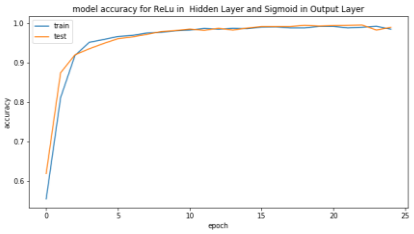
100 Nodes per Hidden Layer	50 Nodes per Hidden Layer	10 Nodes per Hidden Layer
		
`Epoch 11/30 3642/3642 [====] - 0s 83us/step - loss: 0.0489 - acc: 0.9780 - val_loss: 0.0896 - val_acc: 0.9590`	`Epoch 16/30 3642/3642 [====== ====] - 0s 84us/step - loss: 0 .0306 - acc: 0.9893 - val_loss : 0.0421 - val_acc: 0.9789`	`Epoch 25/30 3642/3642 [======= ===] - 0s 81us/step - loss: 0. 0344 - acc: 0.9849 - val_loss: 0.0270 - val_acc: 0.9898`

Fig.16

On increasing the batch size, the model is trained faster but the accuracy of the model decreases. For this dataset I have chosen the batch size of 32 because the model is having the highest accuracy and it does not require much time to train the model

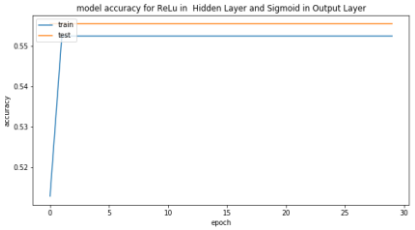
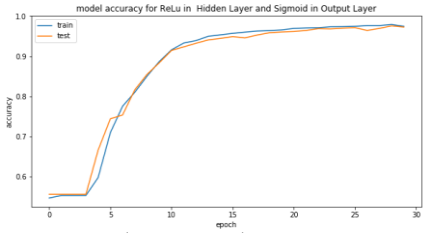
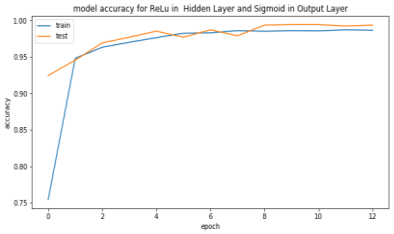
Batch Size of 1000	Batch Size of 100	Batch size of 4
		
Epoch 30/30 3642/3642 [==========] - 0s 3us/step - loss: 0.6765 - acc: 0.5524 - val_loss: 0.6771 - val_acc: 0.5554	Epoch 30/30 3642/3642 [==========] - 0s 24us/step - loss: 0.1011 - acc: 0.9745 - val_loss: 0.1085 - val_acc: 0.9725	Epoch 13/30 3642/3642 [==========] - 2s 645us/step - loss: 0.0357 - acc: 0.9865 - val_loss: 0.0252 - val_acc: 0.9936

Fig. 17

Experimented by using **K fold cross validation with 5 folds**, for the model with 2 hidden layers, using tanh and sigmoid activation function and batch size of 32 got the cross_val_score value equal to 0.547. I have decided to use Train/Test Split method for this dataset because the amount of time required to train dataset is lower than cross validation and there is no significant improvement in the model performance if I use K- fold cross validation for this dataset (the model accuracy has decreased).

So, the final model that I am considering for this dataset is the one with 2 hidden layers,4 neurons in each layer, with batch size of 32 and ReLu and sigmoid activation functions for hidden layers and output layer respectively as I got the maximum accuracy and least loss for this combination. We can get better model by further experimentation and by trying various permutations of model parameters in terms of computational speed and complexity.

K Nearest Neighbor Implementation- Experimented with different values for the number of neighbors.

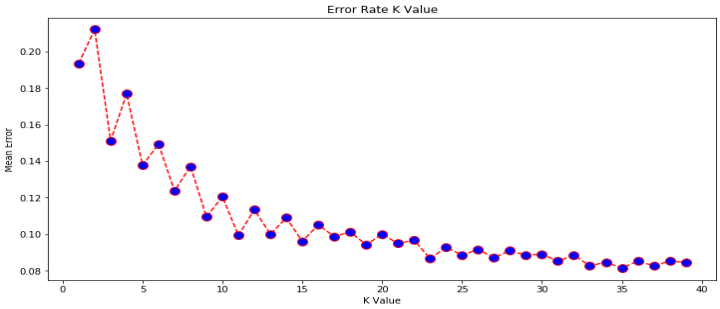


Fig. 18 shows that the mean error rate is minimum if we consider 23 number of neighbors

Fig. 18

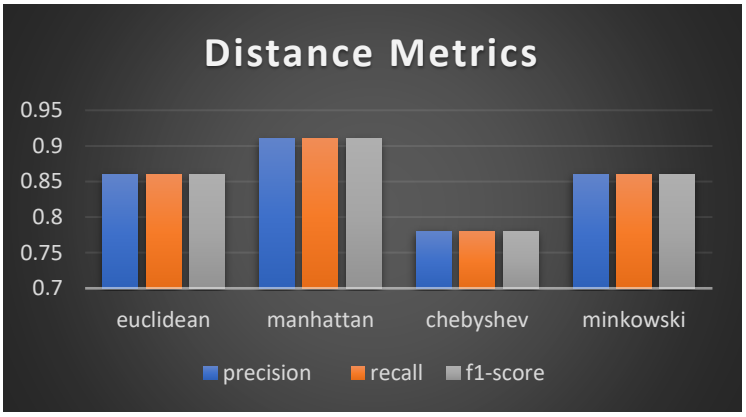


Fig. 19 shows that Manhattan distance metrics perform better for this dataset. The model accuracy using k fold cross validation method is 0.667. So, I will use train/test split method for further experiments

Fig. 19

The best model for this dataset using knn will be one having 23 neighbors and Manhattan distance

Comparison of Algorithms:

Algorithm	Accuracy
SVM	0.986
Decision Tree	1
Boosting	1
ANN	0.995
KNN	0.91

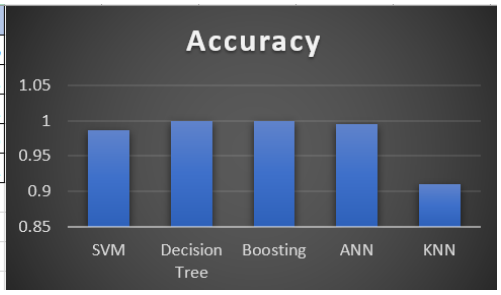


Fig. 20

Fig.20 shows the comparison between different algorithms that we have experimented with in this assignment. We can see that both decision tree and boosted algorithm performs better than other algorithms in terms of accuracy for this dataset. We can use other performance metrics as well depending on the requirement to compare the model performance.

Note: *I have used **Adam** as the optimization algorithm for both the datasets while implementing neural networks.*
Adam works well in practice and compares favorably to other adaptive learning-method algorithms as it converges very fast and the learning speed of the Model is quiet Fast and efficient, and it rectifies every problem that is faced in other optimization techniques such as vanishing Learning rate, slow convergence or High variance in the parameter updates which leads to fluctuating Loss function.

Conclusion

There is no one algorithm for all the types of data we should experiment with different types and different variations (by tuning the parameter values) of algorithm to find the best one for our dataset.

Algorithms	Stengths	Weekness
Support Vector Machines	SVM's can model non-linear decision boundaries, and there are many kernels to choose from.	SVM's are memory intensive, trickier to tune due to the importance of picking the right kernel, and don't scale well to larger datasets.
	They are also robust against overfitting, especially in high-dimensional space, no distribution requirement and doesn't suffer from multicollinearity	
Decision Trees	Simple to understand, interpret, visualize.	Decision-tree learners can create over-complex trees that do not generalize the data well. This is called overfitting.
	The internal workings are capable of being observed and thus make it possible to reproduce work.	Decision trees can be unstable because small variations in the data might result in a completely different tree being generated. This is called variance, which needs to be lowered by methods like bagging and boosting.
	Can handle both numerical and categorical data. Performs well on large datasets and extremely fast.	Decision tree learners create biased trees if some classes dominate. It is therefore recommended to balance the data set prior to fitting with the decision tree.
	Decision trees implicitly perform variable screening or feature selection.	
Boosted Decision Trees	GBTs build trees one at a time, where each new tree helps to correct errors made by the previously trained tree.	Prone to over-fitting and they are harder to get right. It is also more prone to over-fitting.
	With each tree added, the model becomes even more expressive. GBDTs will usually perform better than Random Forest and they have more hyper-parameters to tune.	GBDT training generally takes longer because trees are built sequentially.
Artificial Neural Networks	Neural Network is good for unstructured datasets like image, audio, and text and it does not perform well on structured data sets.	The probably best-known disadvantage of Neural Networks is their "black box" nature, meaning that you don't know how and why your NN came up with a certain output.
		It is very complex to apply.
		Requires high computational power.
		Training time is too high.
		It is not as easy as building a model using scikit-learn/caret.
K Nearest Neighbors	Simple to implement	Large search problem to find nearest neighbours
	Flexible to feature/distance choices	Storage of data
	Naturally handles multi-class cases	Must know we have a meaningful distance function
	Can do well in practice with enough representative data	

We can experiment with more permutations of hyperparameters to improve the model performance.

We can choose the best model as per our requirements. We cannot rank the models as per just one criterion such as accuracy we should also consider other factors such as computational speed, model complexity, interpretability etc.