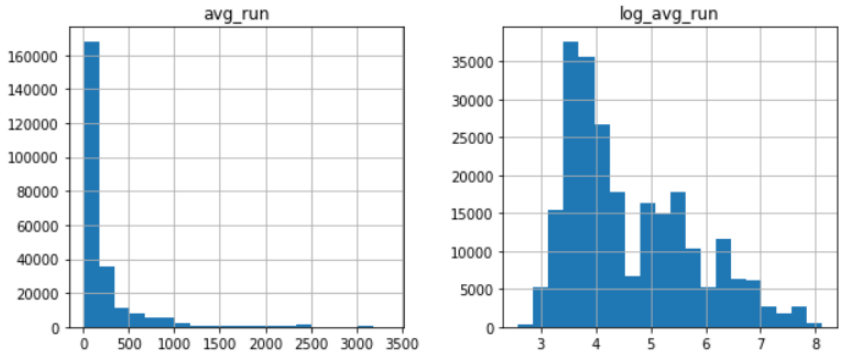
Artificial Neural Networks and K Nearest Neighbors Techniques

Data Description & Preparation

Dataset 1 - SGEMM GPU kernel performance Data

This data set measures the running time of a matrix-matrix product A*B = C, where all matrices have size 2048 x 2048, using a parameterizable SGEMM GPU kernel with 241600 possible parameter combinations. For each tested combination, 4 runs were performed.

- The dataset has 241600 observations with 18 features.
- None of the column contains any missing value.
- The target variable will be the average of the four run times.
- After computing the average, the data in the target variable is right skewed. Taking the log of the values reduces the skewness of the target variable.



- For this report, we will use all the independent variables to train our models.
- The data is divided into train test datasets in 70:30 ratio. The data is then scaled appropriately.

Dataset 2 - Letter Recognition Data

The objective is to identify each of black-and-white rectangular pixel displays as one of the 26 capital letters in the English alphabet. The character images are based on 20 different fonts and each letter within these 20 fonts is randomly distorted to produce a file of 20,000 unique stimuli. Each stimulus is converted into 16 primitive numerical attributes (statistical moments and edge counts) which were then scaled to fit into a range of integer values from 0 through 15.

- The dataset has 20000 observations with 17 features.
- None of the column contains any missing value.
- The target variable will be column “Letter”. The rest of the 16 variables, of datatype Integer, will be used as predictors.
- For this report, we will use all the independent variables to train our models.
- The data is divided into train test datasets in 80:20 ratio. The data is then scaled appropriately.

Artificial Neural Networks

Hyper-parameters

- **Layer size** – This tells the number of hidden layers and number of neurons at each layer.
- **Activation Function** – The activation functions used are identity (f(x) = x), logistic sigmoid function (f(x) = 1 / (1 + exp(-x))), hyperbolic tan function (f(x) = tanh(x)) and rectified linear unit function (f(x) = max(0, x)).
- **Solver Function** – This parameter is the solver for weight optimization.
- **Alpha** - Alpha is a parameter for regularization term that combats overfitting by constraining the size of the weights. Increasing alpha may fix high variance (a sign of overfitting) by encouraging smaller weights. Similarly, decreasing alpha may fix high bias (a sign of underfitting) by encouraging larger weights, potentially resulting in a more complicated decision boundary.

Test Data Accuracies

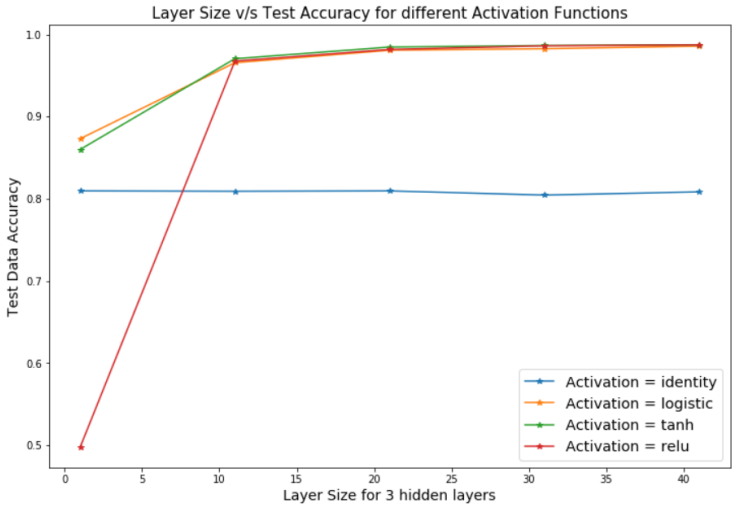
Dataset 1 – Accuracy = 98.78%, Parameters - hidden_layer_sizes=(40,40,40), solver='adam', alpha=0.00001, activation='relu', iter=200

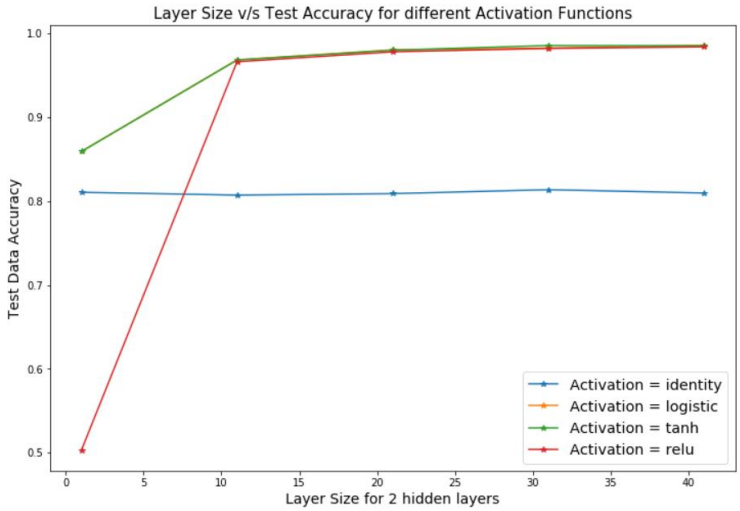
Dataset 2 – Accuracy = 96.15%, Parameters - hidden_layer_sizes=(100,100,100), solver='adam', alpha=0.0001, activation='relu', iter= 300

Experimentation with Hidden Layers and Activation Function

We have run ANN with different activation functions of identity, logistic, tanh and relu. Along with this we have varied the number of hidden layers and the number of neurons within each layer.

Dataset 1





The models are run by varying the hidden layers
from 1 to 3 and varying the number of nodes from 1 to 50 at each layer.

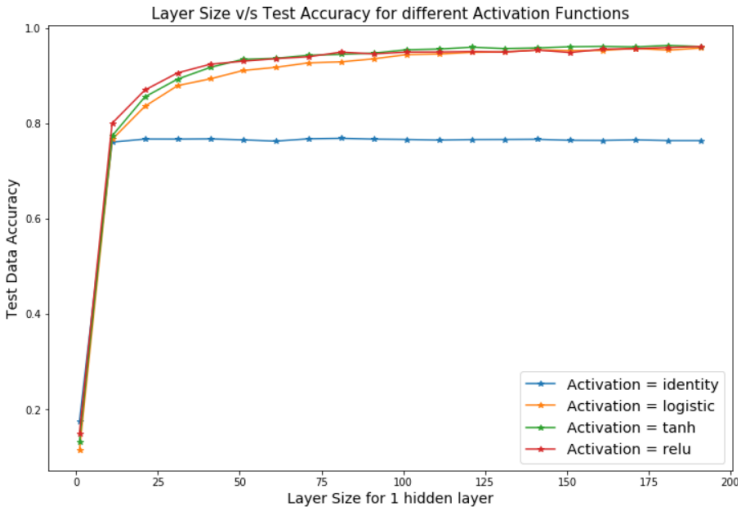
We notice from the runs of all 1-layer, 2-layer and 3-layer models that the activation function 'identity' is not performing as good as the other functions. Activation function 'relu' is not performing good with just one neuron in a layer. Rest the results given by the three activation functions are comparable for a greater number of neurons.

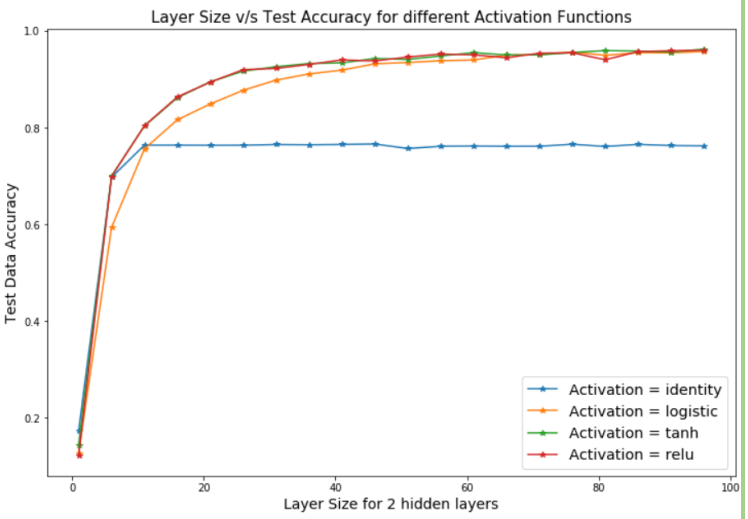
Also, with the increase in the number of neurons at each layer, the accuracy increases. This is expected behavior because of increase in computation. We also notice that maximum accuracy is achieved by using 3 hidden layers.

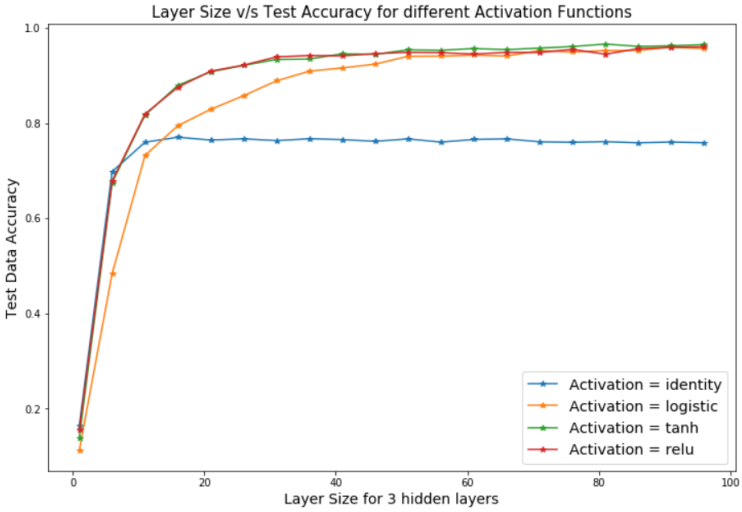
The maximum accuracies for different layers of model are as below:

	Max Test Accuracy (percentage)	Activation Function	Number of Nodes at Each Layer
1 hidden layer	97.884	tanh	41
2 hidden layer	98.557	tanh	31
3 hidden layer	98.740	tanh	41

Dataset 2







The models are run by varying the hidden layers from 1 to 3 and varying the number of nodes from 1 to 100 at each layer.

In Dataset 2 also, we notice that from the runs of all 1-layer, 2-layer and 3-layer models the activation function 'identity' is not performing as good as the other functions.

Initially the test data accuracy increases with increase in the number of neurons at each hidden layer, but after a point the accuracy is getting constant. After that point further

increase in number of nodes is just adding extra computation and not leading to an increase in accuracy. Like Dataset 1, we also notice that maximum accuracy is achieved by using 3 hidden layers.

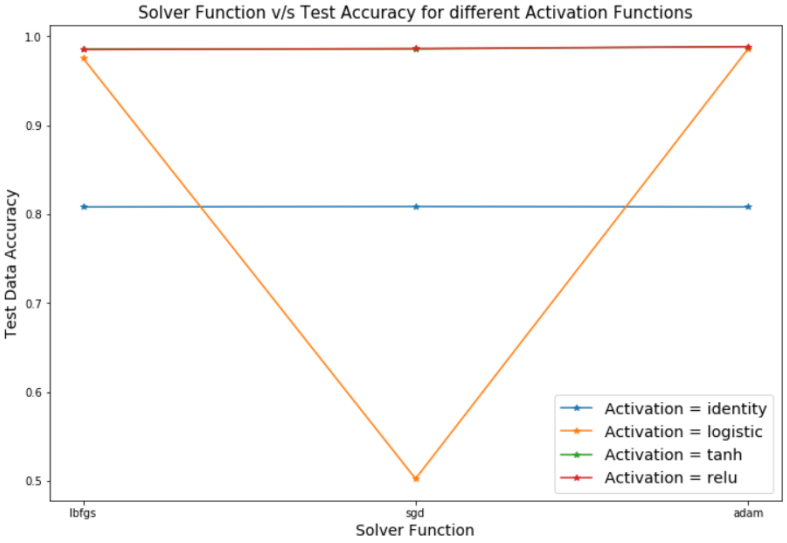
The maximum accuracies for different layers of model are as below:

	Max Test Accuracy (percentage)	Activation Function	Number of Nodes at Each Layer
1 hidden layer	96.333	tanh	181
2 hidden layer	96.283	tanh	96
3 hidden layer	96.617	tanh	81

Experimentation with Solver Functions and Activation Function

In this experiment we have varied the solver function with values 'lbfgs','sgd' and 'adam' along with the activation functions.

Dataset 1



The models are run with the following parameters:

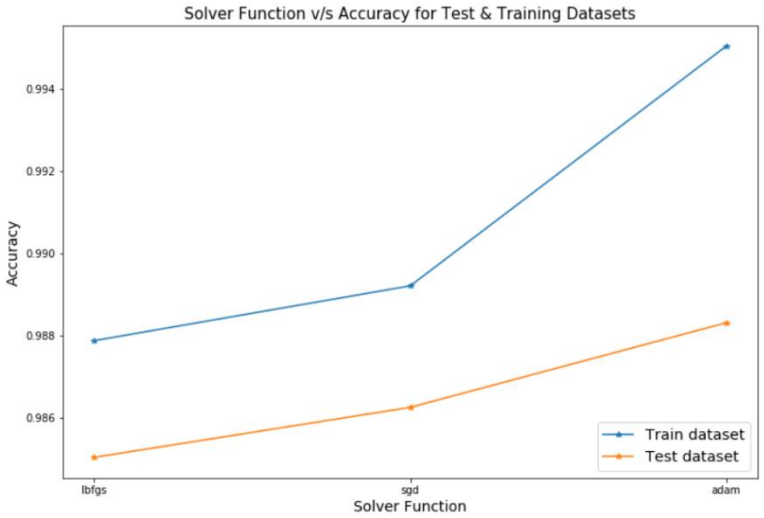
hidden_layer_sizes = (50,50,50) and max_iter = 300

We observe that the solver function sgd is giving the least accuracy with the activation function logistic.

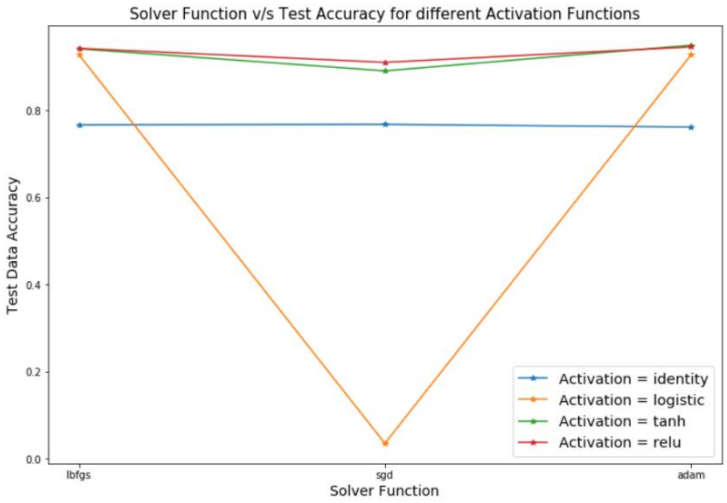
Besides this, the activation functions are giving comparable results for all the solver functions. As noticed before, the activation function identity is performing poorly as compared with others.

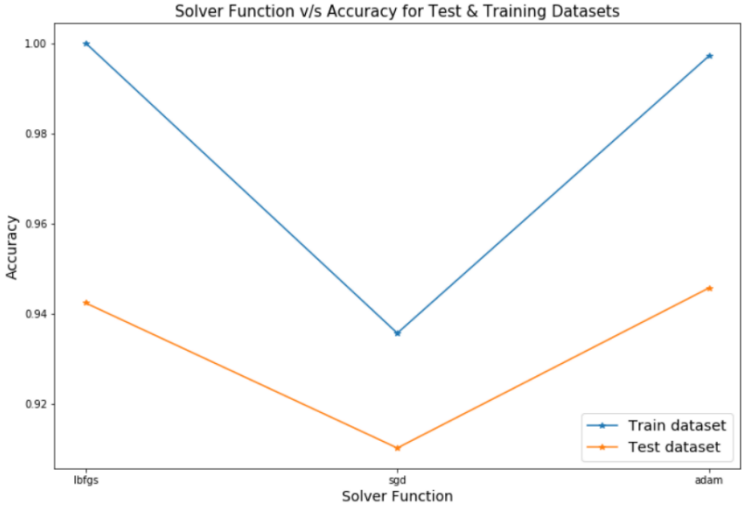
The graph on the right gives test and train data accuracies with the parameters mentioned above and the activation function ‘relu’. We observe that, although the difference in the accuracies is pretty small, the solver function ‘adam’ is performing the best in both train and test sets.

The best accuracy we get is **98.841%** with solver function adam.



Dataset 2





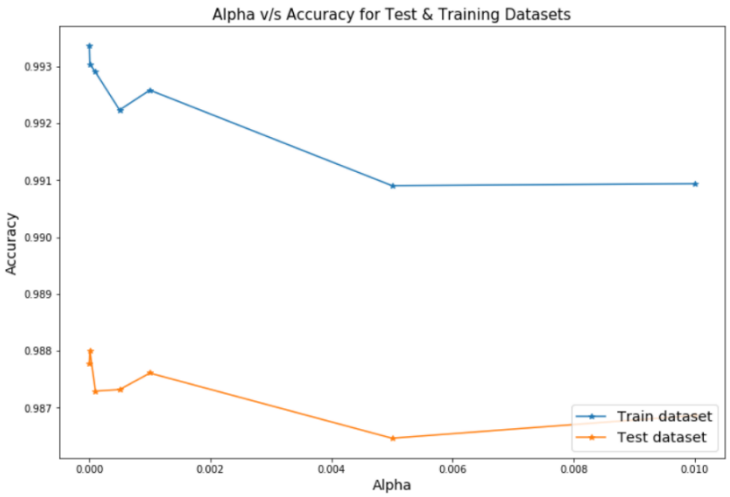
We run the model using the same parameters
as Dataset 1. We observe that the solver function sgd is performing the worst among all the solver functions for all the activation functions.

The graph on the right, gives test and train data accuracies with the parameters mentioned above and the activation function 'relu'. The functions lbfgs and adam are giving comparable results. The maximum accuracy we get is **94.95%** for solver function adam.

Experimentation with Alpha

Alpha combats overfitting by constraining the size of the weights. In this experiment we have varied the values of alpha with values [0.000001, 0.00001, 0.0001, 0.0005, 0.001, 0.005, 0.01].

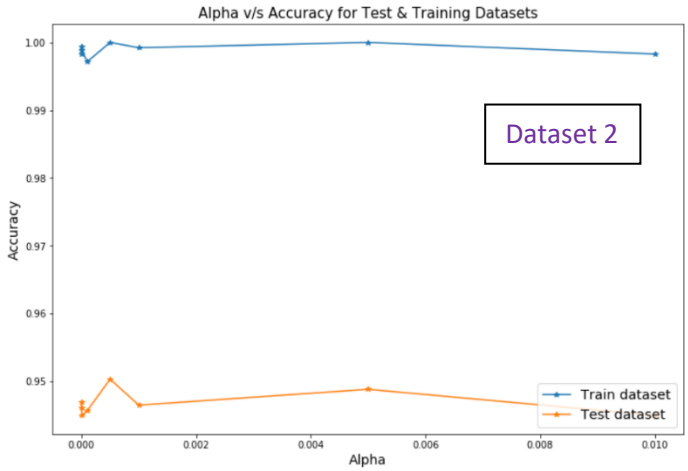
Dataset 1



For dataset 1, the models are run with the following parameters:

hidden_layer_sizes = (40,40,40), max_iter = 200 and activation_function = 'relu'

The high values of alpha are giving low accuracies because it is underfitting the training data and is not able to gauge the shape of the data. The low values of alpha are prone to overfitting. This can lead to low test data accuracies. Therefore, the optimal alpha for this dataset is 0.0001, which is high enough to gauge the shape of the data and low enough to not overfit the training dataset. The accuracy we get at this alpha value is **98.72%**.



For dataset 2, the models are run with the following parameters:

hidden_layer_sizes = (50,50,50), max_iter = 300 and activation_function = 'relu'

The optimal alpha for this dataset is 0.0001, which is high enough to gauge the shape of the data and low enough to not overfit the training dataset. The accuracy we get at this alpha value is **95.03%**.

K Nearest Neighbors

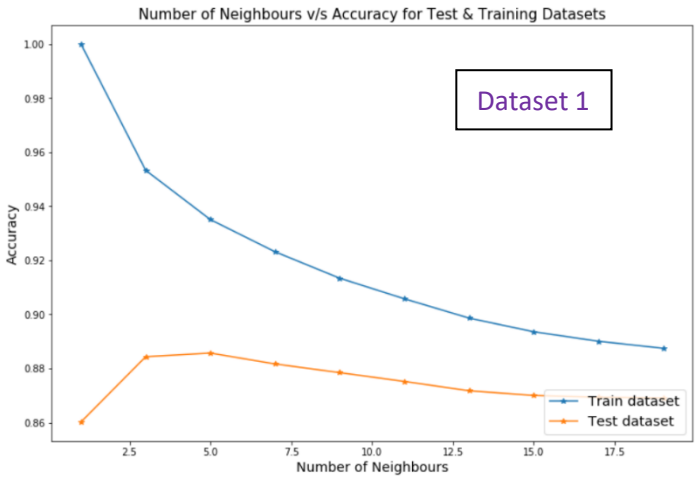
Hyper-parameters

- **Number of Neighbors** – This is the number of neighbors to use for classifying a new record
- **Weight** – This is the weight function used in prediction. 'Uniform' means all points in each neighborhood are weighted equally. 'Distance' means weight is the inverse of their distance.
- **Algorithm** – This is the algorithm used to compute the nearest neighbors. The options are 'auto', 'ball_tree', 'kd_tree' and 'brute'.
- **P** – P=1 is for manhattan distance. P=2 is for Euclidean distance. For arbitrary p, minkowski distance is used.

Test Data Accuracies

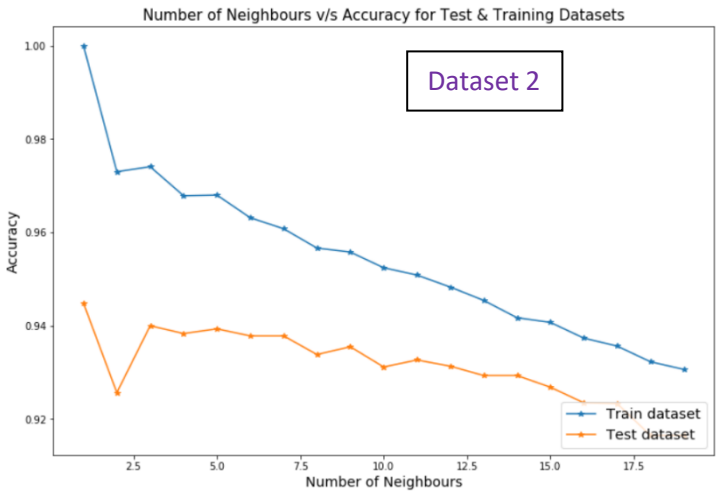
Dataset 1 – Accuracy = 91.68%, Parameters - n_neighbors=5, weights='distance', p=1

Dataset 2 – Accuracy = 95.65%, Parameters - n_neighbors=5, weights='distance', p=1



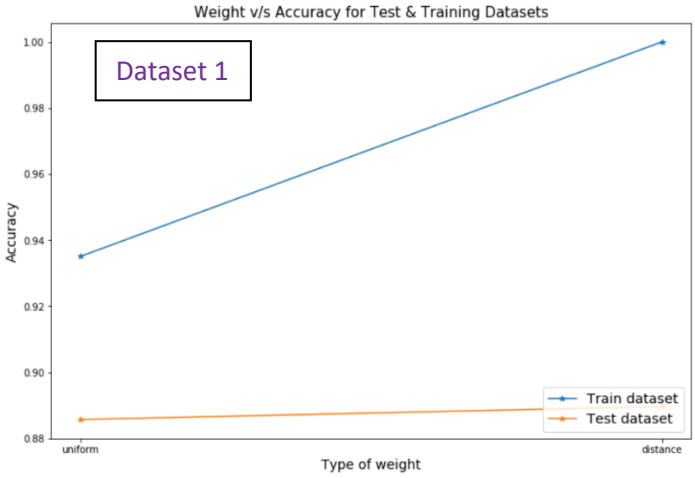
Experimentation with Number of Neighbors

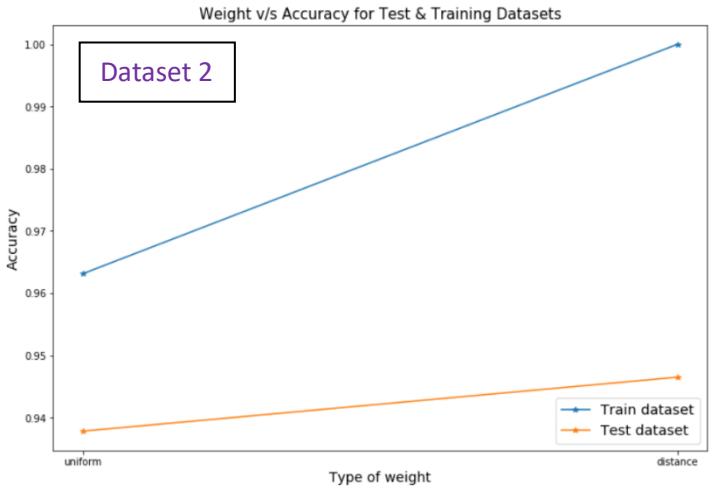
For dataset 1, the maximum test accuracy we get is **88.57%** with 5 neighbors. The train data accuracy is decreasing with increase in number of neighbors. This is because increasing neighbors decreases overfitting. The test data accuracy first increases with increase in the number of neighbors and after that decreases. This is because beyond 5 neighbors, the additional number of neighbors are introducing noise in the decision.



For dataset 2, the maximum test accuracy we get is **94.4%** with 1 neighbor. As expected, the train data accuracy is decreasing with increase in number of neighbors. Also, the test data accuracy is decreasing with increase in number of neighbors. This can be because only one neighbor is enough to correctly classify a record and any additional neighbor is adding noise to the decision.

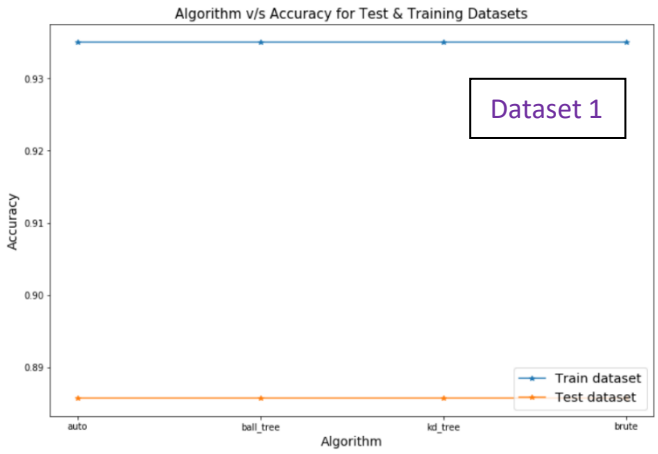
Experimentation with Weight

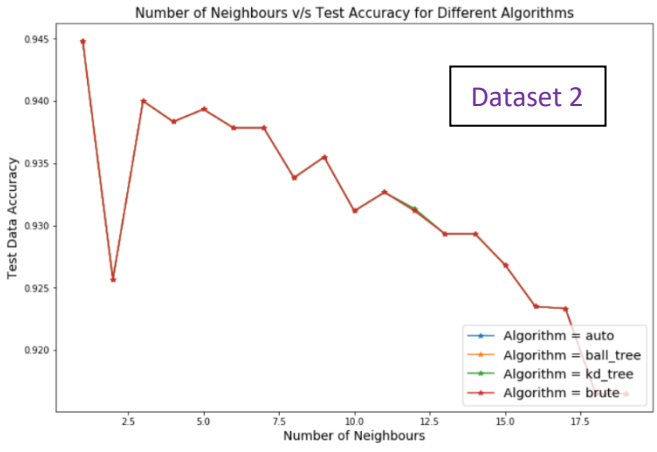




For both the datasets, we observe that the distance parameter for weight is performing better for both training and test datasets.

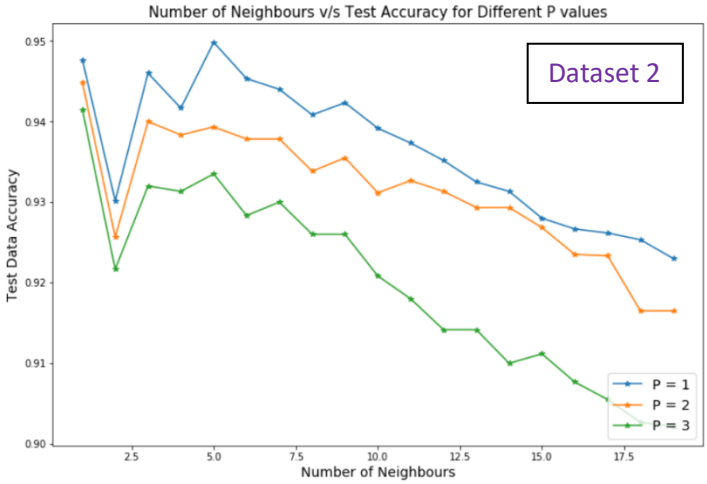
Experimentation with Algorithm





For both the datasets, we observe that the accuracy is not getting affected at all by the algorithm parameter for both training and test datasets.

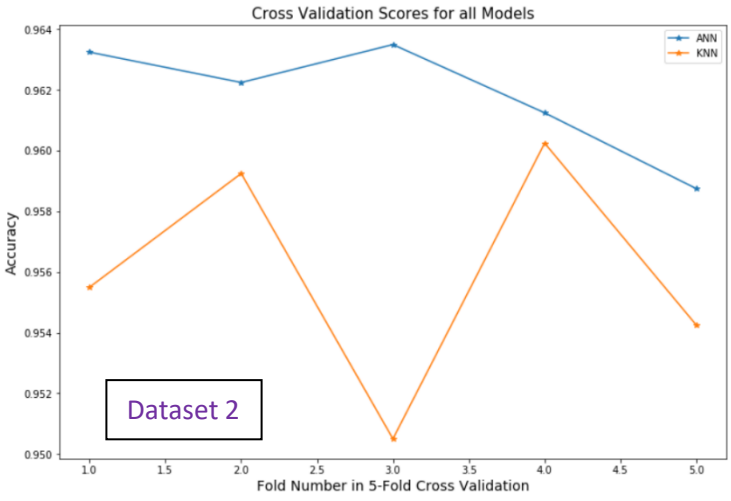
Experimentation with P



The best performing P value for both the datasets is P = 1. For dataset 2 P=1 is performing better for any number of neighbours.

Cross Validation

Cross validation is a model evaluation method. We randomly split our entire dataset into k folds. For each k-fold in the dataset, we will build our model on k-1 folds of the dataset. Then, test the model to check the effectiveness for kth fold. The models are not overfitting and performing good if the accuracy scores don't vary by a huge amount for a single algorithm.



For dataset 1, the ANN model is inconsistent and is prone to overfitting. The KNN model is better at consistency in the accuracies. However, the accuracies given by the ANN model is better than KNN model.

For dataset 2, the ANN and KNN both models are performing good because the accuracies are not varying so much. The accuracies are between 95-96.4%. Also, the accuracies given by ANN model is better than KNN.

Conclusion

Below we have compared the accuracies obtained by algorithms run in this assignment and in the previous assignment.

Dataset 1

Algorithm	Test Data Accuracy %age	Parameters
SVM (Linear)	83.07	C = 1
SVM (Rbf)	96.4	C =1, Gamma = 0.1
SVM (Polynomial)	94.2	C = 1, Degree = 4
Decision Tree	99.1	Fully Grown
Random Forest	98.8	Estimators = 500
Boosting	99.1	Estimators = 100
ANN	98.78	hidden_layer_sizes=(40,40,40), solver='adam', alpha=0.00001, activation='relu', iter=200
KNN	96.15	n_neighbors=5, weights='distance', p=1

Dataset 2

Algorithm	Test Data Accuracy %age	Parameters
SVM (Linear)	84.7	C = 1
SVM (Rbf)	97.4	C = 10, Gamma = 0.1
SVM (Polynomial)	95.1	C = 10, Degree = 3
Decision Tree	86.47	Max Depth = 24
Random Forest	96.3	Max Depth = 24, Estimators = 300
Boosting	97.52	Max Depth = 18, Estimators = 300
ANN	91.68	hidden_layer_sizes=(100,100,100), solver='adam', alpha=0.0001, activation='relu', iter= 300
KNN	95.65	n_neighbors=5, weights='distance', p=1

For dataset 1, maximum accuracy is given by Boosting. For dataset 2, maximum accuracy is given by Boosting. ANN and KNN are also performing good and are giving results comparable to Boosting and other algorithms.