Dataset 1: Appliances energy prediction

Classification Problem: Predict the energy usage (High or Low) of the appliances from the varying house temperature and humidity conditions that were monitor by a wireless sensor network. Other attributes like the weather from the nearest airport station are also included in this dataset.

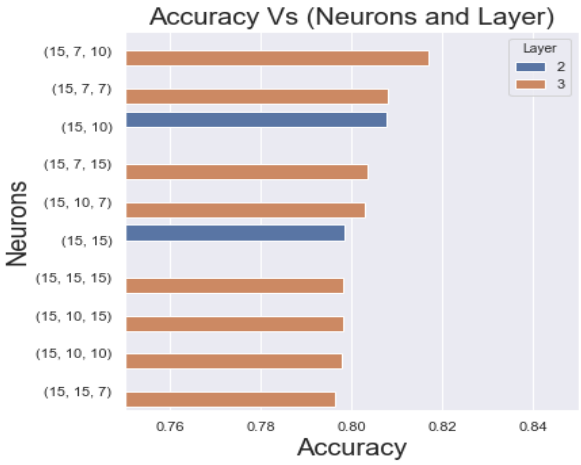
Exploratory Data Analysis:

- Class 0 (Low): 10357 observations; Class 1 (High): 8048 observations
- Dropped features 'T3','RH_4','T5','T8','RH_7','T7','RH_8','T6','rv2','Visibility' from the dataset due to its high multicollinearity with the other feature variables.
- Dropped lights feature as it had almost 80% (15252 observations) of its value to be 0.
- Dropped features date, rv1, rv2 as they are just random and do no have any relation to the target variable
- As the data points are in different scales, feature scaling using StandardScalar is performed before the implementaion of both Artificial Neural Networks and K-Nearest Neighbors algorithms.

Artificial Neural Networks

Finding the best combinations of layers and neurons

Parameters: Epoch = 100, Batch_size = 200, optimizer = Adam, learning_rate=0.005, Activation: Sigmoid

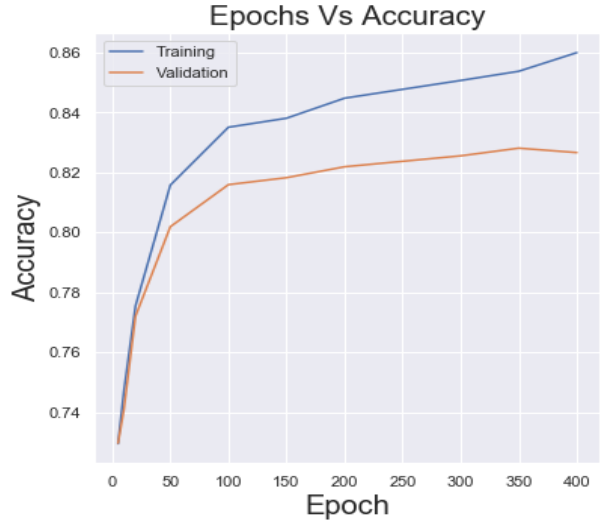


Inputs: Layers = [2,3], Neurons = [7, 10, 15]

For each combination of neurons and layers, neural network model was trained and evaluated on the test data. Plot shows the combinations that had the top 10 accuracies on the testing data. Highest Accuracies where achieved when the first layer had atleat 15 number of neurons (i.e Atleast the number of features) in the first layer. These results give us an estimate of the number of neurons and the layers that can used in the future experiments. We can choose **3 Layers with (15, 7 10) neurons in each of the 3 layers**. Cross Validation is not performed here as it is computational expensive considering there would be 36 neural network algorithms to be run for all the combinations

Hyperparameter Tuning to find the optimal number of epochs

Parameters: Batch_size=200, optimizer=Adam, learning_rate=0.01, Activation: Sigmoid, Layers = 3, Neurons = (15,7,10)



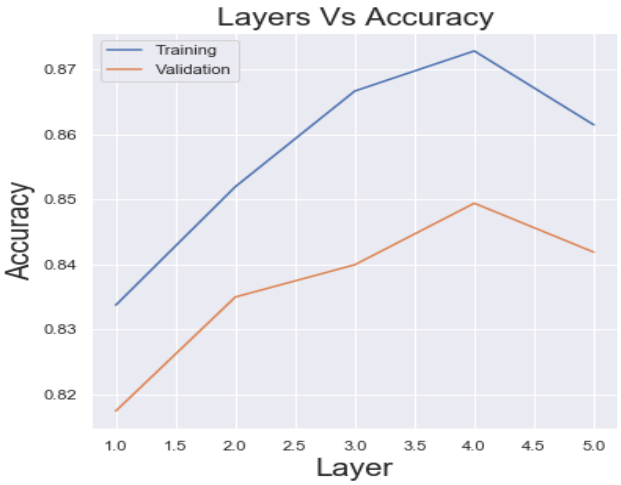
Epoch: [5,10,20,50,100,150,200,300,350,400]

Model implementation with cross validation to find the optimal parameter

With increasing number of epochs, the training and validation scores have a gradual increasing trend. This is because, with increasing epochs the model tries to optimise and reach the point with the minimum loss. There is only a slight increase in the validation accuracy after 100 epochs. **The model seems to have optimised when the number of epochs reached 300**. Although, training score is increasing even after 300 epochs, it is case of overfitting here as the validation scores have reached its saturation at 300 epochs.

Hyperparameter Tuning to find the optimal number of layers

Parameters: Batch_size=200, optimizer=Adam, learning_rate=0.01, Activation: Sigmoid



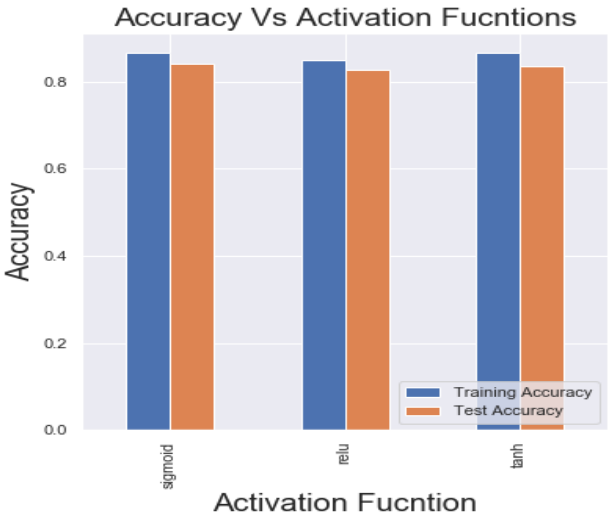
Layer: [1,2,3,4,5]

Model implementation with cross validation to find the optimal parameter

With increasing number of layers, the training and validation accuracy scores have a gradual increasing trend. The model has optimised when the number of layers reached 4. **Both the training and validation accuracies are decreasing after 4 layers**. Cross Validation is helping us to ensure the model is generalising the data well and not having any overfitting issues.

Hyperparameter Tuning to find the optimal number of layers

Parameters: Epoch=300, Batch_size=200, optimizer=Adam, learning_rate=0.01, Layer: 3, Neurons = (15,7,10)



Activation Functions: ['sigmoid','relu', tanh]

Model implementation with cross validation to find the optimal parameter

The model is approximately performing the **same** with any of the any activation functions. For this dataset, the model is independent on using the activation function.
Note: The activation function of the output layer is always kept as Sigmoid, as this a binary classification problem and we want the output to range from 0 to 1.

Using the Optimal parameters to find the Best ANN Model

Optimal Parameters: Epochs: 300, Activation Function: Sigmoid, Layers: 4, Neurons: (15,7,10,10), Batch Size: 200, Learning Rate: 0.01, Loss: Binary_crossentropy, Optimizer: Adam

Confusion Matrix

	Predicted 0	Predicted 1
Actual 0	2592	461
Actual 1	429	2040

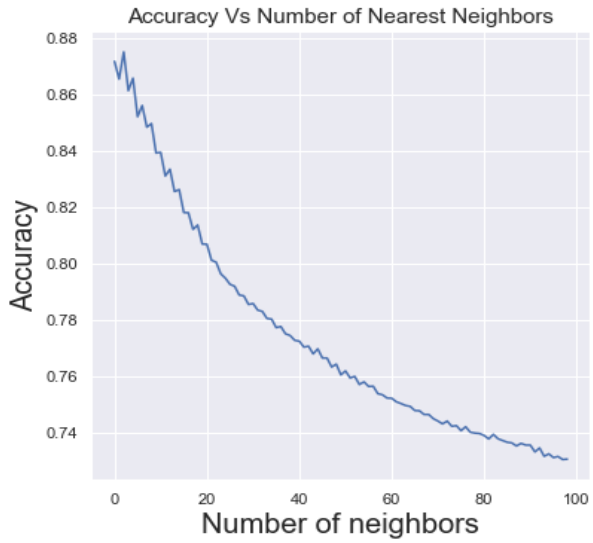
Classification Report

	precision	recall	f1-score	support
0	0.86	0.85	0.85	3053
1	0.82	0.83	0.82	2469
avg / total	0.84	0.84	0.84	5522

Train Accuracy: 0.872
Test Accuracy: 0.838

K-Nearest Neighbors

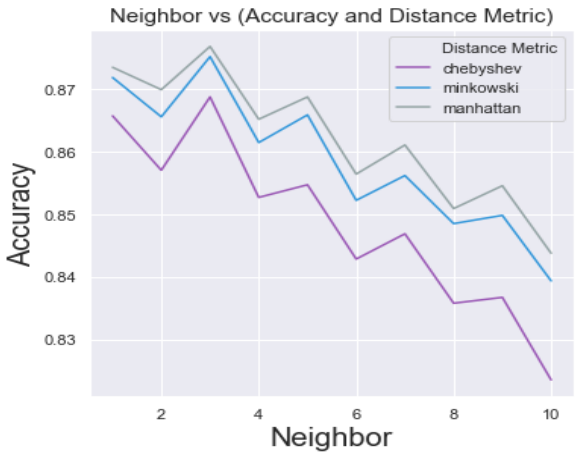
Hyperparameter Tuning to find the optimal number of neighbors



Distance Metric: Minkowski, Weight: Uniform
Model implementation with cross validation to find the optimal parameter

Validation accuracy implemented with cross validation is continously decreasing with increase in the number of nearest neighbors. **Optimal number of neighbors can be chosen as 3**. As the number of neighbors is increasing, the model is trying to fit the noise more than generalising the data. For example, when the number of neighbors is equal to the number of observations in the dataset, the training accuracy will be one, but that is a case of overfitting as the validation accuracy will be very poor.

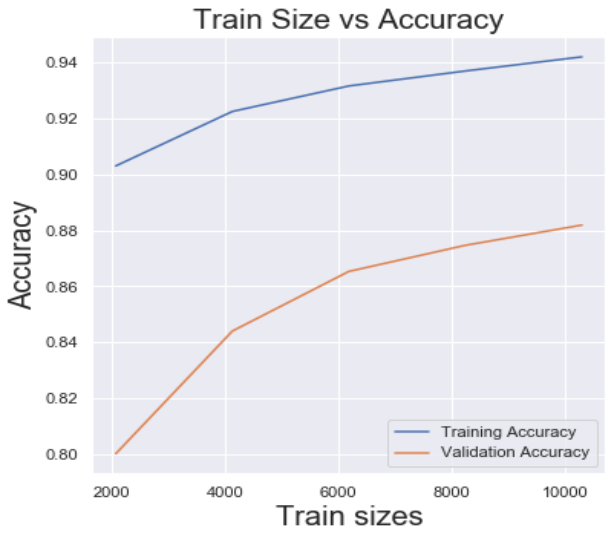
Hyperparameter Tuning to find the optimal Distance Metric



Neighbors: 1 to 10, Weight: Uniform
Distance Metrics: ['chebyshev','minkowski','manhattan']
Model implementation with cross validation to find the optimal parameter

When the number of neighbors is 3, the validation accuracy has reached its maximum for all the 3-distance metrics. **Distance Metric Manhattan is performing the best for this dataset**. Manhattan distance is the sum of the absolute difference between the data points. The default distance metric “minkowski” is also performing well with its validation accuracies very close to manhattan.

Learning Curves: Train Size Vs Accuracy



Neighbors: 3, Weight: Uniform
Distance Metrics: 'manhattan'
Model implementation with cross validation to find the performance with increasing training sizes

The model is continously learning and generalising the data well with increasing training sizes as we have both training and validation accuracies with a gradual upward trend. The model has almost reached its saturation with **8000 observations** as there is only a slight change in accuracies with 10000 observations.

Using the Optimal parameters to find the Best KNN Model

Optimal Parameters: Neighbors: 3, Distance Metric: Manhattan, Weight: Uniform

Confusion Matrix

	Predicted 0	Predicted 1
Actual 0	2793	260
Actual 1	355	2114

Classification Report

	precision	recall	f1-score	support
0	0.89	0.91	0.90	3053
1	0.89	0.86	0.87	2469
avg / total	0.89	0.89	0.89	5522

Train Accuracy: 0.945
Test Accuracy: 0.888

Comparison of the model performances of all the implemented algorithms

Note: Data Cleaning and the metrics used to evaluate is the same across all the algorithms

Ranking of the Test Accuracies for all the Algorithms

Classifier	Test Accuracy
K-Nearest Neighbor	0.88
Boosted Decision Tree	0.86
Radial SVM	0.84
Artificial Neural Network	0.83
Decision Tree	0.80

Is ANN and KNN performing better than the other 3 algorithms?

Based on the test accuracy, KNN algorithm is performing better than all the other algorithms **(Test Accuracy = 0.88)**. ANN is also performing well with test accuracy of **0.83**, which is close to Radial SVM's algorithm.

Observations of all the 5 algorithms

- Overall, all the algorithms are performing well on this dataset as test accuracy of atleast 80% is being achieved.
- Among the algorithms implemented, K-Nearest Neighbor is performing the best with its test accuracy reaching **0.88** when 3 neighbors were used as its parameter. Although KNN, being an unsupervised form of algorithm did not learn anything from the data points. It just fitted the data points and had all the observations plotted in the memory. This is the reason why the implementation of KNN algorithm algorithm on the training dataset. On the contrary, the predictions of classes on the test data is slower compared to the other algorithms, as all the other algorithms had parameters created based on its learning from the training data. KNN does not perform well when the number of features in the dataset is high, which is quite common in a real world scnerio.
- The implementation of the ANN algorithm is quite computational expensive as it requires more time and more memory. Hyperparameter tuning of the ANN algorithm is quite complicated as both number of the layers and number of neurons need to be considered while building its network architechture. But once the right combination of parameters is found, it may perform well and give us good metrics. On this dataset, ANN was able to achieve a test accuracy of **0.83**. ANN may perform better if Boosting is applied to it.
- Although Radial SVM had a test accuracy of 0.84, it was generalising the data better than the Boosted DecisionTree as the cross-validation scores of Radial SVM were greater than the Boosted DecisionTree.
- Finally, the results and algorithm performances are just based on this dataset and varies with other datasets. We cannot conclude on the best algorithm as algorithms performance totally depends on the dataset.