BUAN 6341
APPLIED MACHINE LEARNING
ASSIGNMENT 3
Neural Networks and K-Nearest Neighbors Implementation

Report Summary

- Prepared Two Datasets (Sgemm Product Dataset and Bank Dataset) for classification
- Implemented Neural Networks and K-Nearest Neighbors Classification algorithms to calculate different performance metrics on the datasets
- Implemented Neural Networks Classification with different number of layers and nodes, different activation functions, loss functions and optimizers
- Implemented KNN Classification with different number of neighbors and distance metrics

Data Description

- Dataset 1 is the dataset of factors affecting the GPU Kernel Runtime. This dataset was also used in the assignment 1 for Linear and Logistic Regression. The data has 241600 rows and 18 columns out of which last 4 columns (dependent variable columns) have been combined into one for classification purposes.
- Dataset 2 is a bank client dataset and the dependent variable is y (Has the client subscribed a term deposit? Y/N). This data has 20 features and 1 dependent binary variable. The features include data regarding to clients which help find out if the client will subscribe to a deposit or not.

Data Preparation

Dataset 1

- Took average of 4 RunTime columns to get 1 single dependent variable Run_Avg
- In dataset 1 the values of the dependent variable (Run_Avg) is continuous which could not be used for classification
- Divided the column values into 1 and 0 by changing all the values equal and above the median value to 1 and other values to 0
- Normalized the independent variables in the dataset to get better classification results

Dataset 2

- Converted Text columns including the dependent variable to numerical values
- Applied filters to convert unknown values to known values
- Dependent variable y contained ‘Yes’ and ‘No’ values so converted them to 1 and 0
- Normalized the independent variables in the dataset to get better classification results

Neural Networks Classification

- Implemented 3 models of Neural Networks for each of the 2 datasets with different combinations of layers, nodes, activation functions, loss functions and optimizers
- As we are classifying into 0-1 classes I have used Activation function = sigmoid for the last layer in all the models.

Dataset 1

Model 1

Activation Function = sigmoid, Loss Function = squared_hinge, Optimizer = sgd

No of hidden layers = 4, Nodes in Hidden layers = 25,18,11,4

```
model = Sequential()
model.add(Dense(25, input_dim=14, activation='sigmoid'))
model.add(Dense(18, activation='sigmoid'))
model.add(Dense(11, activation='sigmoid'))
model.add(Dense(4, activation='sigmoid'))
model.add(Dense(1, activation='sigmoid'))
```

```
model.compile(loss='squared_hinge', optimizer='sgd', metrics=['accuracy'])
```

Accuracy = 49.91%

Model 2

```
model = Sequential()
model.add(Dense(50, input_dim=14, activation='sigmoid'))
model.add(Dense(39, activation='tanh'))
model.add(Dense(28, activation='tanh'))
model.add(Dense(17, activation='tanh'))
model.add(Dense(6, activation='tanh'))
model.add(Dense(1, activation='sigmoid'))
```

```
model.compile(loss='hinge', optimizer='adam', metrics=['accuracy'])
model.fit(X_train, y_train, epochs=100, verbose=0)
```

Accuracy = 84.11%

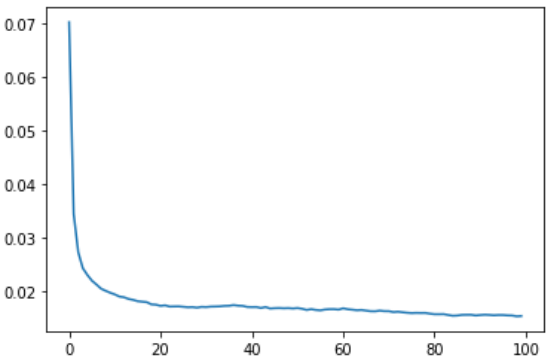
Model 3

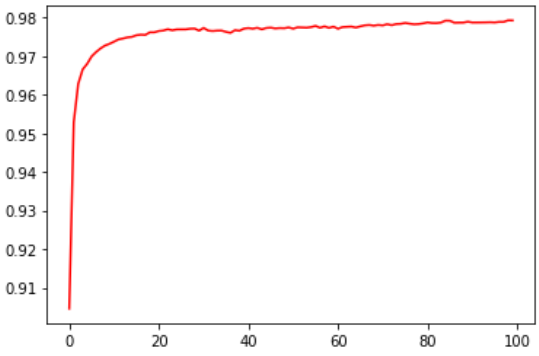
```
model = Sequential()
model.add(Dense(30, input_dim=14, activation='relu'))
model.add(Dense(15, activation='relu'))
model.add(Dense(4, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
```

```
model.compile(loss='binary_crossentropy', optimizer='RMSprop', metrics=['mean_squared_error','accuracy'])
history = model.fit(X_train, y_train, epochs=100, verbose=0)
```

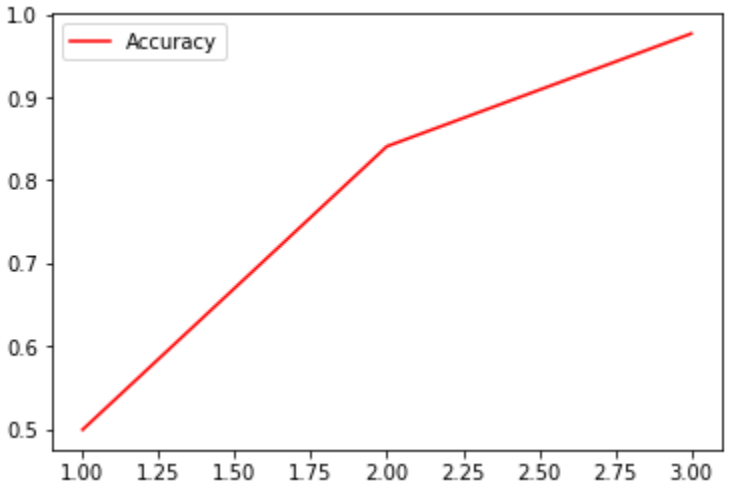
Accuracy = 97.71%

Training MSE vs Accuracy Plot for 100 epochs





Below is the Test Accuracy plot comparing the 3 models that have been described above:



- Here we can see that the accuracy for the first model is very low (50%). While that for the 3rd model is very high (98%).
- The first model is one of the failed models where the combination of the layers, nodes, activation functions, loss functions and optimizers do not fit well, and we got a very low accuracy.
- While in the third model the combination fit well, and we got high accuracy.
- From this example we can see how important all these factors are in building a good model.

Dataset 2

Model 1

```
model = Sequential()
model.add(Dense(40, input_dim=20, activation='relu'))
model.add(Dense(28, activation='relu'))
model.add(Dense(16, activation='relu'))
model.add(Dense(4, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
```

```
model.compile(loss='mean_squared_error', optimizer='sgd', metrics=['accuracy'])
model.fit(X_train, y_train, epochs=100, verbose=0)
```

Accuracy = 91.30%

Model 2

```
model = Sequential()
model.add(Dense(50, input_dim=20, activation='sigmoid'))
model.add(Dense(39, activation='sigmoid'))
model.add(Dense(28, activation='sigmoid'))
model.add(Dense(17, activation='sigmoid'))
model.add(Dense(6, activation='sigmoid'))
model.add(Dense(1, activation='sigmoid'))
```

```
model.compile(loss='squared_hinge', optimizer='RMSprop', metrics=['accuracy'])
model.fit(X_train, y_train, epochs=100, verbose=0)
```

Accuracy = 89.16

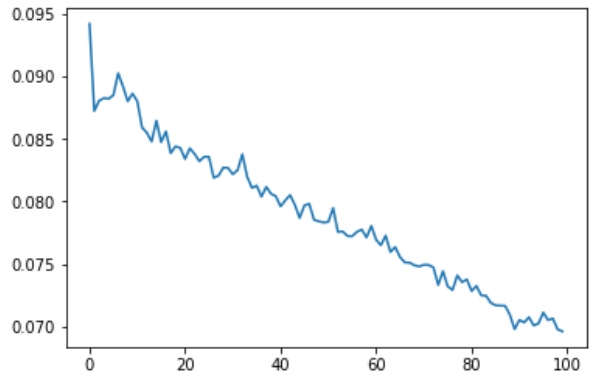
Model 3

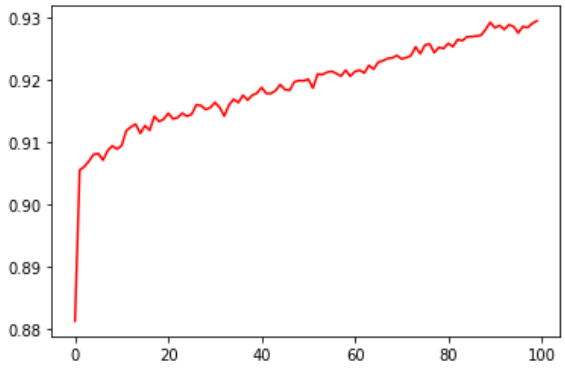
```
model = Sequential()
model.add(Dense(30, input_dim=20, activation='tanh'))
model.add(Dense(15, activation='tanh'))
model.add(Dense(4, activation='tanh'))
model.add(Dense(1, activation='sigmoid'))
```

```
model.compile(loss='hinge', optimizer='adam', metrics=['mean_squared_error','accuracy'])
history = model.fit(X_train, y_train, epochs=100, verbose=0)
```

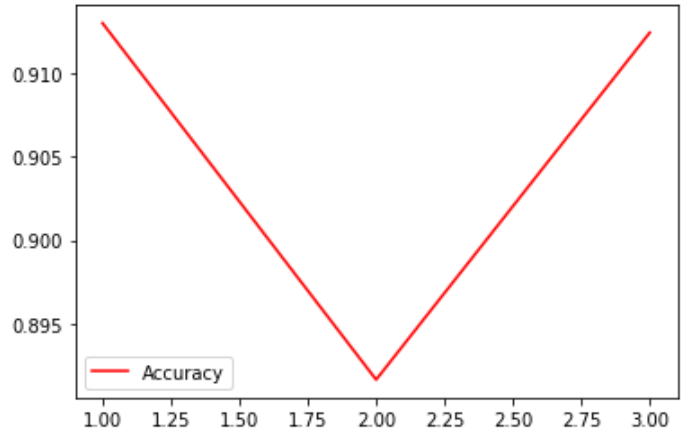
Accuracy = 91.24%

Training MSE vs Accuracy Plot for 100 epochs





Below is the Test Accuracy plot comparing the 3 models that have been described above:



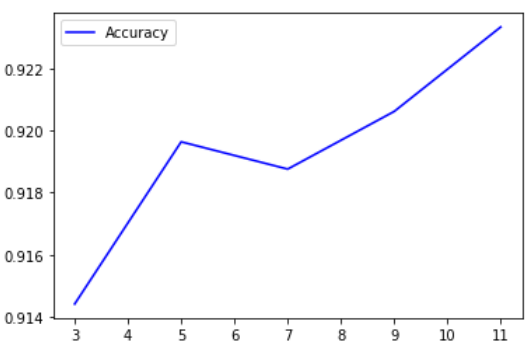
- Model 1 and model 3 perform very well on this data with 91.30% and 91.24% accuracy
- The model does not perform well with the Activation fn. ‘sigmoid’ for all the layers

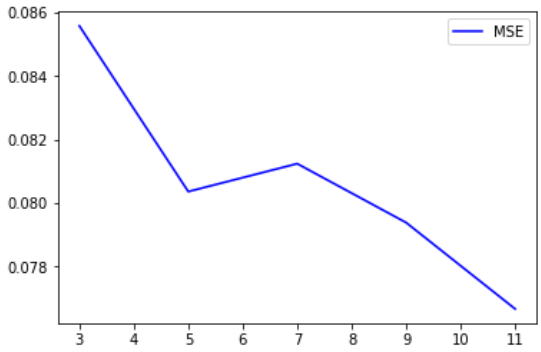
K Nearest Neighbors Classification

- Implemented KNN for different Number of Neighbors and different distance metrics
- Calculated Accuracy and MSE for above mentioned combinations

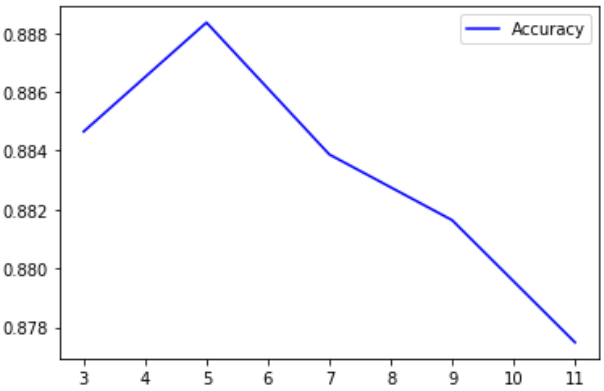
Dataset 1

Distance Metric: Manhattan (Accuracy and MSE vs Number of Neighbors)

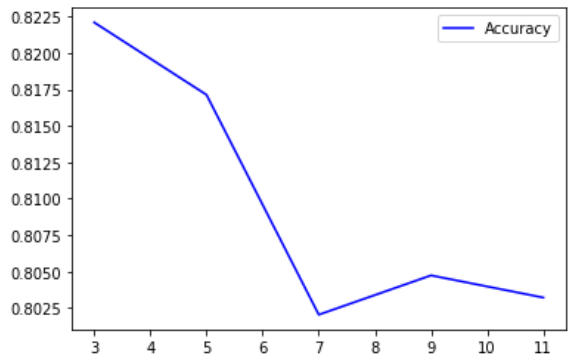




Distance Metric: Euclidean (Accuracy vs Number of Neighbors)

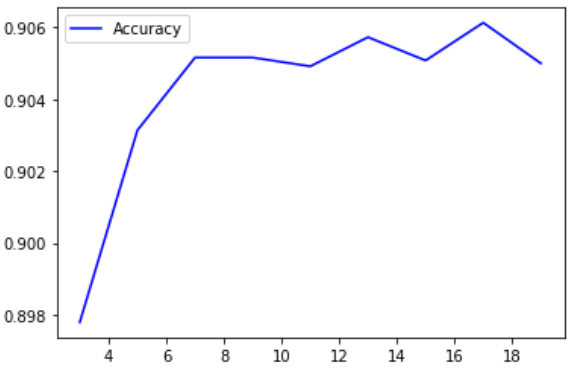


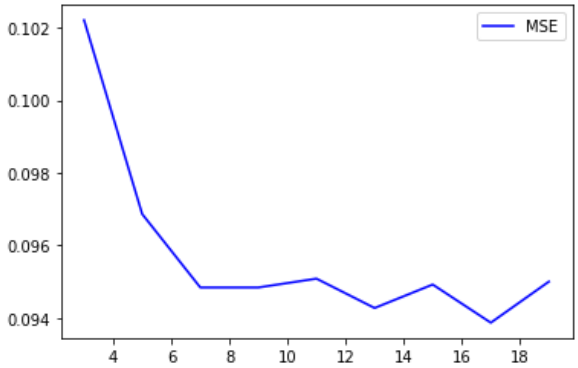
Distance Metric: Chebyshev (Accuracy vs Number of Neighbors)



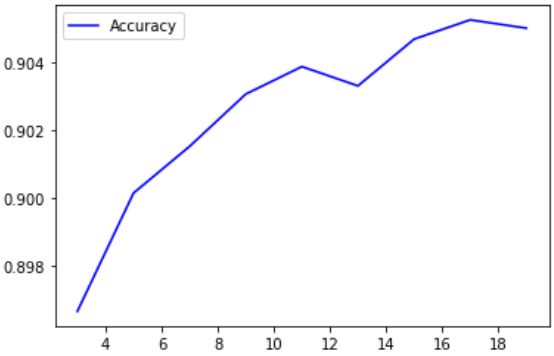
Dataset 2

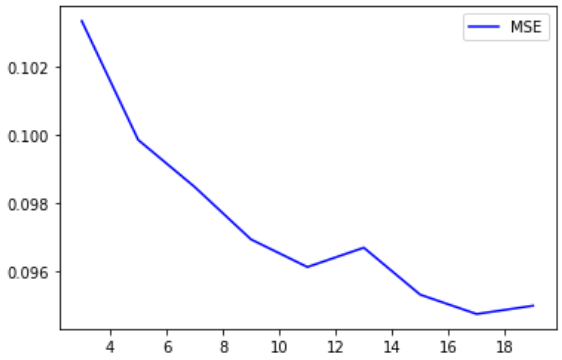
Distance Metric: Manhattan (Accuracy and MSE vs Number of Neighbors)



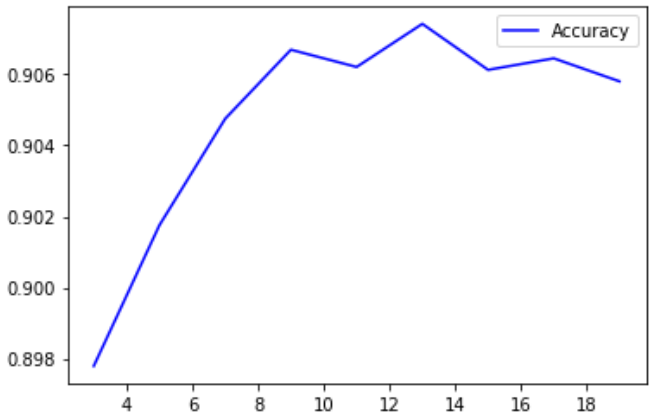


Distance Metric: Chebyshev (Accuracy and MSE vs Number of Neighbors)





Distance Metric: Euclidean (Accuracy vs Number of Neighbors)



Inferences from these Datasets Classification

- Neural Network performed well with Activation fn. ‘relu’ on both datasets
- Neural Network did not perform well with Loss fn. ‘squared_hinge’
- KNN performed well with 7,9 neighbors and Distance Metric ‘Manhattan’ for both these datasets
- KNN did not perform well with Distance Metric ‘chebyshev’ on dataset 1

Comparison of the two learning algorithms

- In the two algorithms implemented above Neural Network Classifier performs much better than the K Nearest Neighbor Classifier
- On the first dataset K Nearest Neighbors gives around 90% accuracy as compared to the accuracy given by Neural Networks which is 95%-97%
- On the second dataset the accuracy given by both the algorithms is similar KNN gives 90% while Neural Networks gives 91%

Performance Comparison (All 5 Algorithms)

- For the First Dataset Neural Networks performs best followed by Boosting while SVM, KNN and Decision Trees give similar accuracy.
- For the Second Dataset all the algorithms gave accuracy between 88% and 92%. Neural Networks sometimes gave 92% accuracy while SVM and Boosting gave 91% accuracy
- I would rank the 5 algorithms in this order:
    1. Neural Networks – Very Efficient
    2. Boosting – Fast and Efficient
    3. KNN - Moderate
    4. Decision Tree -Moderate
    5. SVM – Effective but Slow