Havish Chennamraj
903201642

CS 7641 – Assignment 1 Report

Datasets

[1] Stay Alert! dataset from the The Ford Challenge

The dataset consists of results of a number of "trials", each one representing about 2 minutes of sequential data that are recorded every 100 ms during a driving session on the road or in a driving simulator. The trials are samples from some 100 drivers of both genders, and of different ages and ethnic backgrounds. Every sample in the dataset has 30 features which are continuous values representing physiological, environmental and vehicular data. It's also got a binary variable "IsAlert" around which the classification problem is built. The training dataset has around 600k records and the test dataset has around 121k records to work with.

[2] Otto dataset from Otto Group Product Classification Challenge

The dataset is a collection of transactional data of the Otto Group, one of the world's biggest e-commerce companies. It has around 93 features for about 200,000 products. Each row in the dataset corresponds to a single product and the 93 features which represent count of different events. The feature set is a mixture of binary and continuous valued variables. Additionally, the dataset has a "target" feature which is of particular interest to our classification problem, and has 9 different values to it each corresponding to a particular product category in the company like fashion, electronics etc.

Links to datasets
Stay Alert: https://www.kaggle.com/c/stayalert/data
Otto: https://www.kaggle.com/c/otto-group-product-classification-challenge/data

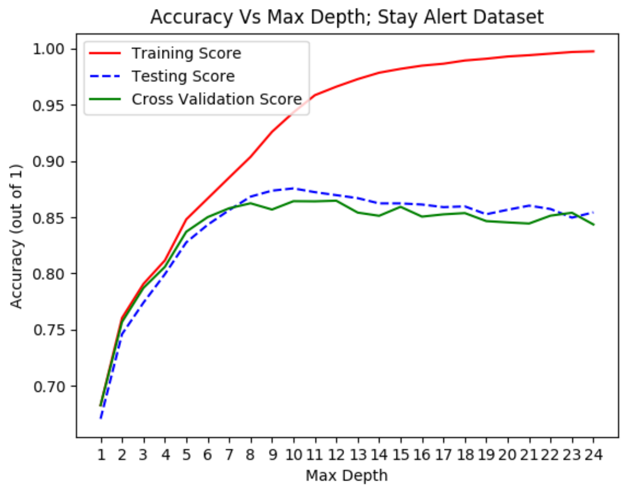
Classification Problem & Motivation

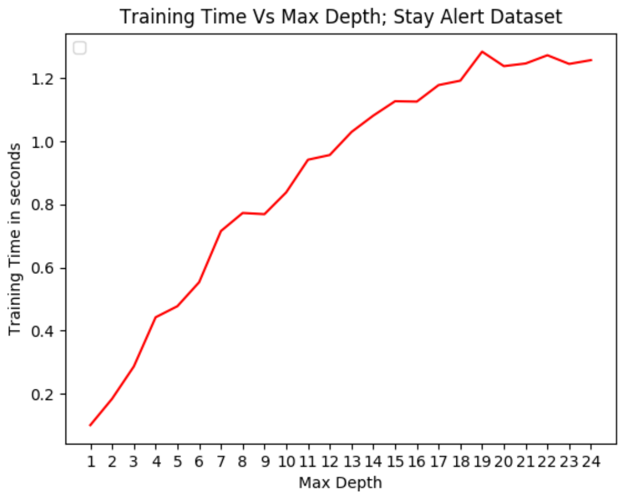
1.) In the first experiment we seek to solve a binary classification problem on the 'Stay Alert!' dataset. The objective of the classification problem is to basically predict whether a driver is alert or not based on a set of environmental, physiological and vehicular observations. It's interesting to solve this problem because it's a very practical application of Machine Learning and not very hard to realize it in practice. Also, there is an element of importance to the problem because driving while distracted, fatigued or drowsy may lead to accidents, and potentially loss of life which makes it all the more of a motivation to get it right. It is important to note that in the interest of time, the dataset has been sub-sampled.
2.) In the second experiment we seek to solve a 9-way classification problem on the Otto products dataset where the objective is to predict the product category of the item based on 93 features (details of which have not been provided). The second problem is interesting because it provides us with to scope to practice classification outside the usual binary space. Also, the practical nature of the problem excites me because of the real time applications of well-built classifier in this problem. Due to a diverse global infrastructure and too many moving parts, its usually difficult to classify products right. Proper classification of products allows the company to deliver consistent analysis of the performance & quality of the products, and also provide better insights about the product range.

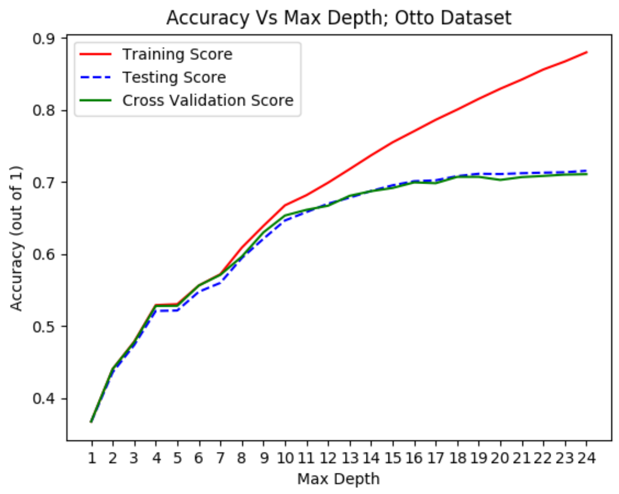
Both problems involve very large datasets and in the interest of time they've been heavily subsampled for the experiments. Also, every experiment on any classifier involves first finding the best hyperparameters (which is done by consulting the validation score obtained by performing 5-fold cross validation on the entire training set) and then measuring the performance of the model against increasing training sample sizes on the entire test set.

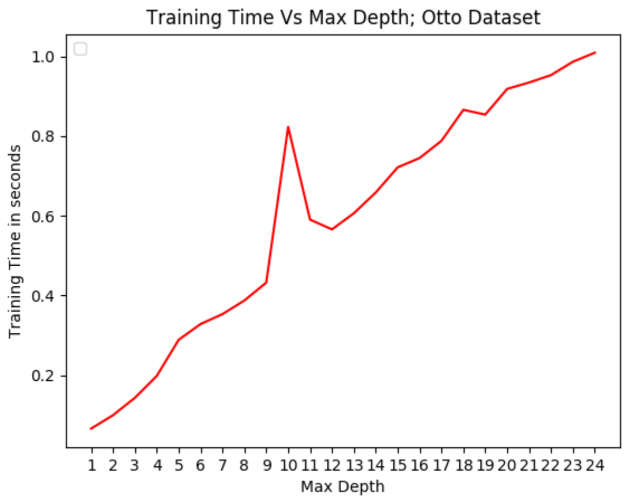
Decision Trees

For the decision tree, we used the *Gini impurity*measure to assess the quality of any potential split, i.e. the feature and threshold which gives the best *Gini impurity*score is selected for the split. Scikit doesn't provide an option(?) for pruning decision trees, therefore I've used max_depth parameter of the decision tree to obtain "pruning-like" procedure. Restricting the depth of the tree ensures that the tree doesn't grow too large and possibly overfit on less useful features.



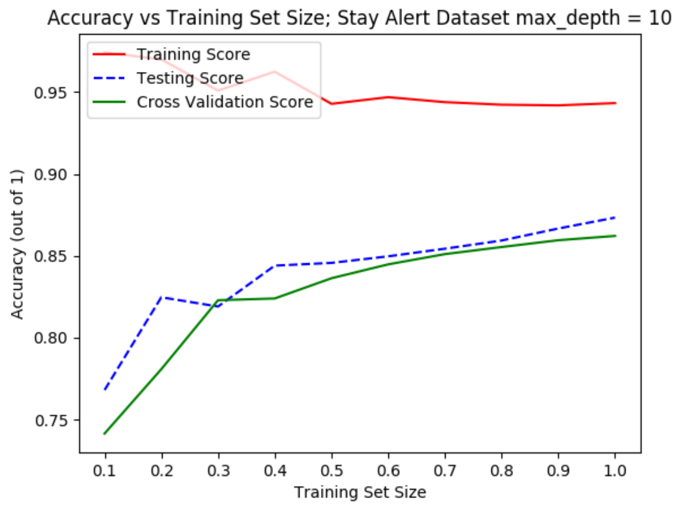


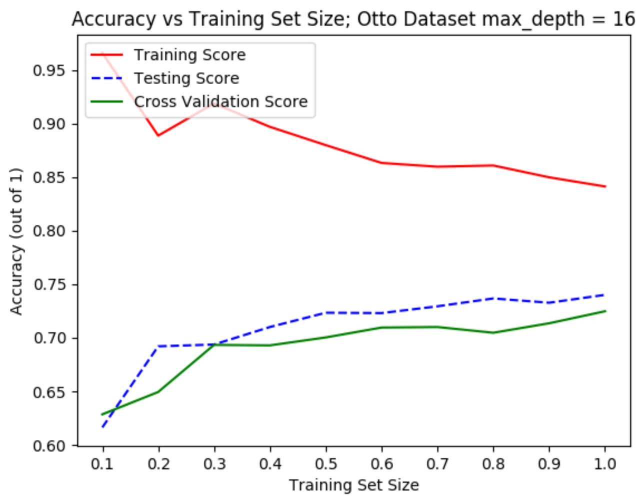




As expected, time taken to train the decision tree in both the experiments increases with max_depth considering we're allowing the tree to grow larger. I've performed the experiment multiple times and I've noticed a random spike at an intermediate depth sometimes, and I'd possibly attribute that interference from the environment like heavy CPU utilization, but the general trend is upwards. However, it is interesting to note that the model 'minorly' over-fits after a *max_depth=10 in* experiment 1 and *max_depth=16* in experiment 2. The way we can infer this is by noticing the trend of Training Vs Cross Validation & Testing Scores. While the training score continues to increase with max_depth, testing and cross validation scores tend to slightly deteriorate/stablize after the above specified max_depth values in the respective graphs. Thus based on this, I've decided to go with

10 & 16 as my maximum permissible depth while training my decision tree for the next segment of the experiment 1 & 2 respectively.





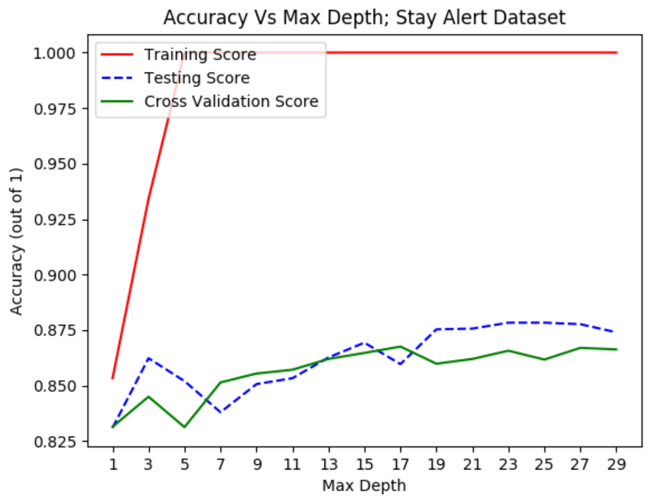
In the above section of the experiment, we compute the training, validation and test accuracy of the decision tree with max_depth = 10 & 16 for experiments 1 & 2 respectively by increasing the training sample size from 0.1 times the Max Training set size to the complete training data. Obviously the higher the score is, the better the model is performing. As you can see the 'trend' is very similar in both the graphs

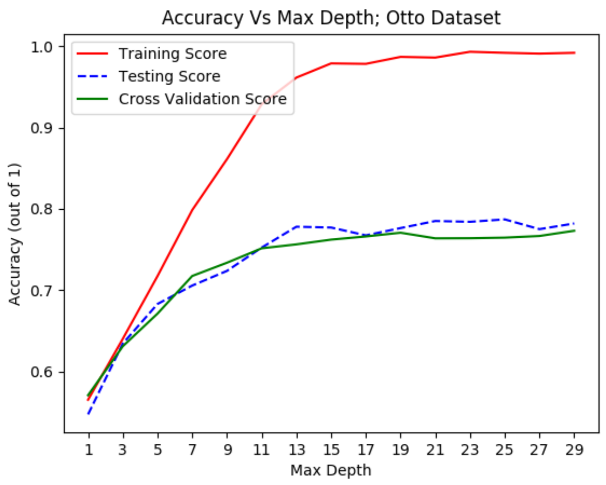
A larger training set decreases the score because it is more difficult for the learning algorithm to learn a model that correctly represents all the training data. However, as we increase the size of the training set, the test score also increases, due to an increase in the model's ability to generalize. As you can notice, all the three scores tend to stabilize as the sample size increases. This means that in order to learn properly, an algorithm requires enough data, just enough for us to get to the right side of this plot. Once we reach that asymptote, we cannot improve the test score by using more training data.

Also note that we've achieved an accuracy score of ~86% on Stay Alert – Test dataset which shows that it is very separable in the domain of decision tress.

Adaboosting (Boosted Decision Trees)

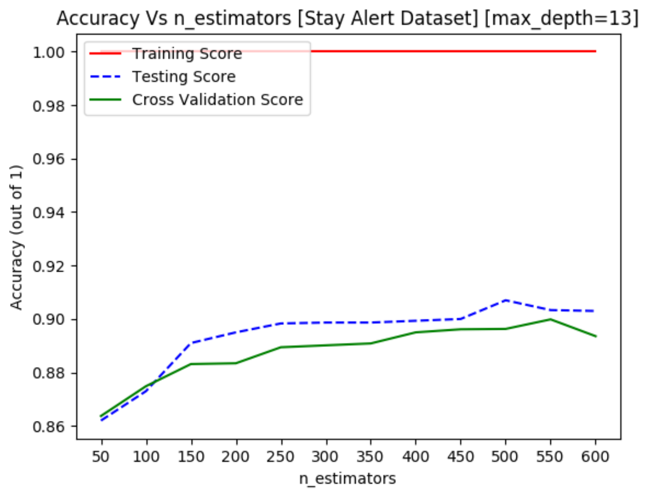
We now experiment with boosted decision trees, in particular adaptive boosting, or AdaBoost, as included in the scikit library. We ideally expect a better performance on the validation/test dataset with adaBoosted decision trees with a more aggressive pruning.

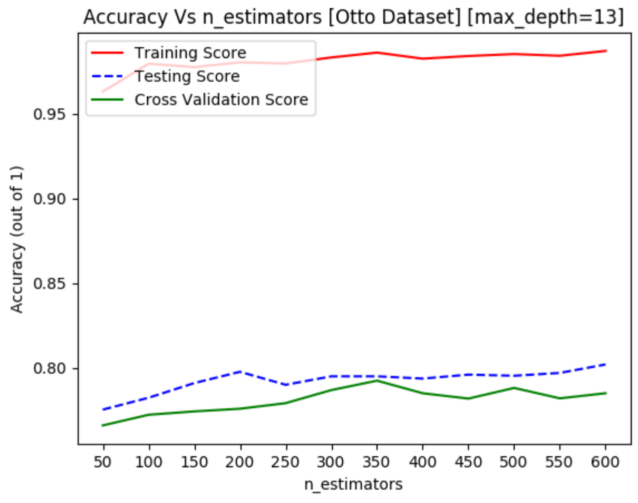




The Otto Dataset seems to adhere to our obvious assumptions achieving a better score at a lower depth. However, the noteworthy part here is our boosted model performed same/slightly worse(?) than a single decision tree at max_depth 10 on ‘Stay Alert’ dataset. The only thing I can possibly infer is that with lesser number of features, as in the case of ‘Stay Alert’ dataset (30 against 93 in Otto), boosting doesn’t provide a significant boost in accuracy over traditional decision trees.

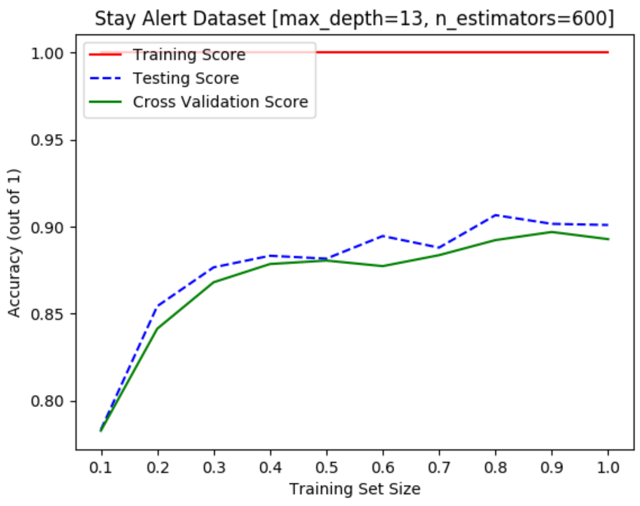
We then proceed to evaluate how hyperparameter n_estimator which basically is the maximum number of estimators at which boosting is terminated, affects our model.

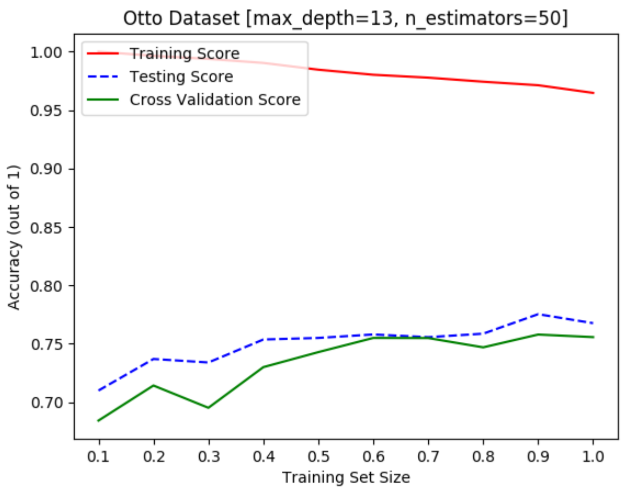




It doesn’t seem to do much in the ‘Otto’ dataset as the scores remain fairly stable with increasing value of n_estimators. We do notice a significant improvement in ‘Stay Alert’ dataset from ~0.86 from our previous experiment (with max_depth = 13 & n_estimator = 50) to ~0.9 with same max_depth but n_estimator = 600. It seems like increasing the number of weak learners made up for the lack of number of features in the ‘Stay Alert’ dataset.

Having found the best hyperparameters for our ‘AdaBoost-ed’ Decision Trees for both datasets, we proceed to evaluate how the models perform with varying training sizes.



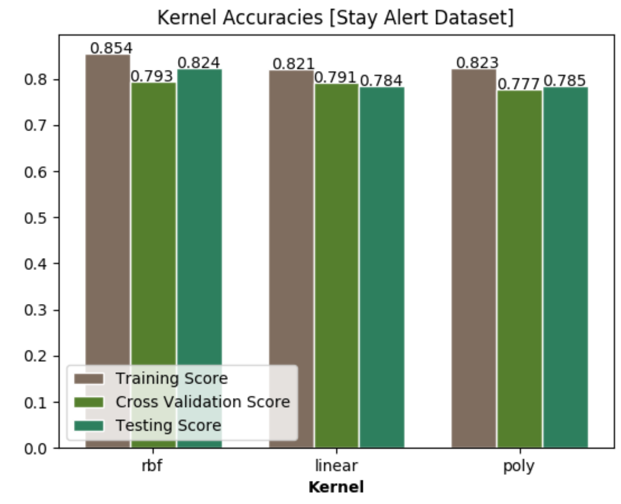


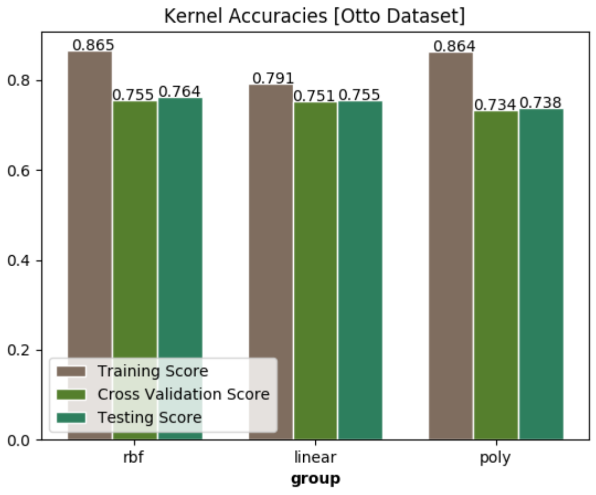
The testing and validations scores increase asymptotically in 'Stay Alert' dataset, and tend to stabilize beyond 0.3 times the training sample size at ~0.90. This shows that model learnt whatever it could with a much lesser sample size than what we fed it initially. This increased ability to generalize on the test/validation set with a much lesser training set to work with probably stems from the greater number of weak learners in the ensemble. However, it isn't a significant improvement over Individual Decision Trees in terms of max achievable accuracy which might indicate that the boosted model can be improved further thorough investigation of parameters such as grid search of confidence factors and minimum samples split.

In the case of 'Otto Dataset', the test/validation scores increase albeit gradually with increasing training size and tend to stabilize beyond 0.6x training size. This is because the model needs more data to reach its potential (~0.77 accuracy score on test/validation sets) because of fewer number of weak learners in the bag. We also notice a dip in the training score, this is because with increasing training size the model finds it difficult to correctly represent all the training samples whilst not overfitting at the same time.

SVM

Support Vector Machine (SVM), which happens to be my favorite classifier, is a discriminative classifier defined by a separating hyperplane. Given the labeled training data, SVM outputs an optimal hyperplane that categorizes new examples or simply put it creates a line or a hyperplane which separates the data into classes. To evaluate which kernel to use, I've ran SVM with three different kernels namely rbf, linear and poly on the entire training set. Please note that the experiments have been performed with the Penalty parameter C set to 1.

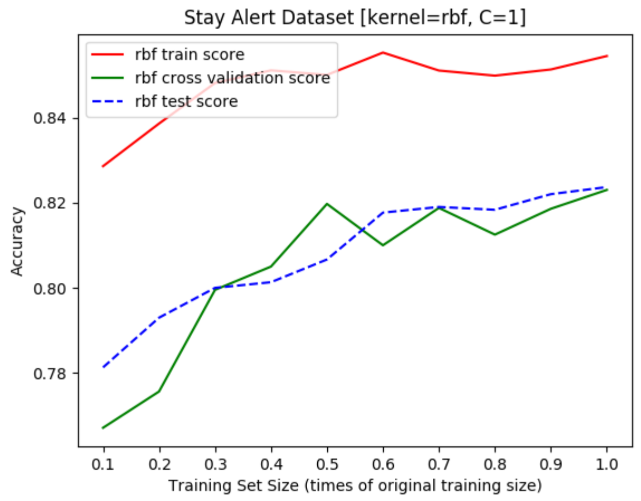


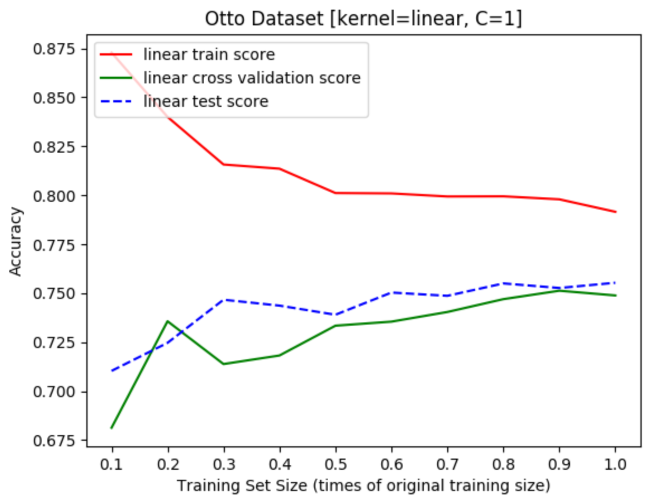


By looking at the collective validation and testing scores, rbf seems to the obvious choice in Stay Alert dataset while linear and poly are quite evenly matched in the Otto dataset. It is easy to justify why rbf performs so well even without tuning any other hyperparameters. A linear function allows us to define linear function which might be very impoverished, while a polynomial function of greater degree might be able to define a function space that correctly represent a much complex dataset. However, RBF (Gaussian) is a squared-exponential function which is able to define a much larger function space than a linear or polynomial function. Thus, RBF is generally more flexible than linear or polynomial functions in the sense than we can model a whole lot more functions with its function space.

Now if RBF is so powerful, how can something like a linear kernel match/slightly outperform it in the Otto Dataset? I believe it's a classic case of over fitting in the Otto dataset. For example, if RBF kernel is set with a low Penalty parameter 'C', the SVM will tend towards a linear classifier, and with a high value the output of the classifier will be very sensitive to small changes in the input, which means that even with margin maximization, the model with likely get over-fitted. RBF in the Otto dataset fit the training sample so well (which is proven from the training accuracy of ~87%) that it lost its ability to generalize on any test samples that deviated from the training trend.

But it is important to note that linear kernel is a degenerate version of RBF kernel [3] meaning a properly tuned RBF kernel will always out perform a linear kernel. This indicates that there is scope for tuning in Otto dataset and possibly in Stay Alert Dataset as well. Considering the long computational times involved especially in training the models, I've decided to proceed with RBF for Stay Alert dataset and Linear for Otto (because of its lesser training time) with the other parameters set to default for the final SVM experiment which is to evaluate how training sample size effects the performance of the model.

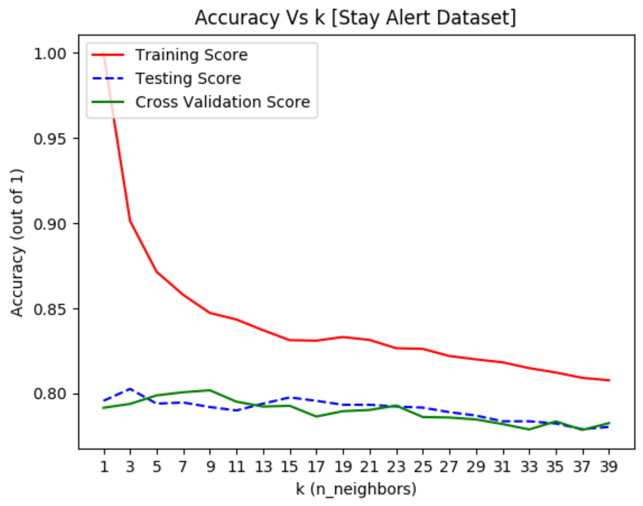


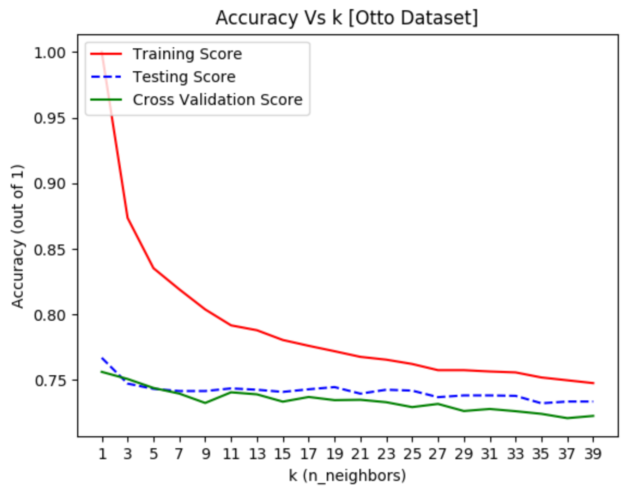


The test/validation scores increase with increasing training size in both datasets; the more the model learns, the better its ability to generalize. However, we notice that whilst the training accuracy increases in Stay Alert database which employs RBF kernel, it decreases in Otto with a linear kernel with increasing training size. Considering the robust and flexible characteristic of RBF kernel, the model is able to keep up with the increasing training size, while linear kernel being able to define only linear function space its corresponding model is struggling to represent all the samples in the training data.

KNN

K-Nearest Neighbors (kNN) is a non-parametric method used for classification. kNN is also instance-based learning where the function is only approximated locally and all computation is deferred until classification. The algorithm makes predictions for a point of given input by looking at the k nearest data points and making a decision on what to predict based off of those k nearest points. Thus, obviously finding the best value of k is of utmost important for building the best possible kNN model.



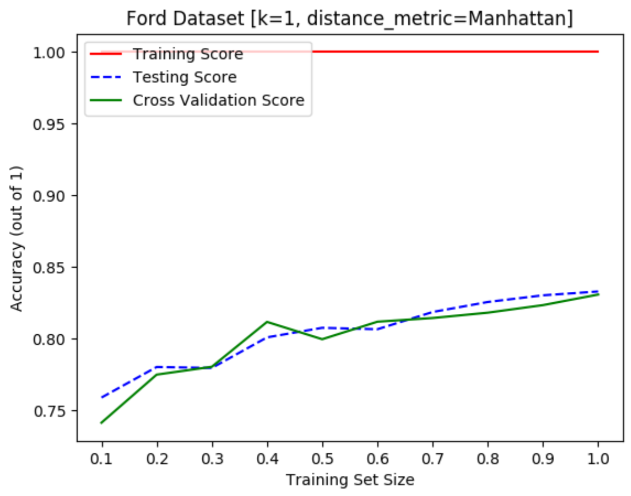


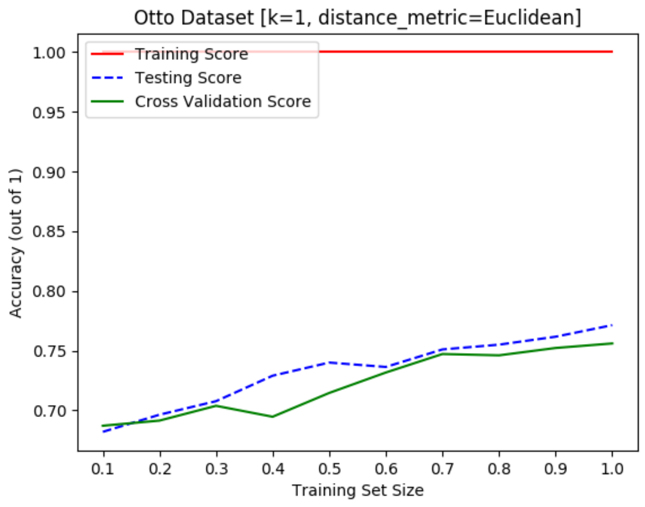
As we can notice, both the graphs have the same exact trend. As k increases, the training accuracy decreases whilst the validation/test accuracy remains constant. From this, we can possibly infer than the validation/test sets are highly skewed towards a particular class label (which is kind of surprising for the otto dataset considering it's a multi-class dataset). As for the decreasing training score, I attribute that to noise in the training sets. As ‘k’ increases, more instances are included and introduce more confusion to the classifier. Based on the above results, we proceed to conducts are further experiments on kNN with k = 1.

Distance metric	**Stay Alert Validation Score**	**Otto Validation Score**
Euclidean	0.7889979241412525	0.7581445823857655
Manhattan	0.8271429922252059	0.7580044604895145
Chebyshev	0.7319999970603841	0.7580044604895145
Hamming	0.7892893135653964	0.7139998381403637
Minkowski (p=2)	0.7813234785403490	0.7483234785403490

Also varying the distance metric which kNN employs to find the k nearest neighbors of a sample, we’ve found that Manhattan and Euclidean give the best results for ‘Stay Alert’ and ‘Otto’ datasets respectively, which make sense as both datasets have features with only continuous variable features with same class labels lying in close proximity with respect to each other.

Having found the optimal value of ‘k’ and distance function for both our datasets, we proceed to train the model with the said optimal values of k and distance function on varying training sizes for both datasets.

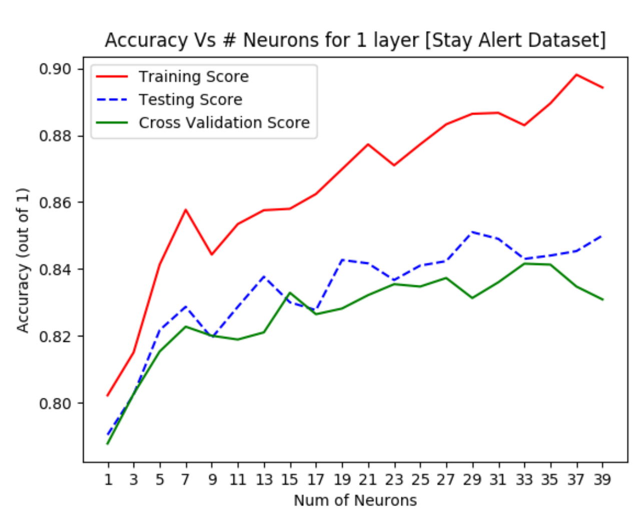


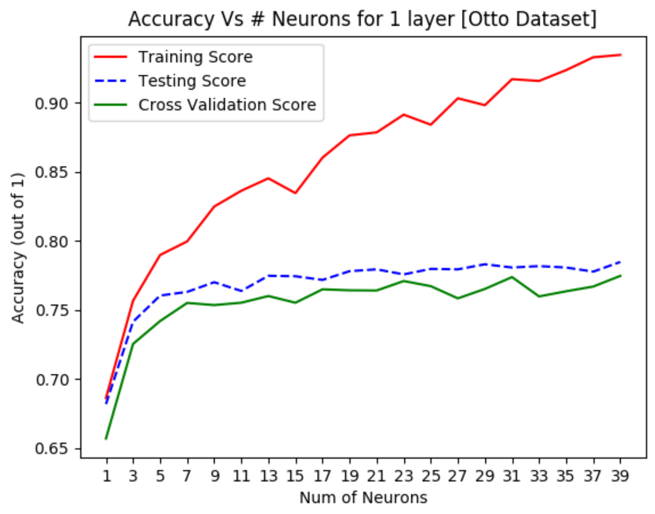


The training score remains constant while the test score increases almost linearly with increasing training size, suggesting that there is no over fitting. Also, there is a substantial increase of ~0.1 from 0.1x Training size to the complete training set which complies with my conjecture before about both the datasets being skewed. By feeding more of the ‘skewed’ data to the model, more samples which have been classified otherwise have been tagged with the majority class label.

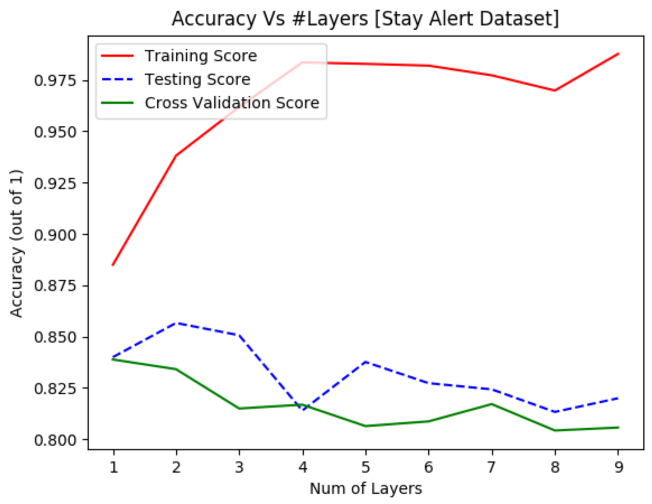
<u>Neural Networks</u>

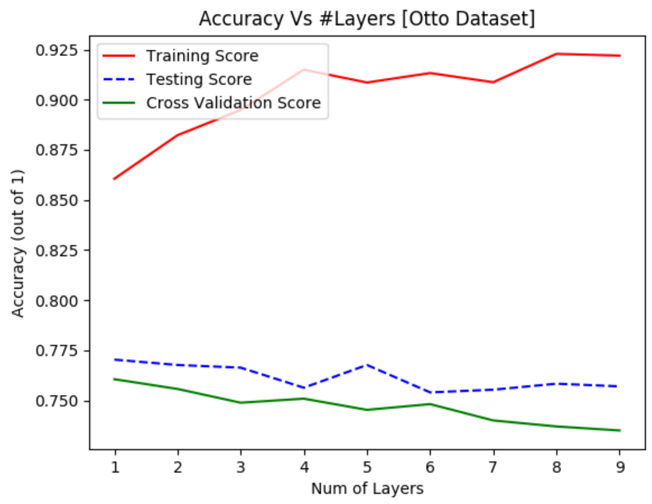
For Neural networks I’ve chosen Multi Layer Perceptron Classifier (MLPC) and experimented on #Neurons, #Layers and.





From the above graphs it is quite evident that after a certain point, increasing the number of neurons doesn't seem to have a significant effect on the validation score. In Stay Alert Dataset, 15 neurons seems to do the trick while in the Otto dataset after 7 the test/validation scores seem to stabilize around 0.75. Also note that training score continues to spike by increasing the neurons in both cases. This might indicate a slight case of overfitting: adding more neurons makes the Neural Network highly sensitive to the training data, thereby hampering its ability to generalize on the validation/test set. We now proceed to experiment with Number of Layers.



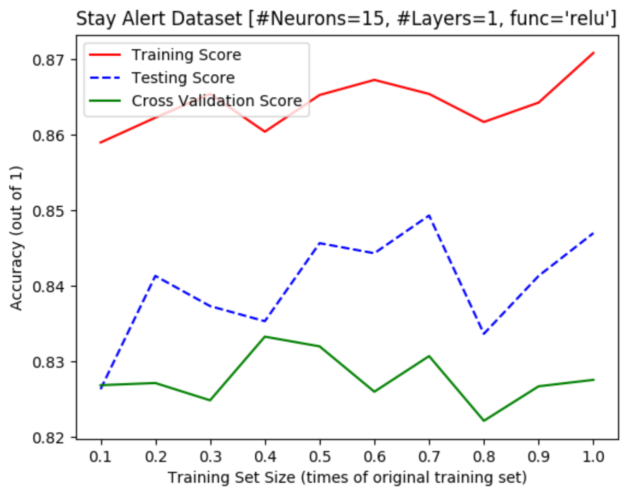


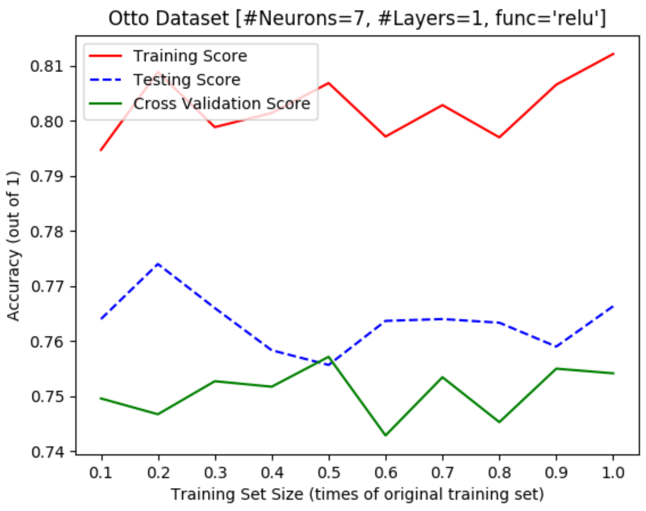
Both graphs show the send trend; Increasing Training accuracy & decreasing Test/Validation accuracy with increasing number of layers. This is a classic case of **over-fitting.**

The above two parameters determined the complexity of the Neural network. If our dataset is very complex, higher number of neurons and layers should be used to better represent and generalize the training and testing sets respectively, while if your dataset is relatively simple, we can get away with lower values for #Neurons and #Layers. Thus, from above we can infer that neither of our datasets are highly complex enough that they warrant complex neural networks. If anything, they discourage that. Activation function also plays an important role in the behavior the neural network and consequently on its performance.

Activation Function	**Stay Alert Cross Validation Score**	**Otto Cross Validation Score**
Identity	0.8272845837234448	0.7500012243169681
Logistic	0.8242832359095636	0.7550022473064776
Tanh	0.8281406243646142	0.7567204590924351
Relu	0.830712175839983	0.7588568597788066

Changing the activation sheds no further insights on the behavior of the neural network or the nature/characteristics of the dataset as the validation scores on both datasets are very evenly matched for all the functions. Since in both cases, 'relu' outperforms the rest albeit by an insignificant margin I can only recommend that for now. We now proceed to evaluate how changing the training size affects the performance of our neural networks trained on the most optimal hyperparameters we've found above.





We notice that highest testing accuracy of ~84% is achieved at 0.7x max training size for Stay Alert Dataset, while at only 0.2x max training size for Otto Dataset with its highest testing accuracy being ~77.5%. This proves the conjecture from before that the neural network tends to overfit easily even with less aggressive hyperparameters.

Conclusion

Below is the table which summarizes the results of the experiments.

Classifier	**Hyperparameters (Stay Alert)**	**Stay Alert Dataset - Best Test Score**	**Hyperparameters (Otto Alert)**	**Otto Dataset – Best Test Score**
Decision Trees	Max_depth = 10	0.874	Max_depth = 16	0.74
SVM	Kernel = 'rbf'; C = 1	0.8236666666666667	Kernel = 'linear; C = 1	0.764
AdaBoost with Decision Trees	Max_depth = 13; n_estimators = 600	0.9016666666666666	Max_depth = 13; n_estimators = 50	0.7676666666666667
Knn	K=1; distance_metric=Manhattan	0.833	K=1; distance_metric='euclidean'	0.7753333333333333
Multi-Layer Perceptron Classifier	#Neurons = 15; #Layers = 1; #Iterations = 200; Activation = 'relu'	0.8312838086447244	#Neurons = 7; #Layers = 1; #Iterations = 200; Activation = 'relu'	0.774

References

[1] https://www.kaggle.com/c/stayalert/data
[2] https://www.kaggle.com/c/otto-group-product-classification-challenge/data
[3] http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.141.880&rep=rep1&type=pdf