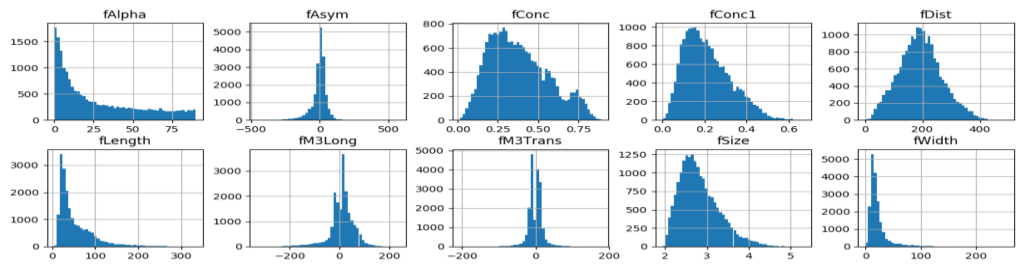
Datasets

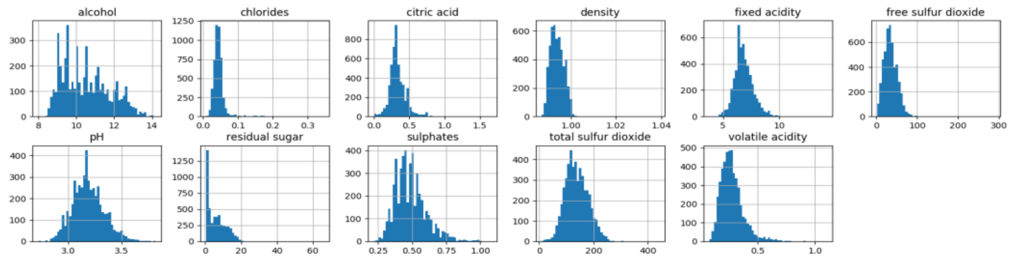
I have selected two datasets from the UCI Machine Learning Repository site: MAGIC Gamma Telescope Data Set and Wine Quality Data Set for this assignment.

The MAGIC Gamma Telescope Data Set (Magic04) is Monte Carlo generated to simulate registration of high energy gamma particles in a ground-based atmospheric Cherenkov gamma telescope using the imaging technique. Cherenkov gamma telescope observes high energy gamma rays, taking advantage of the radiation emitted by charged particles produced inside the electromagnetic showers initiated by the gammas. The dataset has 10 attributes of the Cherenkov radiation (of visible to UV wavelengths) that leaks through the atmosphere and gets recorded in the detector and classification of gamma (signal) and hadron (background). The dataset contains 19020 samples of which 12332 are gamma and 6688 are hadron. Note that in the description of the dataset states: “classifying a background event as signal is worse than classifying a signal event as background” but I am treating this dataset as a simple classification for the purpose of this exercise.



The histograms of the attributes show some are close to be normally distributed and few others have heavy tail. The correlation between the classification and attributes of the dataset identify that fAlpha (-0.46), fLength (-0.31) and fWidth (-0.27) are the highest correlated (negatively) attributes that likely to be important in the algorithms.

The Wine Quality Data Set includes results of objective tests (e.g. PH values, etc.) of white variants of the Portuguese "Vinho Verde" wine. The dataset has 11 attributes of the tests including fixed acidity, volatile acidity, citric acid, residual sugar, chlorides, free sulfur dioxide, total sulfur dioxide, density, pH, sulphates and alcohol. The dataset contains 4898 samples with the classifier based on sensory testing (median of at least 3 evaluations made by wine experts), each expert graded the wine quality between 0 (very bad) and 10 (very excellent). For the purpose of this exercise I have reduced the classification to two classes: excellent wine with quality > 6 based on the original expert opinion and the rest. The final dataset includes 1060 samples of excellent wine.



The histograms of the attributes except for alcohol are close to be normally distributed and also some have heavy tails. The attributes alcohol (0.39), density (-0.28) and chlorides (-0.18) have the highest correlation to the classification. This dataset has smaller size than the first one that can be problematic for some machine learning algorithms, also in the description is indicated that due to the privacy and logistic issues a number of attributes were removed, some of them could be significant.

Data Preparation

Both dataset was scaled by using Scikit-Learn MinMaxScaler to the interval [0,1] that improves the performance of the SVM, kNN and neural network algorithms, other algorithms, like Decision Tree, are not impacted by the scaling. Training and test dataset was created by Scikit-Learn train_test_split function that randomly assign 80% of data to the training dataset and 20% of data to the test dataset. The dataset split to the training and testing data was using the stratify option to achieve similar distribution to the original dataset. Scikit-Learn GridSearchCV class was used to perform exhaustive search on selected parameters (hyper-parameters) for optimal values for each algorithm. The implementation is using cross-validation grid search over the hyper-parameters.

Decision Trees

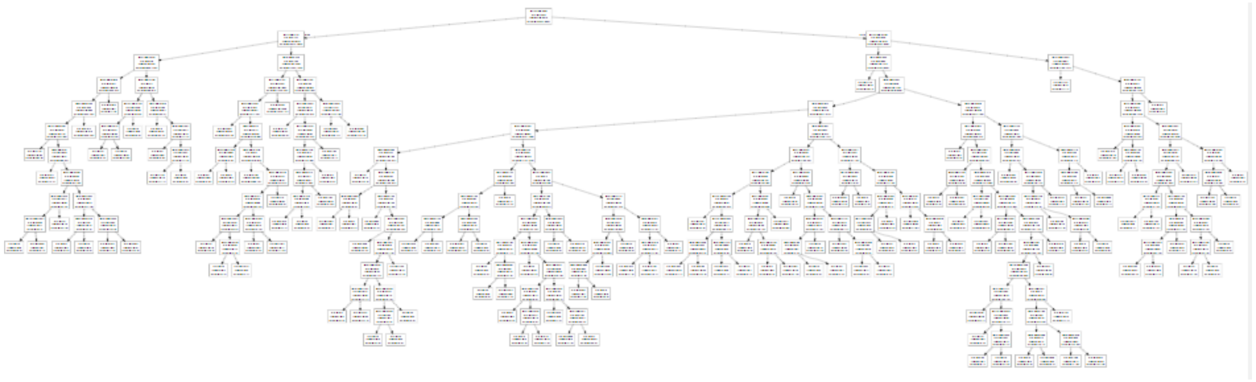
I was using SciKit-Learn DecisionTreeClassifier for the decision tree algorithm implementation, it implements an optimized version of the CART algorithm (similar to C4.5 algorithm). The decision tree algorithm is easy to understand and interpret, robust with outliers, it works with any kind of data and does not require scaling, though for simplicity of the implementation the scaled data was used that was prepared for other algorithms. The decision trees algorithm without pruning does not generalize the data well, it will create complex and deep tree that is prone to the overfitting. Currently the SciKit-Learn DecisionTreeClassifier does not support pruning (reducing the size of the completed decision tree based on some metric) but there are maximum depth of the tree and maximum number of leaf nodes hyper-parameters for the pre-pruning of the decision tree. The pre-pruning approach that makes global decision for the whole tree may not be as effective as post-pruning that allows to make pruning decisions on leaf-per-leaf metrics but it still provides reasonable approach to reduce the overfitting. Also, the decision tree algorithm can be unstable in the sense that any changes of training data can significantly change the generated decision tree. The generated decision tree can also be biased in the case of unbalanced data so that the stratifying option in creating training and test datasets should reduce this problem.

The run time complexity of the decision tree implementation in SciKit-Learn library is O(m*n*log(n)) where the m is the number of features and n is the number of instances. The run time cost of query is O(n).

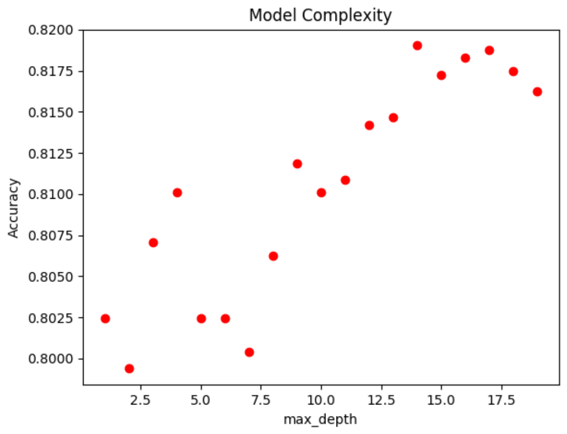
The implementation has number of parameters including: criterion – measure of the split quality, max_depth – maximum depth of the tree, min_sample_split – minimum number of samples to split node, min_samples_leaf – minimum number of samples at leaf, max_leaf_nodes – maximum number of leaf nodes in the tree. After the initial cross-validation runs I have selected criterion = "gini" and the default parameter values for min_sample_split and min_sample_split. The two other parameters were used in the grid cross-validation training to find the best model.

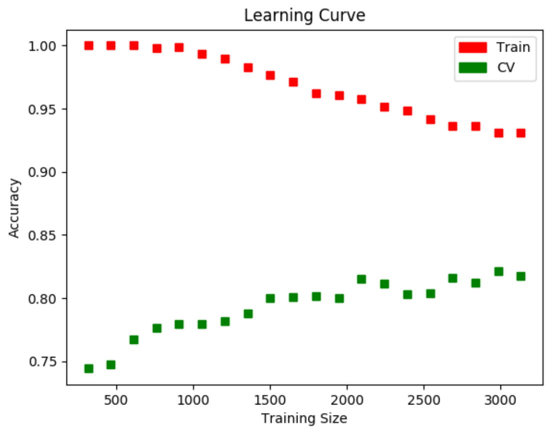
The initial (not pruned, overfitted) decision tree model for the Wine dataset achieves the 100% accuracy on the training set as expected by creating full tree:

The cross-validation testing with max_depth and max_leaf_nodes parameters identify the correspondent best values of 14 and 140. The decision tree is substantially smaller:



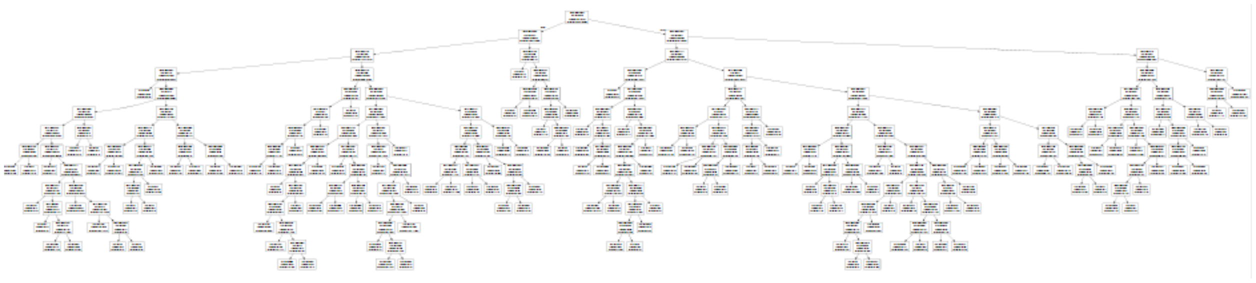
The following Model Complexity graph displays how the cross-validation model accuracy changes with the value of max_depth parameter (parameter max_leaf_node was fixed at 140) and maximum accuracy is achieved at 14. The Learning Curve graph displays the convergence of the training and CV accuracy, for the Wine dataset the difference between training and CV accuracy is substantial that indicates high variance and model might improve with additional data but looks like the dataset is noisy and Decision Tree may be not appropriate algorithm.



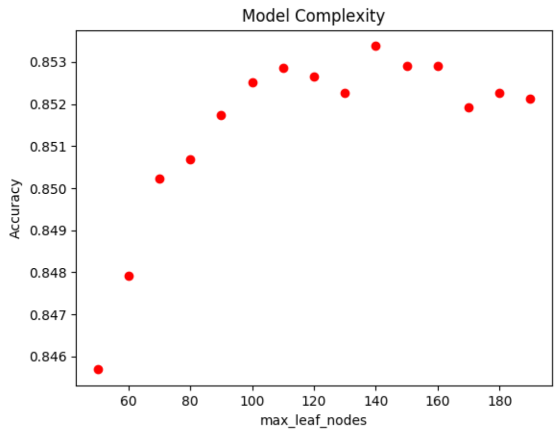


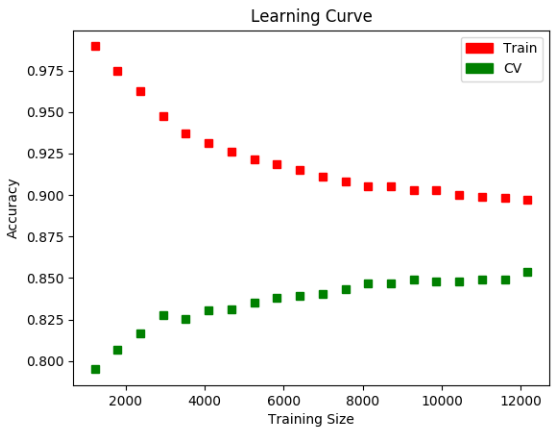
The final decision tree model for the Wine dataset achieved 83.67% accuracy on the test set.

Similar initial decision tree model for the Magic04 dataset achieve 100% accuracy before pruning, it is much bigger in size and is not displayed here. After cross-validation testing the best values of max_depth and max_leaf_nodes parameters were 12 and 140 with the pre-pruned decision tree:



The following Model Complexity graph displays how the cross-validation model accuracy changes with the value of max_leaf_node parameter (parameter max_depth was fixed at 12) and the maximum accuracy is achieved at 140. The Learning Curve graph displays the convergence of the training and CV accuracy, for the Magic04 dataset the difference between training and CV accuracy is reduced to 5% and with curves flattening, we might achieve reasonable results for this dataset and algorithm.





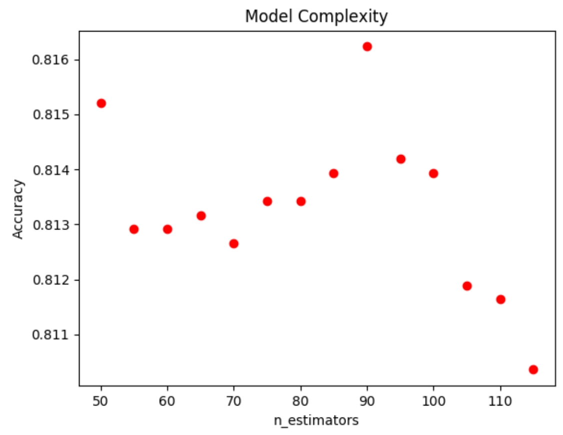
The final decision tree model for the Magic04 dataset achieved 85.02% accuracy on the test set.

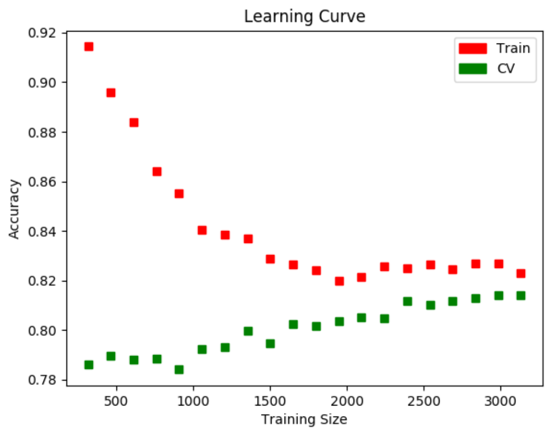
Boosting

I was using SciKit-Learn AdaBoostClassifier for the boosting algorithm implementation, it implements the AdaBoost-SAMME version of algorithm. Ada-boost is one of the ensemble classifier algorithms that combines results of multiple weak learners to provide overall solution. It uses multiple weak learners with training sets based on the accuracy of the previous steps and combines them with weight-age of the achieved accuracy. Ada-boost, as well as other boosting algorithms, usually avoids overfitting because multiple weak learners are used with different training datasets modified by weights. The solution provided by the boosting algorithms are usually not as easy to interpret as for the decision tree.

The implementation has number of parameters including: n_estimators – maximum number of estimators at which boosting is terminated, learning_rate – shrinks contribution of each classifier, algorithm – flavor of algorithm, base_estimator – type of estimator for ensemble. After the initial cross-validation runs I have selected default values for algorithm = “SAMME.R” and base_estimator = “DecisionTreeClassifier”. The two other parameters were used in the grid cross-validation training to find the best model.

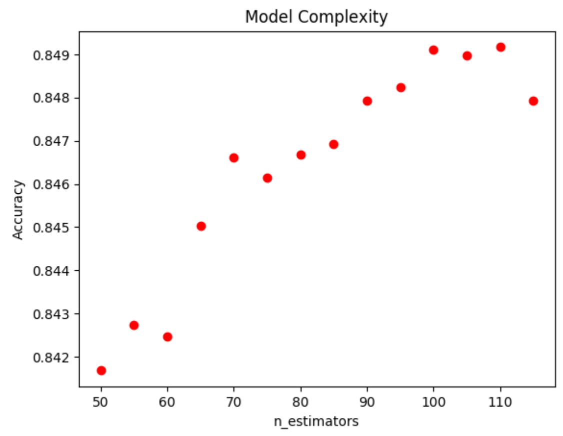
The boosting model for Wine dataset using cross-validation training identified the best values for n_estimators and learning_rate parameters equal to 90 and 0.5, the following Model Complexity graph displays the accuracy values for different n_estimators parameters (learning_rate was fixed at 0.5). The Learning Curve graph display convergence of the training and cross-validation errors with the increasing number of instances.

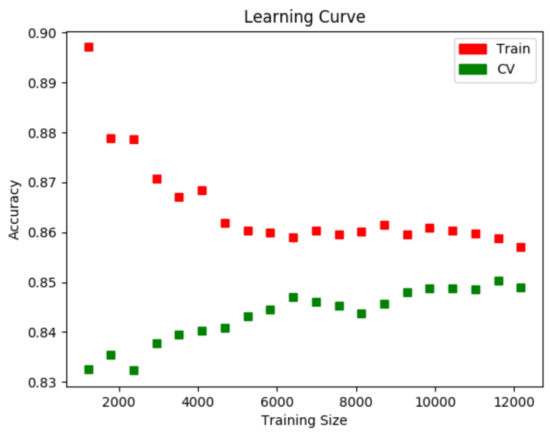




The final boosting model for the Wine dataset achieved 81.94% accuracy on the test set.

Similar the boosting model for Magic04 dataset using cross-validation training identified the best values for n_estimators and learning_rate parameters equal to 110 and 0.7, the following Model Complexity graph displays the accuracy values for different n_estimators parameters (learning_rate was fixed at 0.7). The Learning Curve graph display convergence of the training and cross-validation errors with the increasing number of instances.





The final boosting model for the Magic04 dataset achieved 84.31% accuracy on the test set.

K Nearest Neighbors

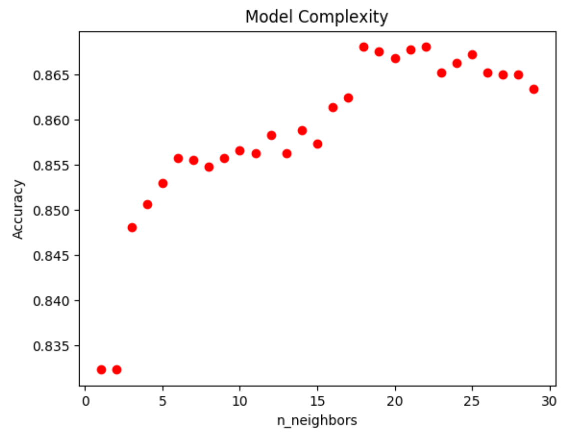
I was using SciKit-Learn KNeighborsClassifier for the k nearest neighbors algorithm implementation. Nearest neighbors classification is instance based learning algorithm that is different from the rest of the algorithms in this assignment. It belongs to the lazy algorithms that are not generalizing model during the training but just storing all the instances of the training set. The generalization happens at the query time when the classification is determined by majority votes of the k nearest neighbors of the new data point.

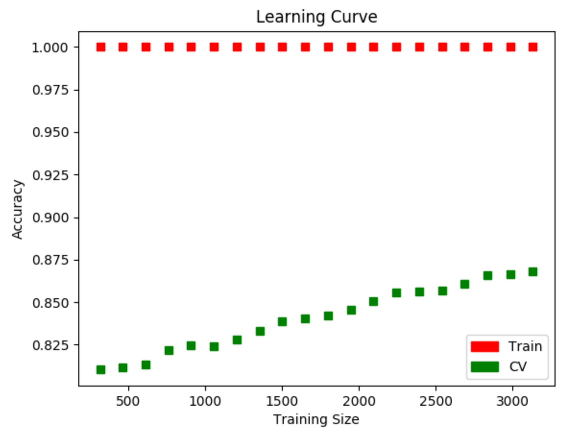
The most important parameter of the nearest neighbors algorithm is the number of neighbors K and it's highly data dependent. The larger values of K reduce the effects of noise on the classification prediction but also make classification boundaries less distinct. The other important parameters of the algorithm are the distance (similarity) metrics for determining the nearest neighbors and the weights that can be assigned to every neighbor such that the closest points are more influential on the classification result. In this exercise instead of simple majority vote, the weighted majority vote is used for classification.

The run time complexity of the nearest neighbors implementation in SciKit-Learn library is O(n) during training because the algorithm just store samples. The run time cost of query is O(m*n*n) for brute force approach and O(m*n*log(n)) using tree-based storage where the m is the number of features and n is the number of instances.

The implementation has number of parameters including: n_neighbors – the number of neighbors, weights – weight function, algorithm – computation of nearest neighbors, metric – distance metrics used, P – power for Minkowski metrics. After initial cross-validation runs I have selected weights = "distance" and the default parameter values for algorithm and metrics. The two other parameters were used in the grid cross-validation training to find the best model.

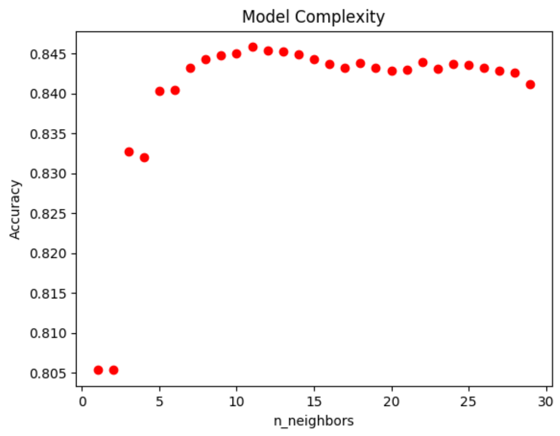
The KNN model for Wine dataset achieves 100% accuracy on the training set because the "distance" weight parameter sets the highest weight on the closest data point which is the original classification value. The cross-validation training with n_neighbors and P parameters identify the corresponding best values of 18 and 3. The following Model Complexity graph displays how the cross-validation model accuracy changes with the value of n_neighbors parameter (parameter P was fixed at 3) and the maximum accuracy is achieved at 18. The Learning Curve graph is not as useful for the KNN algorithm with "distance" weights because the training set stays at 100% accuracy but we can see that CV accuracy is still increasing and model can be improved with additional data.

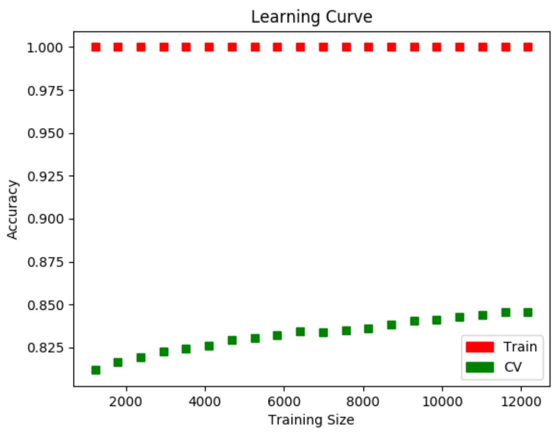




The final KNN model for the Wine dataset achieved 86.53% accuracy on the test set.

Similar initial KNN model for the Magic04 dataset also achieves 100% accuracy on the training set. After cross-validation testing the best values of n_neighbors and P parameters were 11 and 1 (1-degree Minkowski or Manhattan distance). The following Model Complexity graph displays how the cross-validation model accuracy changes with the value of n_neighbors parameter (parameter P was fixed at 1) and the maximum accuracy is achieved at 11. The Learning Curve graph, similar to Wine dataset graph, does not display convergence of training and CV accuracy but the CV accuracy is flattening so we are achieving reasonable results with this bigger dataset.





The final kNN model for the Magic04 dataset achieved 84.28% accuracy on the test set.

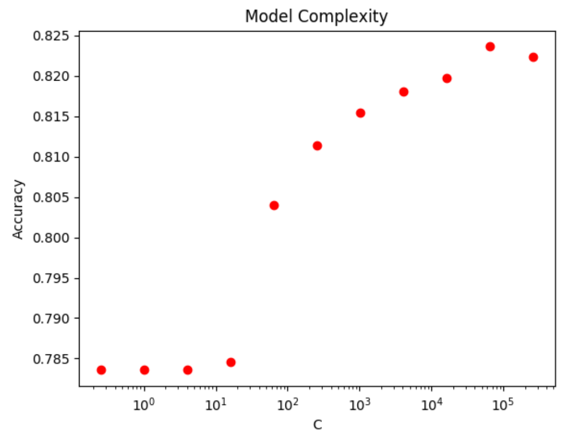
SVM

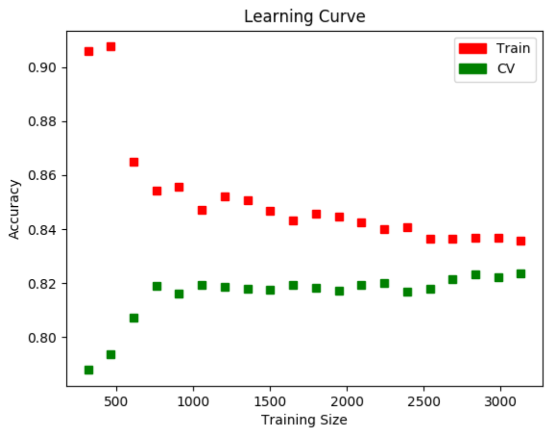
I was using SciKit-Learn SVC (support vector classification, libsvm-based implementation) for support vector machine algorithm implementation. SVM is one of the most popular classification algorithms that is searching for hyperplanes or multi-dimensional curves (in the case of non-linear kernel) that separates the data instances. SVM algorithms are very robust against overfitting and still effective in high dimensional spaces but it requires the data scaling to achieve the optimal coefficients calculation. The algorithm is most effective when there is clear separation of instance classes which is rarely happens in real problems but it still can achieve good results when the mix of classes is limited.

The run-time complexity of the SVM implementation in SciKit-Learn library is between O(m*n*n) and O(m*n*n*n) depending on the dataset and effectiveness of cache where the m is the number of features and n is the number of samples. The run time cost of query is O(n). SVM is memory intensive and do not scale well to larger datasets.

The implementation has number of parameters including: C – penalty parameter of the error term (trade-off between classification error and margins), kernel – the kernel type used in algorithm, also kernel specific parameters – degree and gamma. After the initial cross-validation runs I have selected kernel = “rbf” that was providing better accuracy then other kernels: “linear”, “poly” and “sigmoid” for multiple combinations of hyper-parameters. The C parameter was used in the grid cross-validation training to find the best model.

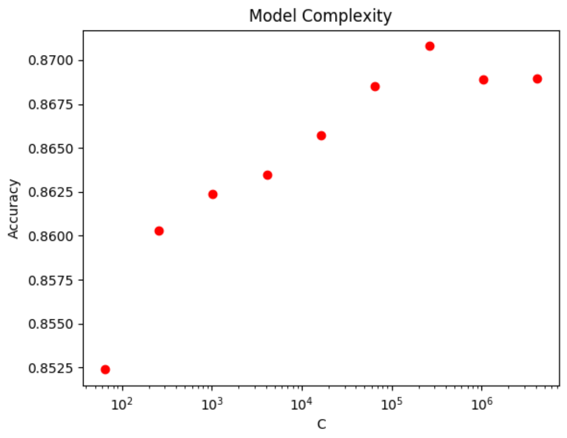
The SVM model for Wine dataset using cross-validation training identified the best value for C parameter equals to 65,536 (2 in the power 16), the following Model Complexity graph displays the accuracy values for different values of C parameters. The Learning Curve graph display convergence of the training and cross-validation errors with the increasing number of instances.

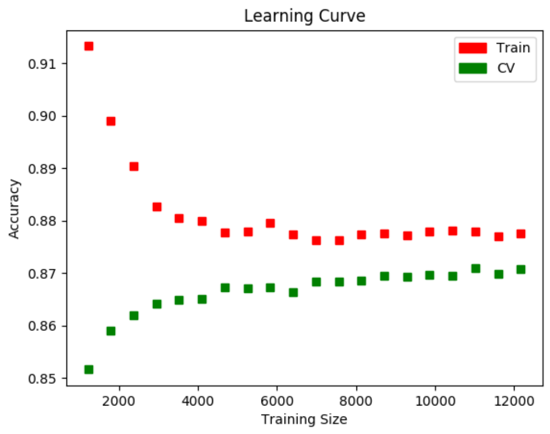




The final SVM model for the Wine dataset achieved 82.24% accuracy on the test set.

The SVM model for Magic04 dataset using cross-validation training identified the best value for C parameter equals 262,144 (2 in power 18), the following Model Complexity graph display the accuracy values for different C parameters. The Learning Curve graph display convergence of the training and cross-validation errors around 7,000 samples and both curves are flattening at that point.





The final SVM model for the Magic04 dataset achieved 86.25% accuracy on the test set.

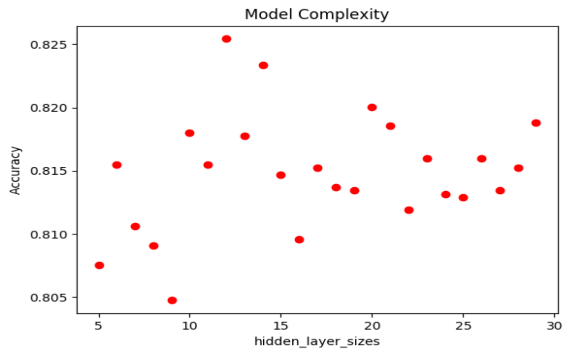
Neural Networks

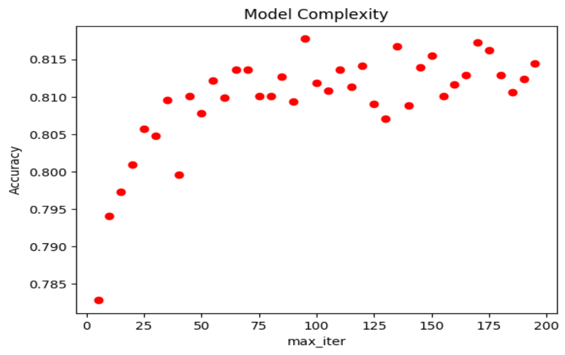
I was using SciKit-Learn MLPClassifier (Multi-Layer Perceptron) for neural network algorithm implementation. It is a supervised machine learning algorithm that can learn non-linear function approximation for classification using backpropagation for training. It consists of input layer, output layer and one or more hidden layers. The multi-layer perceptron algorithm is capable to learn non-linear models in real time (on-line learning). The algorithm is sensitive to the data scaling and it has non-convex loss function so that the calculated solution could be a local minimum and different initial values for the weights can provide different solutions. Currently MLPClassifier implementation has only cross-entropy loss function, and stochastic gradient descent (SGD), L-BFGS and Adam (stochastic gradient descent optimizer) algorithms for backpropagation calculations. The neural networks have the highest number of hyper-parameter out of these classification algorithms.

The run-time complexity of the neural network implementation in SciKit-Learn library is $O(n \cdot m \cdot h^k \cdot o \cdot i)$ where the m is the number of features, n is the number of samples and the model has o output neurons, k hidden layers each with h neurons, repeated i iterations (h, k, and o can be considered constants when model is selected). The run time cost of query is O(n).

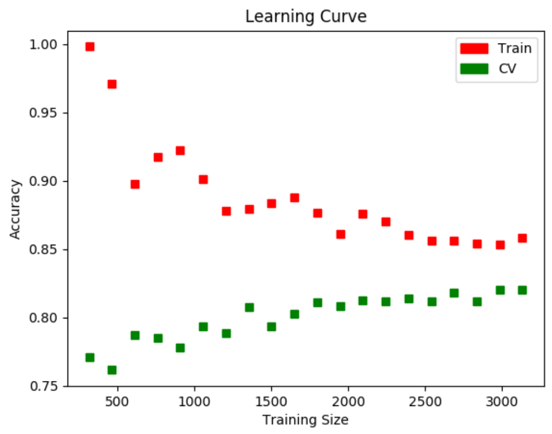
The implementation has number of parameters including: hidden_layer_sizes – the number of neurons in each hidden layer (tuple), activation – function for the hidden layer neurons, solver – gradient descent implementation for the backpropagation, alpha – regularization term parameter, learning_rate – schedule for the weight updates, max_iter – maximum number of iterations (epochs) until convergence. After the initial cross-validation runs I have selected solver = "lbfgs" that is converging faster and performs better on not-so-big datasets, activation = "tanh" that was providing better accuracy then other activation functions: "identity", "logistic", "relu" and the default learning_rate = "constant". Also, I have decided to limit the number of hidden layers to 2 because the results show some improvement compared to the single layer but the further increase did not improve the model performance. The other two parameters hidden_layer_sizes and alpha were used in the grid cross-validation training to find the best model.

The neural network model for Wine dataset using cross-validation training identified the best values for hidden_layer_sizes and alpha parameters equal to 12 and 0.001, the following Model Complexity graphs display the accuracy values for different values of hidden_layer_size (the number of neurons at each of the two layers) and max_iter. The best accuracy achieved at 12 neurons per layer but graph does not show any obvious pattern. The second graph displays that the reasonable accuracy achieved at 70 iterations and the further iterations does not provide significant benefits.



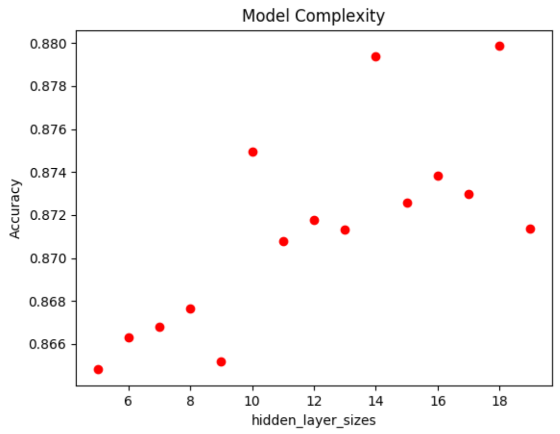


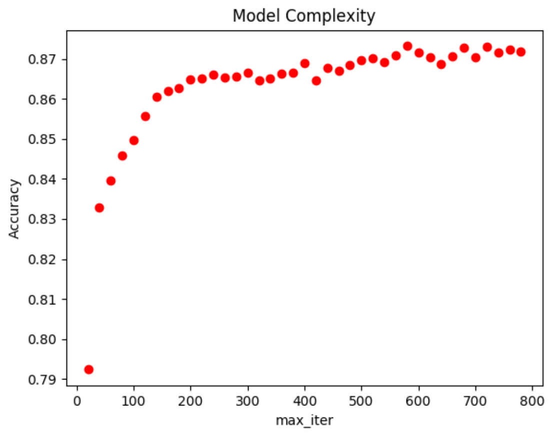
The Learning Curve is flattening but does not display convergence yet for Wine dataset, like other algorithms, the neural network can profit from additional data to reduce variance in the model.



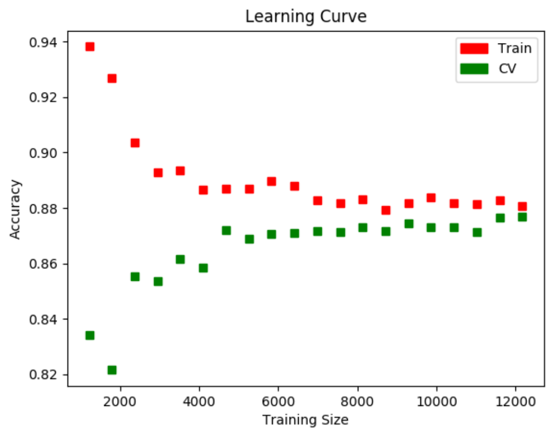
The final neural network model for the Wine dataset achieved 83.57% accuracy on the test set.

Similar neural network model for Magic04 dataset using cross-validation training identified the best values for hidden_layer_sizes and alpha parameters equal to 18 and 0.0001, the following Model Complexity graphs display the accuracy values for different values of hidden_layer_size (the number of neurons at each of the two layers) and max_iter. The best accuracy achieved at 18 neurons per layer and after 250 iterations there is only limited improvements in the model accuracy.





The Learning Curve is converging very nicely and there is not significant difference between training and cross-validation accuracy in the end.



The final neural network model for the Magic04 dataset achieved 86.67% accuracy on the test set.

Conclusion

In this assignment I have applied five different supervise machine learning classification algorithms to two different datasets. Even though the results of the different algorithms were consistent, for Wine dataset the accuracy is within 5% and for Magic04 dataset within 2.5%, it also highlights differences between algorithms. There is significant difference between performance of SVM and kNN algorithms. For the Wine dataset, the kNN algorithm has the best performance and SVM one of the worst and just opposite for the Magic04 dataset. I think the difference in behavior was caused by data, the Magic04 has data samples that are better separable by hyper-curves and SVM works well but the Wine has “islands” of classifications and the local algorithm kNN works better. The boosting algorithm converges well for both datasets and provides low variance (no overfitting) solution but sacrifices some accuracy. The Decision Tree algorithm with pre-pruning gives one of the best solutions but it has high variance and is sensitive to the data noise that is present in both datasets. The neural network algorithm provides the best solution for the Magic04 datasets but still have high variance for the Wine dataset. It is also expected behavior because the performance of the neural network algorithm depends on the amount of training data and we are limited to only 4898 samples in Wine dataset.

Generally speaking the Wine dataset is difficult to predict. As described in the beginning, it is missing some attributes that could be predictive of the classification and has a limited data size. The excellent wine classification includes 1060 samples out of 4898 which is 21.64% (the rest is 78.36%) so our results 81.94% to 86.53% of accuracy is not that far off from baseline. The Magic04 dataset displays better results, the gamma (signal) classification has 12332 samples out of 19020 which is 64.84% so our results 84.28% to 86.67% of accuracy is a good improvement from baseline. Though results for both datasets show high bias (especially Wine dataset) and additional attributes can improve performance. The different sizes of the datasets also highlight the dependence of the machine learning algorithms (especially neural networks and decision tree) on the number of available samples. The smaller Wine dataset has many learning curves that did not completely converge and can benefit from additional data. The learning curves look better with bigger Magic04 dataset.

The exercise also displays the expected run-time behavior of the machine learning algorithms. The instance-based learning kNN algorithm has very fast training time but much slower query time where the rest of the algorithms are opposite. The training performance of the neural network and especially SVM algorithms depends on the number of samples so we see significant increase in the training time with the size of dataset.

Below are consolidated results of the algorithms performance on the Wine and Magic04 datasets. It includes the training and query runtime, the accuracy on the training set, cross-validation with the best hyper-parameters and test set.

Dataset		Decision Tree	Boosting	kNN	SVM	Neural Net
Wine	Training, sec	0.038	0.279	0.019	10.490	1.627
	Query, sec	0.002	0.013	0.463	0.037	0.002
	Training Accuracy	92.45%	81.90%	100%	83.64%	84.81%
	Cross-Valid. Accuracy	81.93%	81.62%	86.80%	82.36%	82.54%
	Test Accuracy	**83.67%**	**81.94%**	**86.53%**	**82.24%**	**83.57%**
Magic04	Training, sec	0.176	1.836	0.028	318.523	12.650
	Query, sec	0.001	0.041	0.228	0.386	0.005
	Training Accuracy	89.32%	85.68%	100%	87.81%	87.74%
	Cross-Valid. Accuracy	85.38%	84.92%	84.58%	87.08%	87.98%
	Test Accuracy	**85.02%**	**84.31%**	**84.28%**	**86.25%**	**86.67%**

References

R. K. Bock. Major Atmospheric Gamma Imaging Cherenkov Telescope project (MAGIC). http://wwwmagic.mppmu.mpg.de (rkb@mail.cern.ch)

P. Cortez, A. Cerdeira, F. Almeida, T. Matos and J. Reis. Modeling wine preferences by data mining from physicochemical properties. In Decision Support Systems, Elsevier, 47(4):547-553. ISSN: 0167-9236.