Assignment 1: Supervised Learning

Chang Sun, Spring 2020

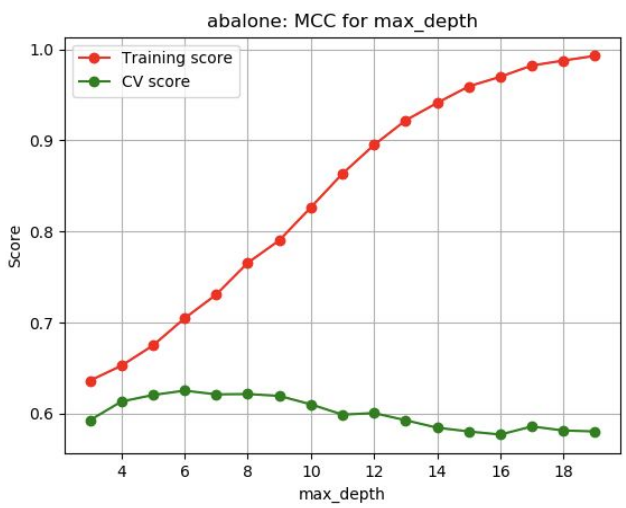
Problem 1: Predicting how many rings an abalone has
My first dataset is a collection of properties from abalone photographs, as well as labels that indicate how many rings the abalone has in real life. For the abalone dataset, I reconstructed the data to create a classification problem instead of a regression problem. The features are measurements of properties taken from photos of abalones, including various physical dimensions and weight, and the goal is to predict the relative number of rings the abalone has. The target set has three classes, based on the relative frequency of different ring numbers: `{< 9, 9-10, > 10}`. These bounds were chosen to keep the dataset balanced between the actual distribution of the abalone ring numbers.

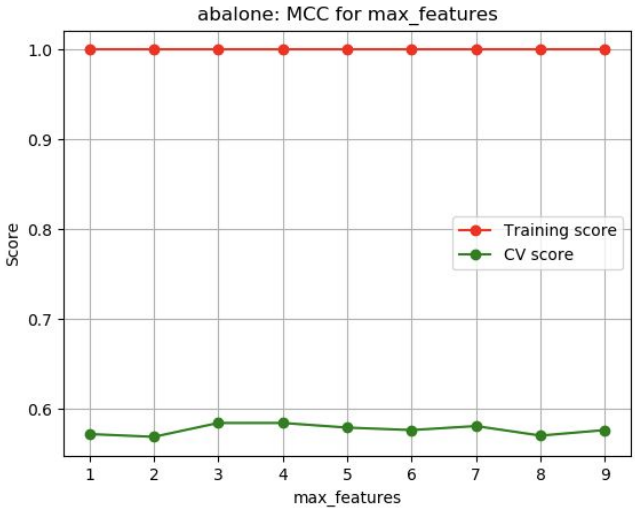
Problem 2: Predicting what browser online shoppers use
My second dataset is a dataset of customers visiting an online shopping portal. Here, I actually decided to restructure the dataset into one that predicts what type of browser the shopper is browsing with, rather than predicting whether or not they've purchased something. This is because the latter problem performs uniformly well across all of the algorithms due to how cleanly the data is structured as is. The reformulated problem takes into account features such as browsing date, whether they spent money, whether their search was related to a product, and what types of pages a customer hit. As such, this dataset is less balanced than the abalone dataset.

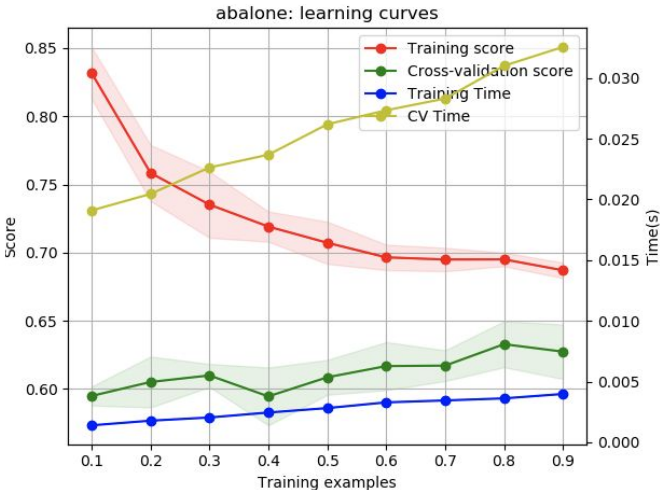
Why choose these datasets?
I chose the datasets I did because they are non-trivial enough to elude simple analysis. After encoding and transforming the data, the abalone dataset has 10 features, and the online shoppers intention dataset has over 15. Both have thousands of data points, which makes it possible to easily split the data into training and test sets, and possible to run 5-fold cross validation without dealing with low sample sizes. Other datasets I examined, such as the banknote authentication dataset on the UCI repository (see references), were too simple, and lacked the feature complexity required to make this exercise interesting. Plotting out learning curves for those revealed training and test set scores that were both above 0.9, and didn't diverge in meaningful or interesting ways.

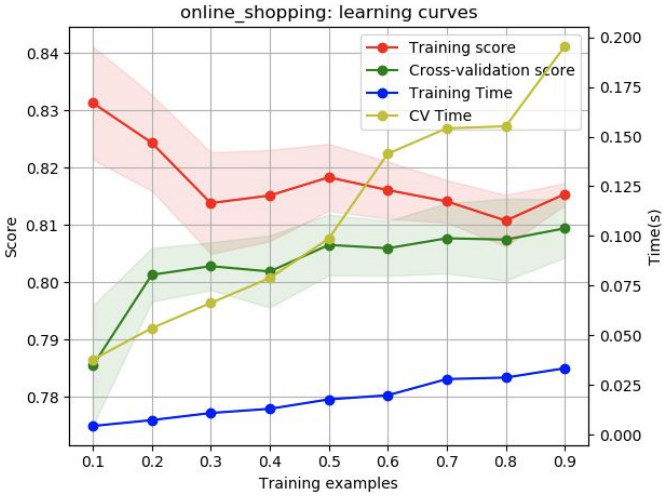
*(**Note:** I was enrolled in this class in Fall 2019 before dropping due to time commitments, and the writeups and charts in this assignment are largely based on what I submitted then)*

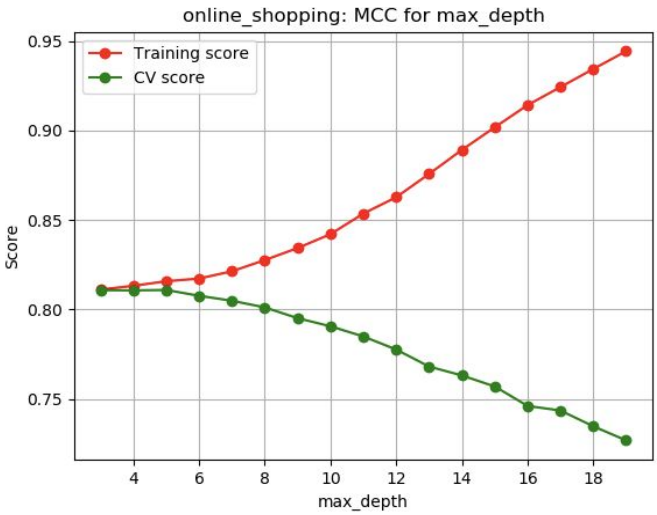
Decision Trees (note: datasets are labeled in the title of each chart)

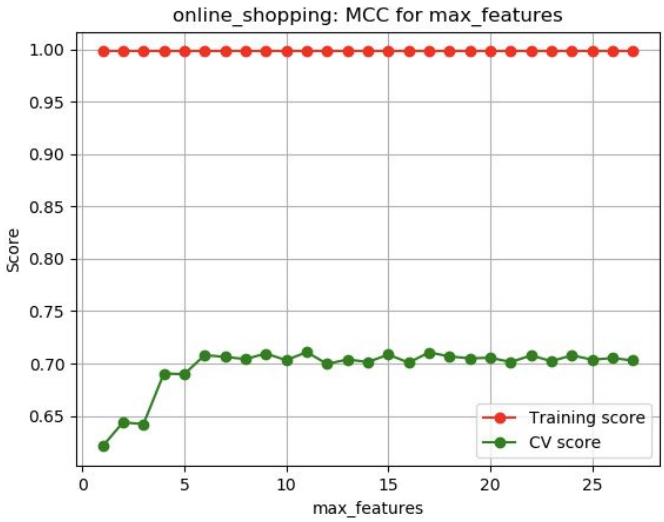












DTClassifier training set score for abalone: 0.6767020184741703
DTClassifier holdout set score for abalone: 0.5956937799043063
DTClassifier training set score for online_shopping: 0.8128837909859807
DTClassifier holdout set score for online_shopping: 0.8102189781021898

My DTClassifier performed much better on the online shopping dataset than the abalone dataset. I can think of several reasons: first, the former dataset has 3 possible result classes, whereas the latter only has 2 - it's likely that binary classification problem is easier than one with 3 output classes. Second, it appears that the online shopping dataset contains human-generated and classified data, and has been curated to a greater degree than the abalone dataset. Generally, we have an intuitive understanding about what factors might cause a particular online shopper to be a higher intent customer. Thus, we can pick a bunch of features that we think may be likely to contribute.

The learning curves show that as we increase the number of training samples, the training and CV scores converge as expected - though they converge faster for the shopping problem. The non-CV learning curve shows the bias-variance tradeoff clearly: when we don't have enough training examples, the model will overfit to the few examples available - this implies that more data does help performance. With proper CV, it appears that we can mitigate the effects of this tradeoff. Based solely on clock speed, the model performed better on the abalone classification problem. This is because the abalone dataset contains less features than the online shopper dataset - thus, the algorithm doesn't have to cycle through as many features when calculating out which one results in the best split.

Effect of varying hps, stopping criteria, etc.
I pre-pruned the sklearn decision tree algorithm by varying the `max_depth` hyperparameter - thus limiting the overall depth that the tree could form. Pre-pruning in this manner is intended to curb overfitting by making sure that we aren't indexing too much on the training data. Generally, we would expect a shallower-depth decision tree to perform worse on the training data, but better than the test data versus a decision tree where we don't prune at all. Sure enough, as seen in the model complexity curve above, when we vary `max_depth`, we see this exact behavior play out - the training and test set model scores begin diverging dramatically as we increase the depth of the tree.

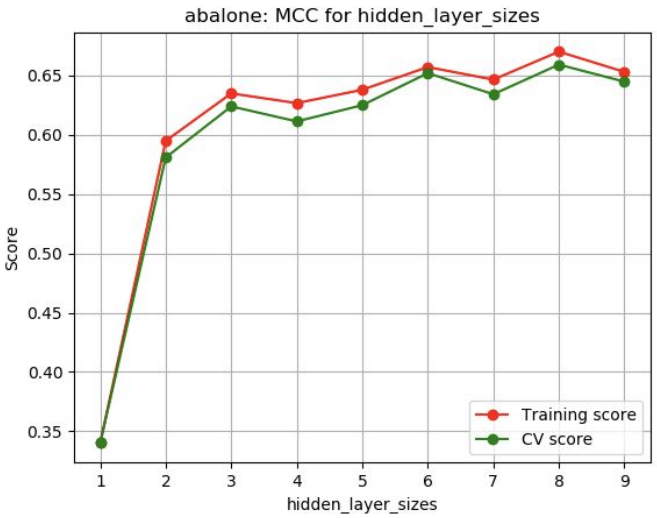
On the other hand, varying `max_features` doesn't seem to have as big of an impact on model overfitting. If anything, if there's a trend at all, it appears that the model performs marginally better on both datasets as we increase the number of features considered for each split.

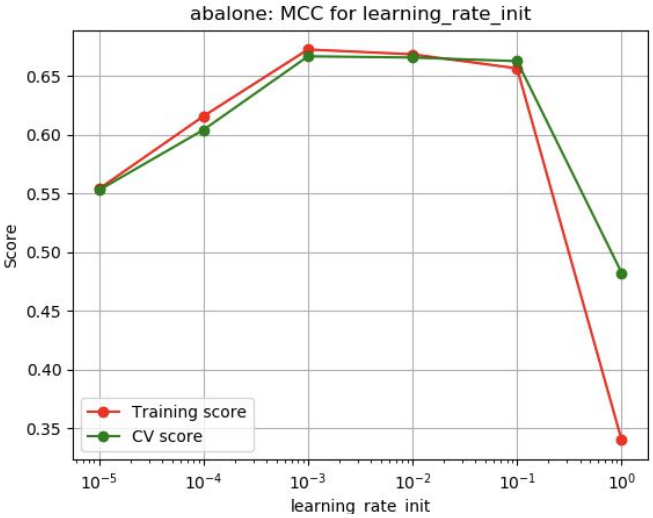
Would cross-validation help?
Cross validation helped a great deal when working with the training set. In comparing the non-cross validated vs. the cross-validated training scores, the latter were much closer to the final model scores when run on the test/holdout set.

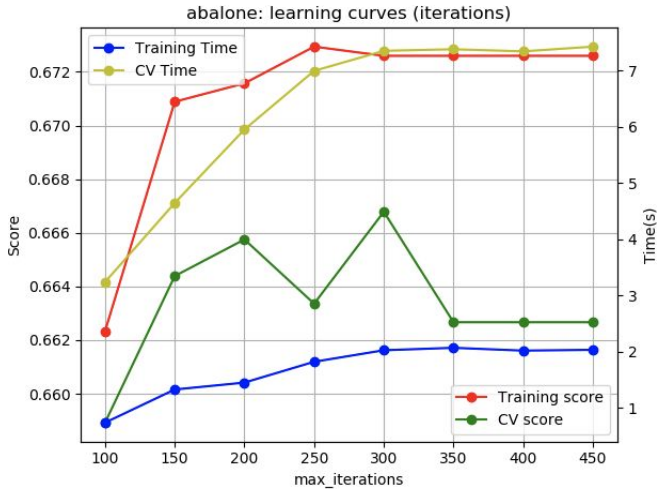
What sort of changes would I have made?
Given more time, I would consider trying to post-prune the tree instead of pre-pruning - theoretically this would yield higher scores than pre-pruning because we would be making more informed choices about when to stop growing the tree, whereas pre-pruning via `max_depth` is just an educated guess.

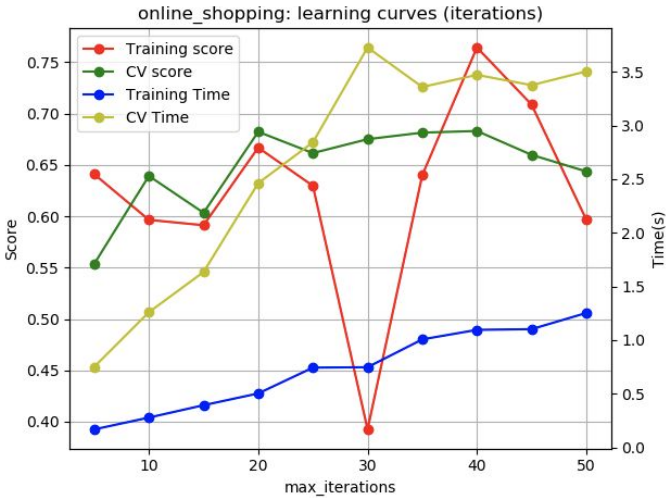
Neural Networks

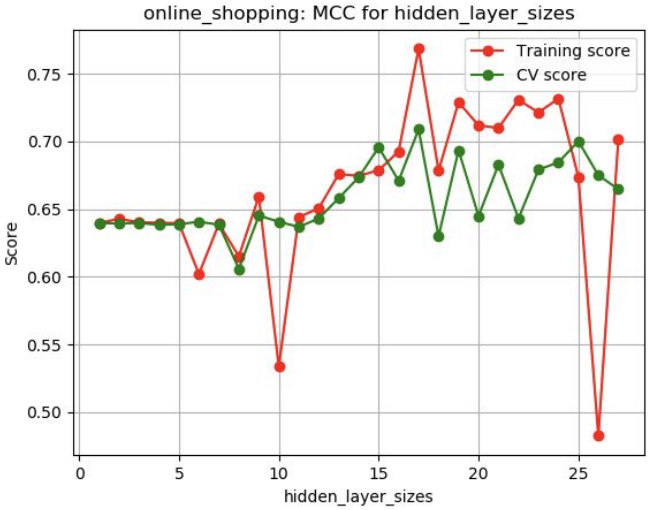
(note: only 1 hidden layer used, hidden_layer_sizes refers to the no. of hidden units in that layer)

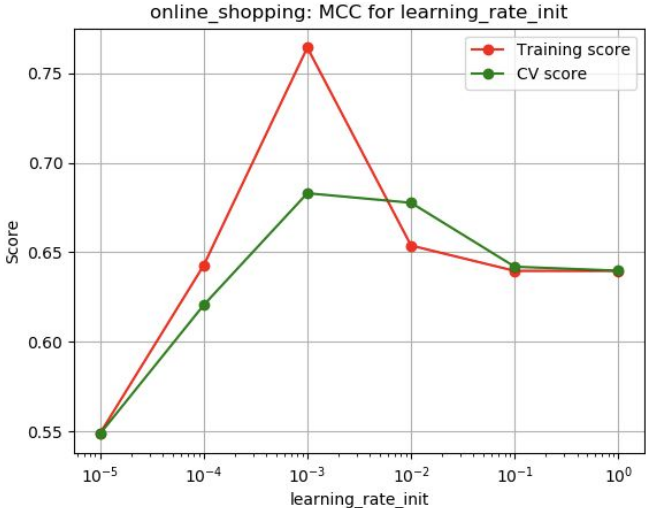












MLPClassifier training set score for abalone: 0.6702018474170373
MLPClassifier holdout set score for abalone: 0.6483253588516746
MLPClassifier training set score for online_shopping: 0.7686247248291044
MLPClassifier holdout set score for online_shopping: 0.7766964044336307

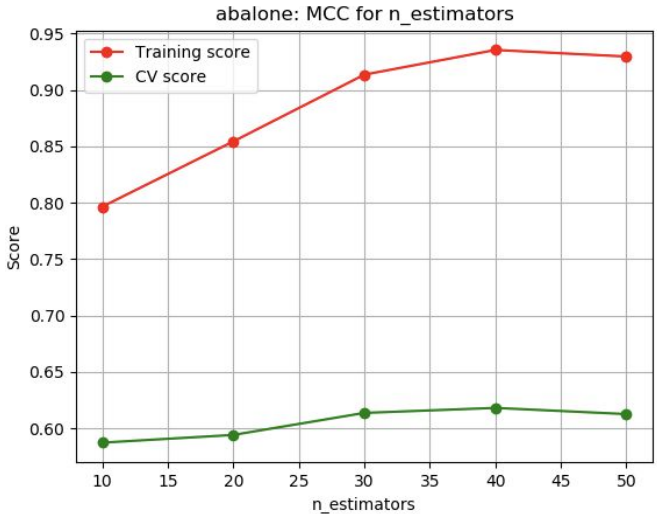
For my neural net classifier, I quickly discovered that one of the main bottlenecks to convergence was the `max_iters` parameter. As such, I decided to tune for that first, and use it in the subsequent model complexity analyses for the other hyperparameters. My `MLPClassifier` performed better on the online shopping dataset than the abalone dataset. As shown by the clock speed comparison below and my observations in the decision tree analysis, this is likely due to the abalone dataset being generally noisier than the shopping dataset. Interestingly, when looking at the actual test set scores, my `MLPClassifier` performed better than my `DTClassifier` on the abalone dataset, but worse on the online_shopping dataset. The result for the shopping dataset is interesting, as we don't necessarily see strong evidence of overfitting when examining the train vs. test scores in the complexity analyses.

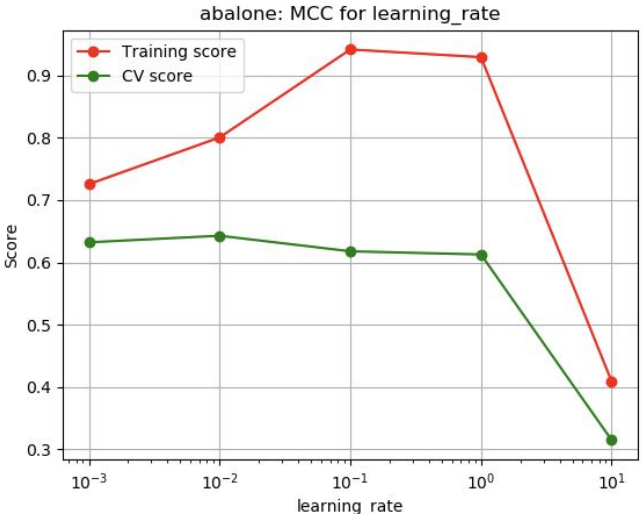
For my learning curves, I examined the effect of varying the number of epochs I trained the dataset on. For the shopping data, not using CV results in much noisier results, suggesting that CV is useful to reduce the model's sensitivity to the # of epochs. The ideal number of epochs for the abalone data is around 300, versus around 40 for the shopping data. This is not shocking, as the abalone dataset is more complex than the shopping dataset. Interestingly, it's harder to tease out a distinct pattern in the effects on the bias-variance tradeoff. The shopping learning curves appear to, after 25 iterations, swing wildly between underfitting and overfitting the data - this could be due to this dataset being more unbalanced than the abalone data, or other factors. In the abalone dataset, the cv scores generally don't diverge too much from the training scores except when we tune parameters to an extreme (e.g. set learning rate of 1). The abalone learning curves do not appear to show significant differences in model performance after 150 iterations beyond very slight overfitting.

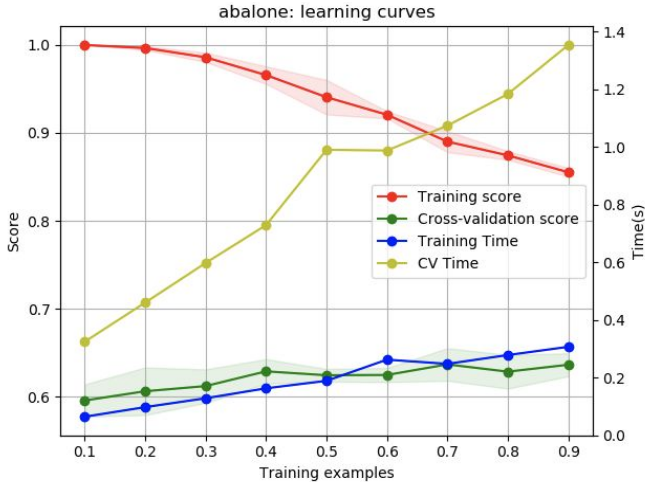
Interestingly, the model complexity curves for `hidden_layer_sizes` and `learning_rate_init` show a relatively small gap between the training and CV scores for both data sizes - the main difference is the relative noisiness of both hyperparameters. When looking at the effects of varying these HPs, we can see that learning rate had a bigger effect on the abalone dataset than the shopping dataset. For both datasets, the complexity curves show that we can converge on an ideal-ish learning rate (in the {0.001, 0.01} range (log scale)). Varying the number of hidden units within the hidden layer had an effect as well, though primarily on the abalone dataset. Perhaps due to its complexity, including only one hidden unit in the abalone dataset resulted in substantially poorer performance.

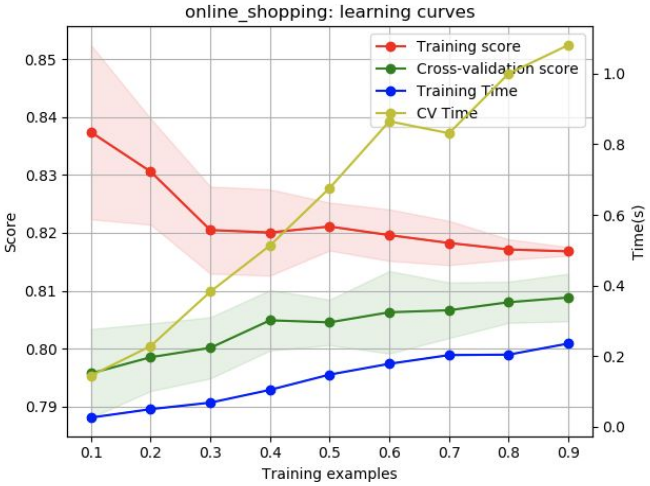
Looking at clock speed, this model performed better across different `max_iters` on the online shopping classification problem, in spite of it having more features than the abalone dataset. This is likely because the abalone dataset is noisier than the shopping dataset (as seen by their relative performance using the decision tree classifier). With a noisier dataset, gradient descent may hop around a lot more before converging on an optimum. Cross-validation still helps, but the effect is not as drastic as we saw in the decision tree. In the shopping dataset, the cv scores are actually higher than the training scores, which is certainly interesting. This may be another indication that this dataset is less noisy or more linearly separable.

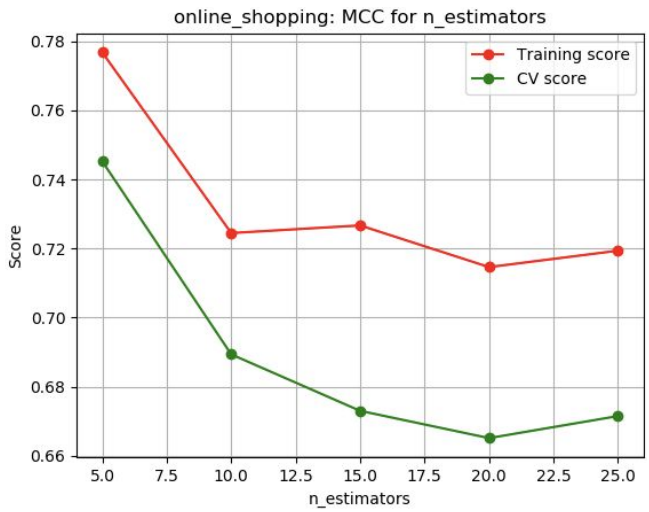
Boosting

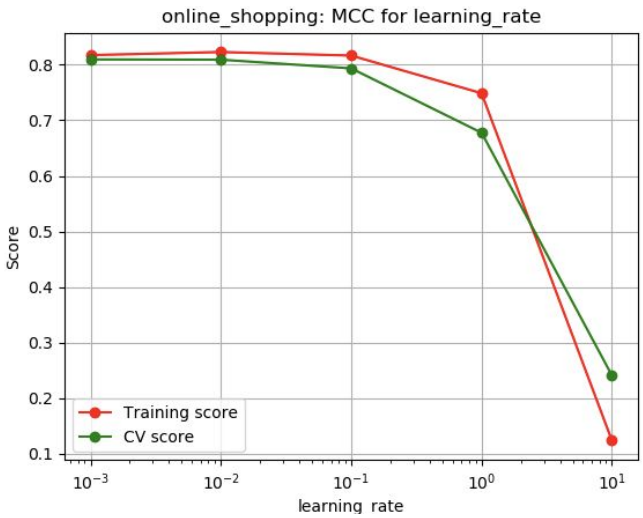

abalone: MCC for n_estimators
Training score
CV score
Score
n_estimators


abalone: MCC for learning_rate
Training score
CV score
Score
learning_rate


abalone: learning curves
Training score
Cross-validation score
Training Time
CV Time
Score
Time(s)
Training examples


online_shopping: learning curves
Training score
Cross-validation score
Training Time
CV Time
Score
Time(s)
Training examples


online_shopping: MCC for n_estimators
Training score
CV score
Score
n_estimators


online_shopping: MCC for learning_rate
Training score
CV score
Score
learning_rate

AdaBoostClassifier training set score for abalone: 0.7909681833732467
AdaBoostClassifier holdout set score for abalone: 0.6267942583732058
AdaBoostClassifier training set score for online_shopping: 0.8160120495886919
AdaBoostClassifier holdout set score for online_shopping: 0.8158961881589619

I quickly noticed similarities between my `adaboost` classifier and my decision tree classifier. My DTClassifier is a weak learner - consistently performing better than random guessing. Both outputted similar model scores to each other - which makes sense, given that I selected my tuned `DTClassifier` as the `base_estimator` for the adaboost algorithm. Yet again, `adaboost` performed better on the online shopping dataset than the abalone dataset. However, what's interesting is that the gap between `adaboost`'s training and test scores for the abalone dataset was much larger than that of `DTClassifier`. It seems that here, adaboost actually magnifies the overfitting issue. Nevertheless, `adaboost` still performs better on the holdout sets - it would've been surprising if it performed worse.

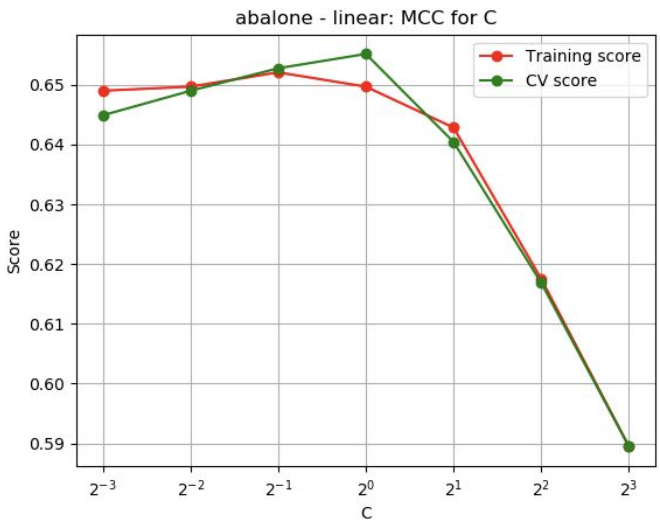
Similar to the decision tree (but unlike the neural network), cross validation was a critical part of training for this algorithm. The large gaps we see between the training scores and cv scores in the model complexity curves indicate that tuning based on cv scores yields a substantial improvement over tuning based on training scores alone. For the abalone data, not using CV would result in significant overfitting - we can tell because of the large gap between the training and cv scores in the learning curve that never really closes (contrast this to the relatively small gap in the learning curve for the shopping data). Looking at the learning curves, we can see that with smaller training sets, we train a model with low bias and high variance (overfit). Eventually, as we increase the training size, we steadily increase the bias and decrease the variance to achieve a more balanced model.

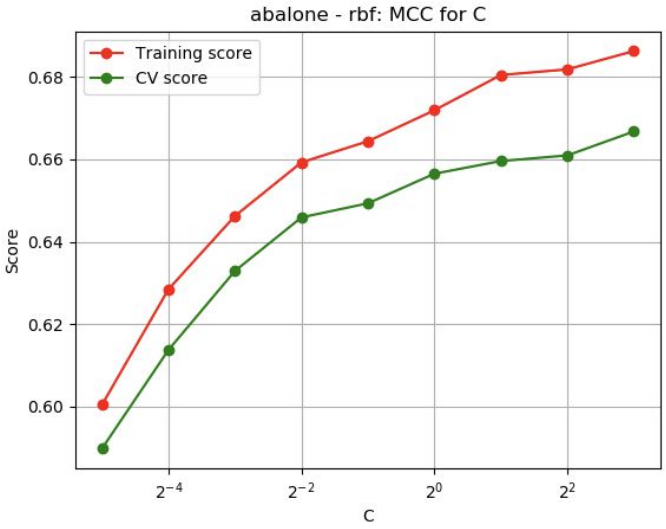
Something interesting to note here is clock speed - yet again, `adaboost` trains faster on the shopping data than the abalone data. This is due to the fact that I tuned the number of decision trees `adaboost` would use before plotting out learning curves. As the online shopping dataset converged on a lower `n_estimators` (5), it will inevitably run faster on each iteration than the abalone dataset, which converged on a much larger `n_estimators` value (40).

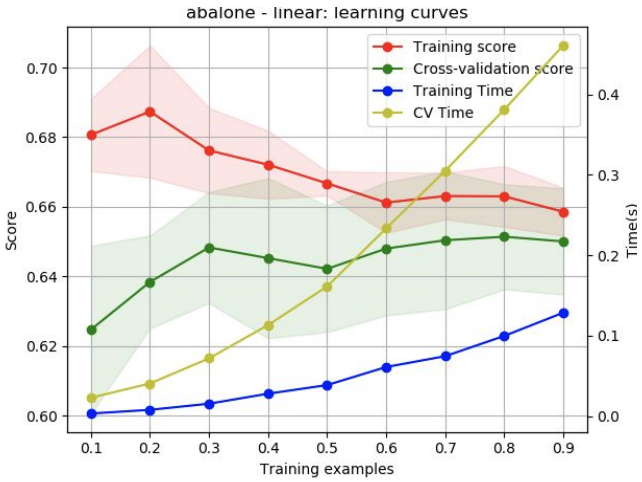
When looking at the effects of individual hyperparameters, we can see that adjusting the learning rate had a bigger effect on the abalone dataset than the shopping dataset. For the abalone data, the complexity curve shows an ideal-ish learning rate around 0.1. The shopping data seems to prefer a much gentler learning rate - if we work under the assumption that this dataset is less noisy, then we don't have to worry about the lower learning rate causing the algorithm to get stuck in a local optimum. It makes sense that we wouldn't want or need to use a high learning rate for either, which might cause the algorithm to bounce around too much and fail to converge.

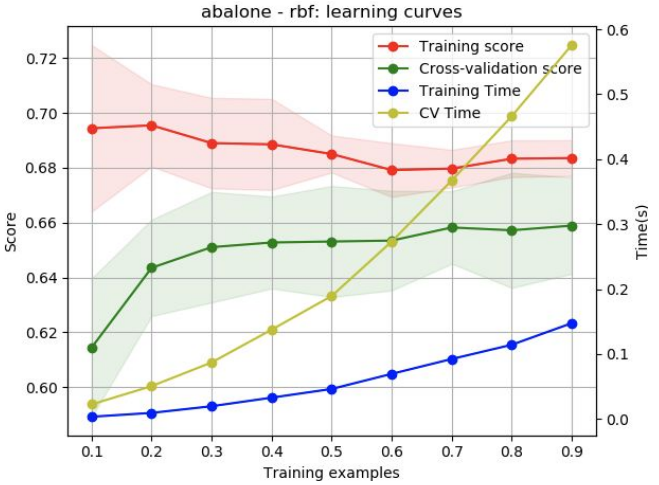
One improvement I would want to make would be to feed in different types of tuned classifiers into `adaboost`, to see which ones would perform well for the datasets we have to work with. I'd be curious to see whether boosting reduces or accentuates the performance differences between different algorithms.

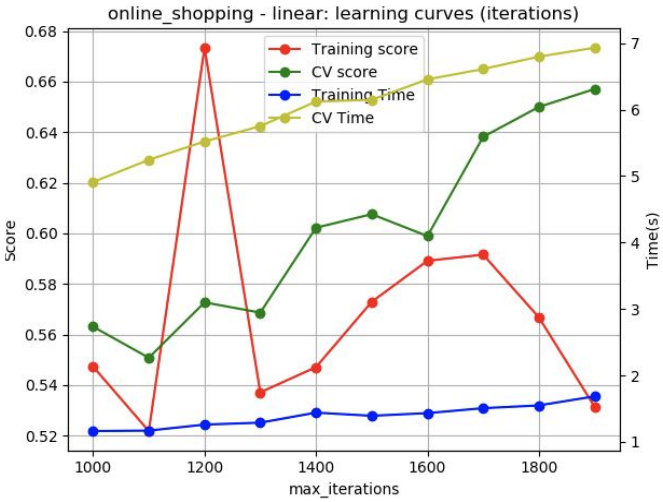
SVM (left side are graphs using linear kernel, right side are graphs using rbf)

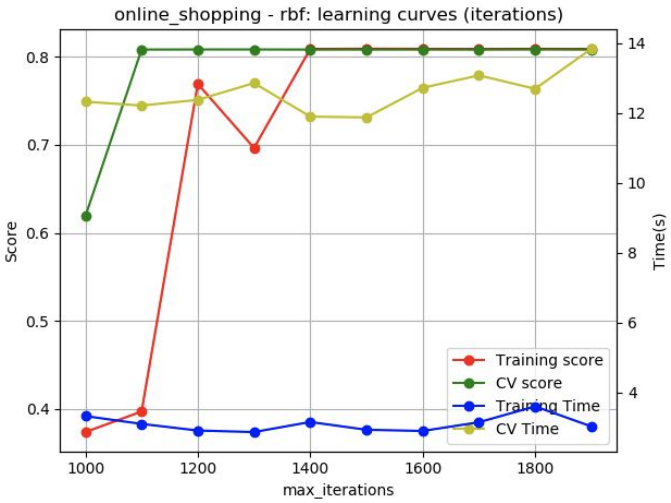












SVC linear (training, holdout) scores for abalone: 0.650, 0.637
SVC rbf (training, holdout) scores for abalone: 0.680, 0.640
SVC linear (training, holdout) scores for online_shopping: 0.810, 0.813
SVC rbf (training, holdout) scores for online_shopping: 0.460, 0.444

Cross validation was not as critical part of training this algorithm as it was for others (like the tree-based algorithms). The small gaps between the training scores and cv scores in some of the model complexity curves, as well as the relative lack of training-cv divergence indicate that tuning based on cv scores yields only a modest improvement over tuning based on training scores alone.

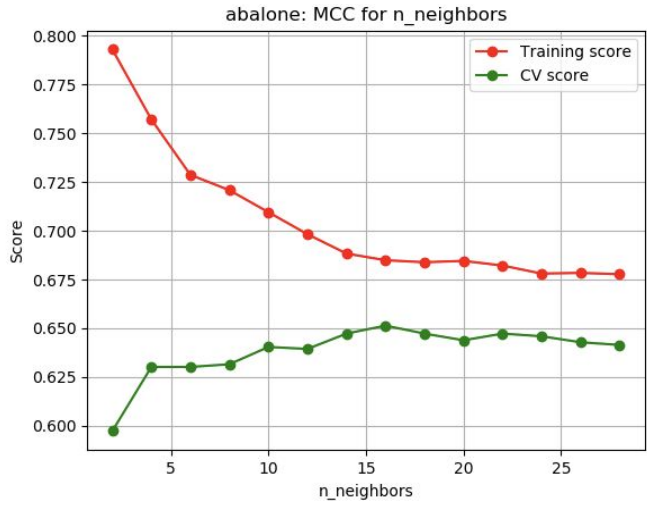
Looking at the effects of individual hyperparameters, it seems that varying the cost parameter - had a large effect on the model's performance on the abalone dataset. For both kernels, I chose the value of C that resulted in the best CV score. For the linear kernel, I set C to 2, and for the rbf kernel, I set C to 8. Interestingly, the effect differs depending on whether we're using the linear kernel or the rbf kernel. For the linear kernel, lower values of C yield better model scores, whereas for the rbf kernel, higher values of C tend to yield a better model score. This suggests that the abalone data is not as linearly separable as the shopping data - which implies a need for a non-linear kernel. When we use the rbf kernel, we can afford to be more strict with C - setting a greater penalty for data points in the margins. This also suggests that the rbf kernel is better-suited for the abalone dataset - which is borne out by the holdout set scores. Again, if we consider the fact that the abalone dataset is more complex, this makes sense. Another interesting result here is the drastic difference between the performance of the two kernels on the shopping dataset - the holdout score is almost double for the linear kernel than the rbf kernel. This suggests that the shopping dataset is much more linearly separable, and thus is well-suited for the linear kernel.

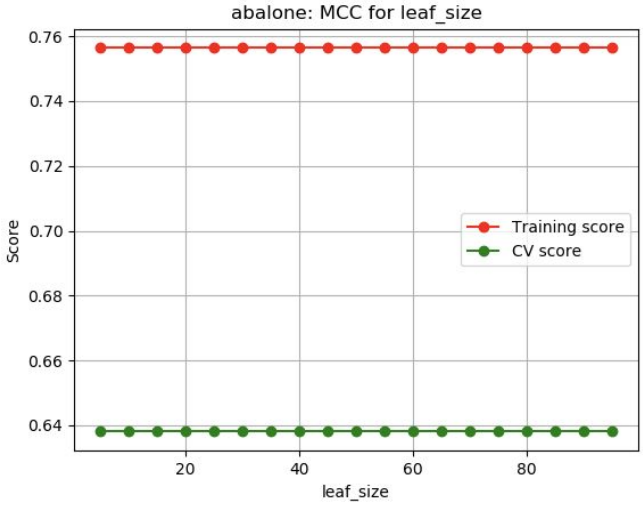
For the abalone data, we see that, as expected, our model suffers from high variance and does not generalize well at lower sample sizes. Using cross validation, unsurprisingly, reduces overfitting and increases generalizability. For the online shopping data, varying the number of training examples did not have a significant effect on model performance, but varying the number of iterations did. Using the same ranges of max_iters, we see that the cv scores for the rbf kernel converge faster than those for the linear kernel. This is an interesting result, especially considering that the model's performance on the holdout set is actually much worse for the rbf kernel than the linear kernel.

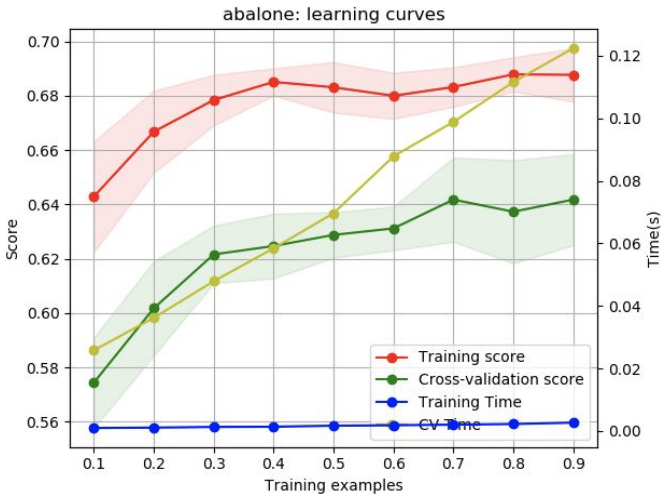
Comparing clock speeds is also worthwhile - it looks like for the shopping data, the linear kernel trains much faster than the rbf kernel. This isn't too surprising - as mentioned above, if the shopping dataset is linearly separable, it makes sense that it would converge faster using a linear kernel vs a radial one. Additionally, it makes sense that SVC took orders of magnitude longer to run on the shopping data vs. the abalone data. The shopping data has a much larger feature space, so transforming those features, especially via the rbf kernel into higher-dimensional space, is considerably more computationally expensive than transforming a smaller feature space.

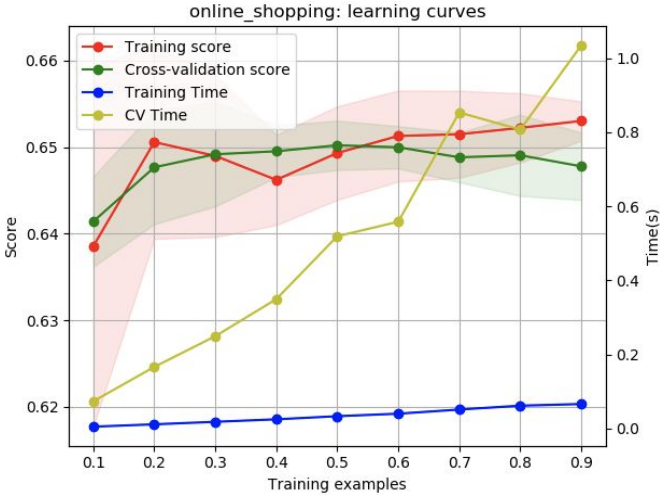
The main improvement I'd want to make is to try and close the gap between the cross-validated scores and the holdout set scores for the shopping data. Perhaps I could better split the training and holdout sets for the shopping data. Or perhaps I'd run a gridsearch for the shopping dataset to see if the gap is due to improper tuning. In any case, this is the biggest potential improvement I could make.

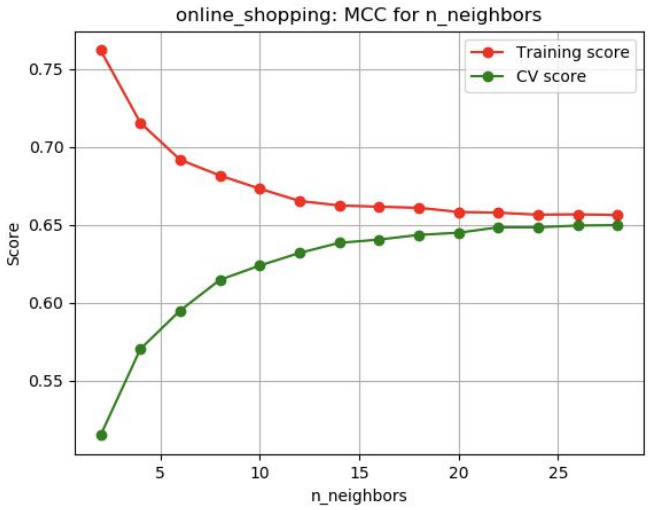
KNN

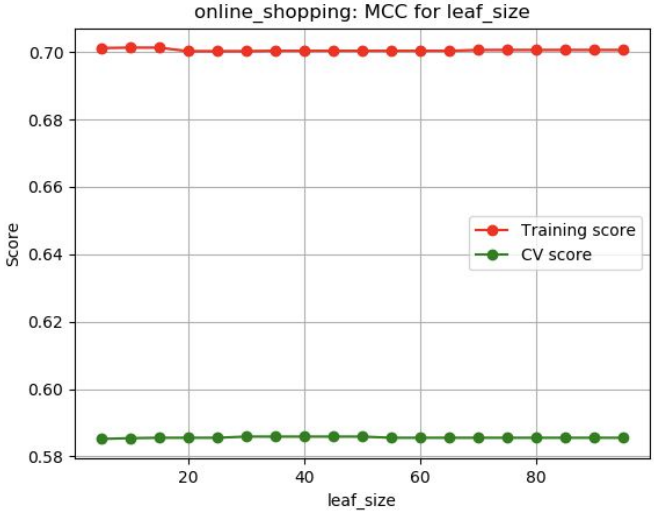

abalone: MCC for n_neighbors
Training score
CV score
Score
n_neighbors


abalone: MCC for leaf_size
Training score
CV score
Score
leaf_size


abalone: learning curves
Training score
Cross-validation score
Training Time
CV Time
Score
Time(s)
Training examples


online_shopping: learning curves
Training score
Cross-validation score
Training Time
CV Time
Score
Time(s)
Training examples


online_shopping: MCC for n_neighbors
Training score
CV score
Score
n_neighbors


online_shopping: MCC for leaf_size
Training score
CV score
Score
leaf_size

KNeighborsClassifier training set score for abalone: 0.6849127608621279
KNeighborsClassifier holdout set score for abalone: 0.6299840510366826
KNeighborsClassifier training set score for online_shopping: 0.6562391379909628
KNeighborsClassifier holdout set score for online_shopping: 0.6715328467153284

For kNN, cross validation was important for the n_neighbors hyperparameter - however, it was almost irrelevant for leaf_size. The sensitivity of the model's performance to n_neighbors makes sense - this is probably the single most impactful hyperparameter that decides how the model converges. For this hyperparameter, both datasets see a convergence as n_neighbors varies - the gap in between the training and cv scores shrinks. This suggests that the ideal number nearest neighbors the algorithm should consider for these datasets is relatively high (around 15). This is doable because both datasets have over 4000 datapoints - thus, there are enough datapoints such that considering 15 nearest neighbors doesn't require too much statistical stretching. We also want to avoid using a number too small, as that would inevitably cause the model to overfit.

For leaf_size, I was curious to see whether it would have any impact on model performance beyond increasing the time required to train. I tried using several different ranges of this parameter to see if I could produce some divergence from the near-straight lines seen in the charts above, to no avail. Sadly, there appears to be very little discernible effect - simply using leaf_size is insufficient to reduce kNN model overfitting.

In the abalone data, our model suffers from high variance and does not generalize well at lower sample sizes. By adding more samples, we add in some bias - forcing the model to make more inferences in order to reduce the variance in our outputs. Interestingly, the online shopping data shows barely any gap between the training and cv scores - the model only starts overfitting more as we use more training samples. It appears that the abalone dataset benefits from having more training samples more than the shopping dataset does. If the abalone dataset is actually more complex, as hypothesized, this would make sense.

Looking at clock speeds, it looks like the shopping dataset took orders of magnitude longer to train. One explanation is the nature of the kNN algorithm - perhaps the biggest determinant of the training time required is the volume of data, and the shopping dataset has over three times the number of training samples as the abalone dataset. As such, at prediction time, it takes the algorithm longer to go through and compute a nearest neighbors calculation for so many more data points. However, as training time should only scale linearly with sample size, the remainder of the discrepancy is probably better-explained by the different hyperparameters chosen for each dataset - for example, the shopping dataset uses 25 nearest neighbors, whereas the abalone dataset only uses 16.

One improvement I would want to make would be to explore different ways of scaling the data to see whether it has a noticeable effect on model performance. Additionally, as with the other algorithms, I would want to run a gridsearch to tune the hyperparameters for the online shopping dataset - I'd want to compare the linear and rbf kernels to see if the performance gap persists after tuning.

Summary

Model performance (based on holdout set performance)

- Abalone dataset
    - SVM, KNN, neural network, boosting, decision tree
- Online shopping dataset
    - Boosting, decision tree, neural network, KNN, SVM

What's interesting here is that these rankings are almost mirror images of each other best performing algorithms on one dataset are the worst on the other! The abalone dataset is more complex - it has fewer features than the shopping dataset, and those features don't necessarily map onto the output variables as closely as in the online shopping dataset. Thus, it's perhaps not shocking that the tree-based algorithms don't perform as well on the abalone data - it may not be as clear which features should be used as the split, and what the split criteria should be set at. Meanwhile, the shopping dataset contains several boolean and categorical features, which may provide better direct signal. Given more time, I would want to confirm this hypothesis by looking at which features the tree algorithms decided to split on with what level of consistency.

Clock time

In order, here is an approximate ranking algorithms from fastest to slowest, measured by wall clock time:

- Decision tree
- kNN
- Boosting
- SVM
- Neural networks

Overall, decision tree-based algorithms (including a DT-powered boosting algorithm) performed on the faster side, whereas the iterative algorithms (SVM and NN) performed the slowest. These rankings are approximate, as there were differences in wall clock time between the two datasets as well. I go into more detail about wall clock times in the analyses above.

Overall

All in all, the algorithm that performed the best was the boosting algorithm - as measured by a combination of consistently high holdout set scores and a relatively low wall clock time. I felt that the "best" performing algorithm should spike on both of these important metrics, as both are critical in evaluating a learning algorithm's real-world usefulness on a particular piece of data.

References

- Decision tree implementation: https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html
- Neural network implementation: https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html
- Boosting implementation: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.AdaBoostClassifier.html
- SVM implementation: https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html
- kNN implementation: https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html
- Plotting learning curves: https://scikit-learn.org/stable/auto_examples/model_selection/plot_learning_curve.html
- Abalone dataset: https://archive.ics.uci.edu/ml/datasets/Abalone
- Online Shoppers Purchasing Intention dataset: https://archive.ics.uci.edu/ml/datasets/Online+Shoppers+Purchasing+Intention+Dataset
- Banknote authentication dataset (referenced in analysis, but not used): https://archive.ics.uci.edu/ml/datasets/banknote+authentication