Abstract

The world is full of problems, which implies there's learning to do. If you've got labels, I've got supervised learning techniques. Specifically, if the problem happens to be identifying the quality of used cars or identifying a non-linear classification with vast amounts of noise added, then the supervised learning algorithms for Decision Trees, Boosting, Artificial Neural Networks, K Nearest Neighbors and Support Vector Machines may be of help. For each of these datasets, and in turn each algorithm, the accuracy of classification was tested under cross validation over a variety of hyperparameters (learning rate, regularization, etc.) using sci-kit learn's GridSearchCV. The resulting hyperparameters, model performance, learning curve's and expressiveness' are examined. This analysis gives way to further insight into both the algorithms and problems.

Datasets

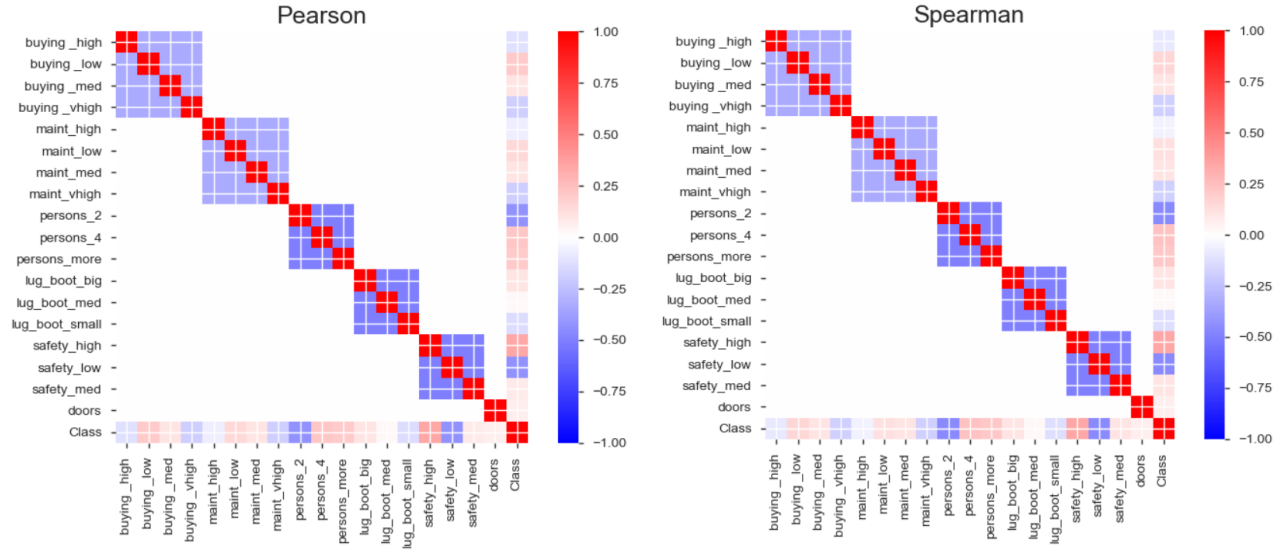
Interesting analysis requires interesting problems, and in order illuminate the strengths, weaknesses and quirks of the examined supervised learning algorithms two well known datasets from the UCI machine learning repository data are examined.

Cars

Instances: 1728 | *Attributes*: 6 (19 when one hot encoded) | *Data Types*: Categorical (6) |
Classes: 0, 1, 2 for Cars with ratings of Unacceptable (0), Acceptable (1), Good/Very Good(2)

This dataset measures the hierarchical characteristics of cars to predict the overall quality/acceptability rating of the car. There are two overarching characteristics measured: price (buying and maintenance) and technical (number of doors, rider capacity, trunk size and safety rating) . Within those six attributes measured, each has multiple ordinal levels. The dataset was originally designed to showcase a simple hierarchical model for decision making. It serves as a test for a learner's ability to recognize structure within the six dimensions. After processing the categorical values to a single dimension for each (e.g. rider capacity processed from one dimension with three levels into three dimensions with one level, each designating a separate condition (capacity-2, capacity-4, capacity5+), the dataset expands to 1728 instances with 19 dimensions. A learner will have to distinguish which of the dimensions are related (e.g. capacity-2, capacity-4, capacity5 are not independent) and model them accordingly. Additionally, the classification problem is imbalanced with the vast majority of cars classified as unacceptable (70%), while 22% are acceptable and the final 8% classified as good/very good. The distribution of each category is balanced between levels (e.g. capacity-2, capacity-4, capacity5 each have 1/3 of the total instances in each level).

There are no dead giveaways relating any of the categoric levels to the classes, as evidence by the correlation matrix. Since the labels are roughly ordinal in that a higher level indicates a higher quality, the Pearson and Spearman correlation should indicate a relationship for a categoric level being associated with an increase or decrease in label:



The bottom row of the figures highlight the relationship between the categoric levels and quality class. Most have a weak relationship (+- .25) at best, which will require the learner to recognize the combinations of the weak relationships in patterns related to determining class level.

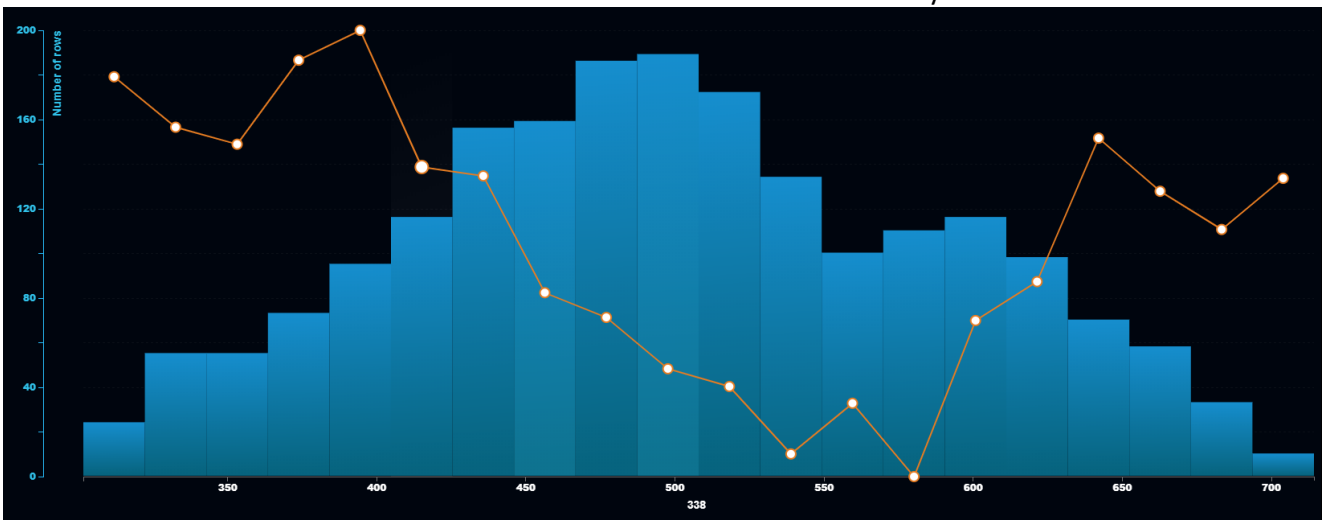
Madelon

Instances: 5000 | *Attributes*: 440 | *Data Types*: Continuous (440) | *Classes*: 0, 1

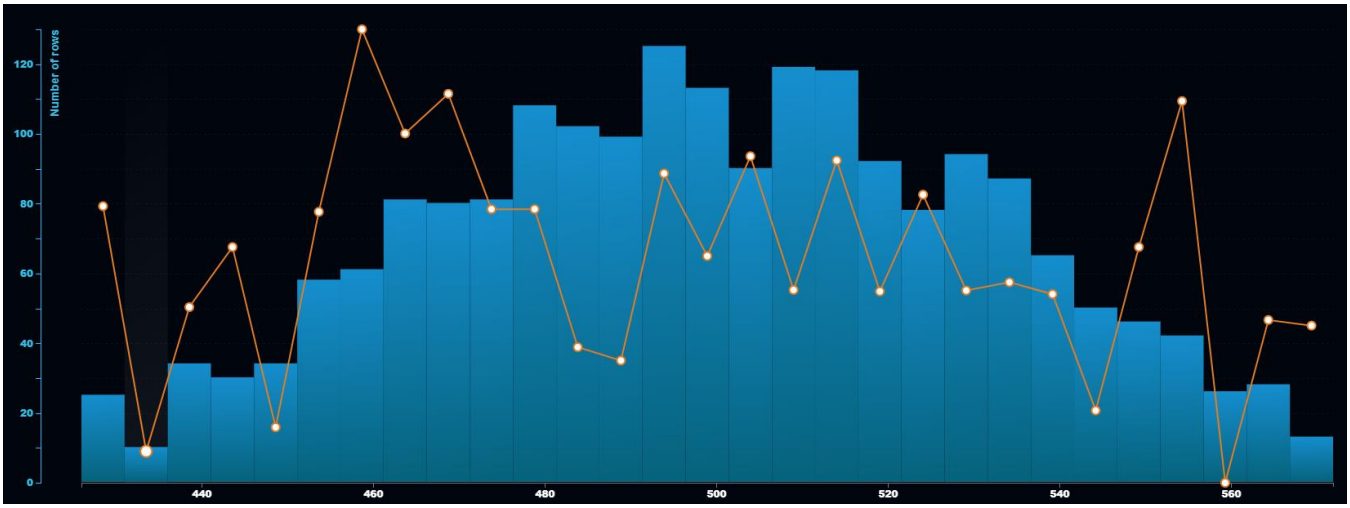
The MADELON dataset is an artificial dataset created in 2003 for the NIPs conference as part of a feature selection challenge. The target class comes from a group of 32 clusters on the vertices of a five dimensional hypercube. Those points were randomly assigned a class (either 1 or -1). Additionally the five dimensions were transformed by linear combinations to form fifteen more features. To complicatedthe problem, 480 features of random noise were added to the dataset.

Of interest here is that the Madelon dataset presents a highly non-linear problem where the signal-to-noise ratio is very low. 1% of the features are truly useful (the 5 dimensions) while 15 (3%) are superfluous albeit still informative. This leaves 96% as completely useless to learn from. To alleviate some of the imbalance in signal-to-noise ratio, sklearn's feature selection method SelectFromModel in tandem with a RandomForestClassifier was implemented. The feature selection was repeated four times with a threshold set to 'median', i.e. any feature deemed to be in the lower half of feature importance is dropped. In other words, the more important half of the features were kept with this repeated four times leaving 31 features for the algorithm to learn from. In the best case scenario, this would leave the 20 informative features and 11 noise features.

These plots reveal the difficult of the non-linearity and noisy features. Below is feature 338 with an orange dot representing the number of classes=1 for its associated bin. There appears to be a consistent pattern, albeit non linear with several inflections as the values increase. This is likely one of the 15 informative features.



Feature 31 appears to have some potential information (especially with peaks around values 450 and 550) but is so noisy throughout that it is highly likely this is just chance and not actual signal.



The learners will have to distinguish between many of these noisy features to properly classify the Madelon dataset. In addition to the noise issues, the non-linearity of the problem presents an interesting challenge to the learning algorithms. Algorithms without the expressiveness to describe non-linear patterns, e.g. a linear SVM, may struggle on the dataset while others, e.g. an SVM with RBF kernel, may perform well.

Algorithms & Methodology

All algorithms were implemented via the python machine learning package sci-kit learn. For each algorithm, the learner was five fold cross validation trained using balanced accuracy as the performance metric across a variety of hyperparameters. Balanced accuracy was chosen in light of the Cars dataset's class imbalance, as without the cost balancing most learners would likely not focus on correctly classifying 'good' cars as they represent only 8% of the sample. The best parameters were stored, with the best performing classifier then trained on varying amounts of the data with its performance and wall clock time recorded to illustrate its learning curve and computation cost. The variance or expressiveness of the iterative/ensemble learners (ANN, Boosting, SVMs) was tested by measuring the train and test accuracy across an increasing number of iterations using hyper parameters with high expressiveness (i.e. regularization parameters set to very low values). All models are presented with their best selected hyperparameters and their respective learning curves and timing curves. If the graphs revealed something interesting, the 'expressiveness' curves were presented as well. All hyper parameter results and subsequent discussion can be cross checked with the result csv files under 'reports/output/{learner}_{dataset}_reg.csv'. The change in hyper parameters across a dimension is discussed but rarely plotted due to space constraints. The files 'reports/output/{learner}_{dataset}_reg_pivot_table.csv' illustrate how scores varied across certain parameters, which is further discussed. All figures for the plots used (plus additional unused graphs) are under 'reports/figures'.

Artificial Neural Network
Hyperparameters Searched: Activation Function, Learning Rate, Hidden Layer Size
Decision Tree
Hyperparameters Searched: Splitting Criteria, Pruning Level
Boosting
Hyperparameters Searched: Number of Estimators, Pruning Level (of base estimator Decision Tree)
K Nearest Neighbors
Hyperparameters Searched: Distance Metric, Number of Neighbors, Weighting of Neighbors Method
SVM
Hyper Parameters Searched: Kernel (Linear, RBF), Regularization Alpha, Number of Iterations, Gamma (RBF)

Results

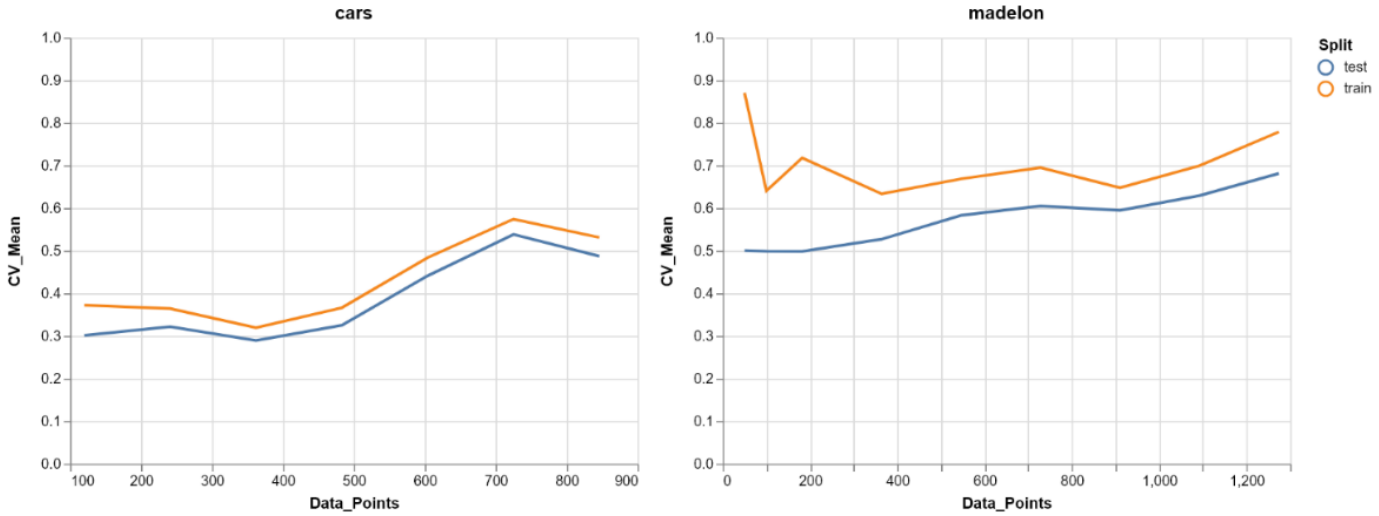
Artificial Neural Network

Dataset	5 Fold CV Score	Activation Function	Learning Rate	Hidden Layer Size
Cars	.5159	Relu	.01	(18)
Madelon	.7628	Relu	.0001	(62, 62, 62)

The ANN was implemented by a single hidden layer perceptron. The ANN performed very poorly on the hierarchical Cars dataset with a CV score roughly .4 lower than the next lowest score. When looking at the activation functions (see reports/output/ANN_Cars_reg.csv for hyperparameter variation and subsequent scores), Relu consistently outperformed Logistic. In fact, anytime the logistic activation function was implemented, the max test score was .33. What I expect is that Relu as an activation function is resulting in a more sparse hidden layer, effectively reducing the features into independent groups (i.e. the learner realizes when capacity-2 is present, the other features capacity-4 and capacity-5+ don't matter and go to 0). The sigmoid activation function likely results in very high and low values in the hidden layer unit and these extreme values cause the learner to reach a local optima on the training data and get stuck at .33 on the test set. The logistic activation may also result in the 'vanishing gradient' problem which is why the learner gets stuck at a score of .33. The hidden layer dimensions and learning rate only make a difference on the relu activated learners. The hidden layer size appears to be more important than the learning rate, as once again all scores are consistent along most learning rates. The optimal size (18,,) once again hints that a simpler representation without the expressiveness of a large hidden layer results in better results.

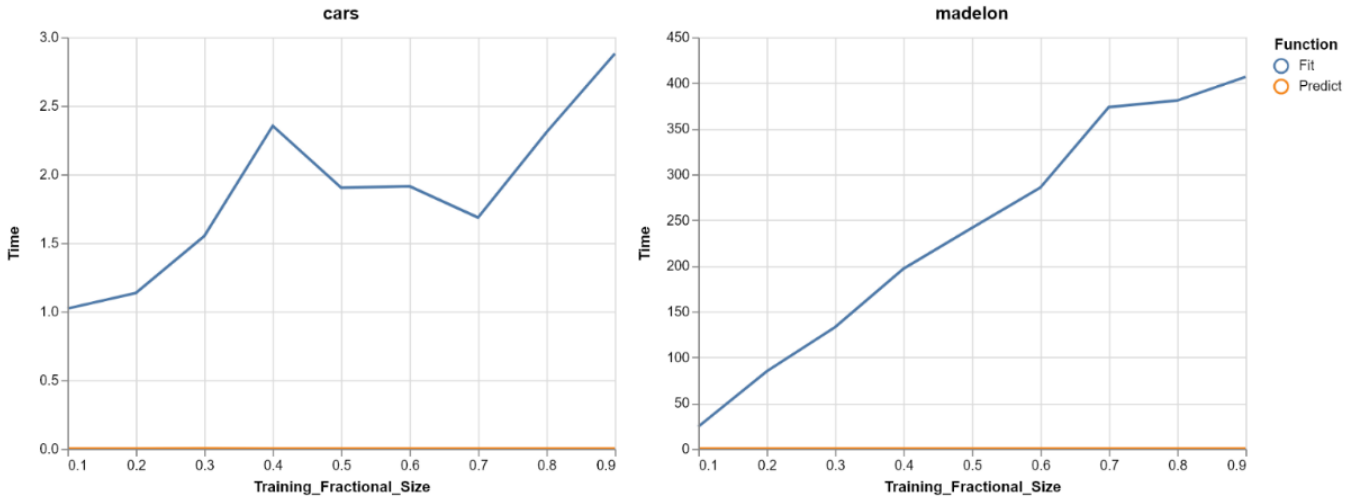
For Madelon, the opposite is true. The best parameters are those with maximum expressiveness (hidden layer size 62, 62, 62) and checking for small changes in the gradient with a tiny learning rate. For all parameters (activation function and learning rates), the test score increased as the hidden layer size did, showing that it was the most important part of tuning. This aligns with the highly nonlinear problem of the Madelon dataset.

Learning Curve



The learning rates are mostly standard, with Madelon showing a converging and ever increasing train/test scores. With more data, it appears the ANN may have eventually properly approximated the true function. The Cars dataset posed more of an issue, with the model learning the structure before dropping slightly at the high end of the learning curve. This could be an aberration or point to the unsuitable nature of an ANN for this type of problem.

Timing Curve



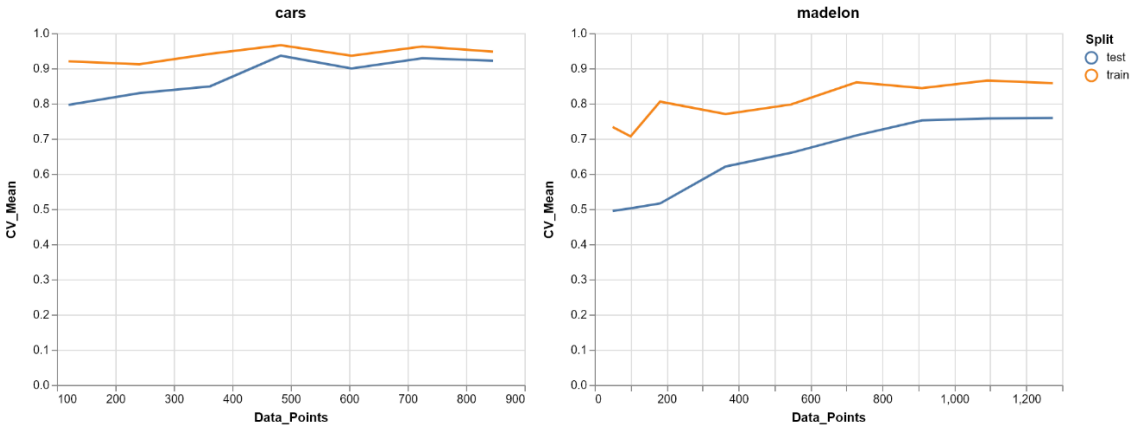
The timing curves reveal a roughly linear increase in computational complexity for the ANN. The Cars set is much smaller and this is reflected in the minimal fit time, however it increases roughly linearly throughout the fraction of the training size. Despite being highly nonlinear, the Madelon set also increases linearly, but its much larger dimension shows the ubcreased computational expense (upper bound of Cars is ~3 seconds vs. 400). The madelon set took a great deal of time compared to other learners, illustrating the huge expressiveness NN are capable of. Although the dimensionality increased the training time greatly, both models predict at roughly the same scale.

Decision Tree

Dataset	5 Fold CV Score	Splitting Criteria	Alpha (Pruning)	Node Count	Class Weight
Cars	.9555	Gini	0	30	Balanced
Madelon	.8102	Gini	.0001	49	Balanced

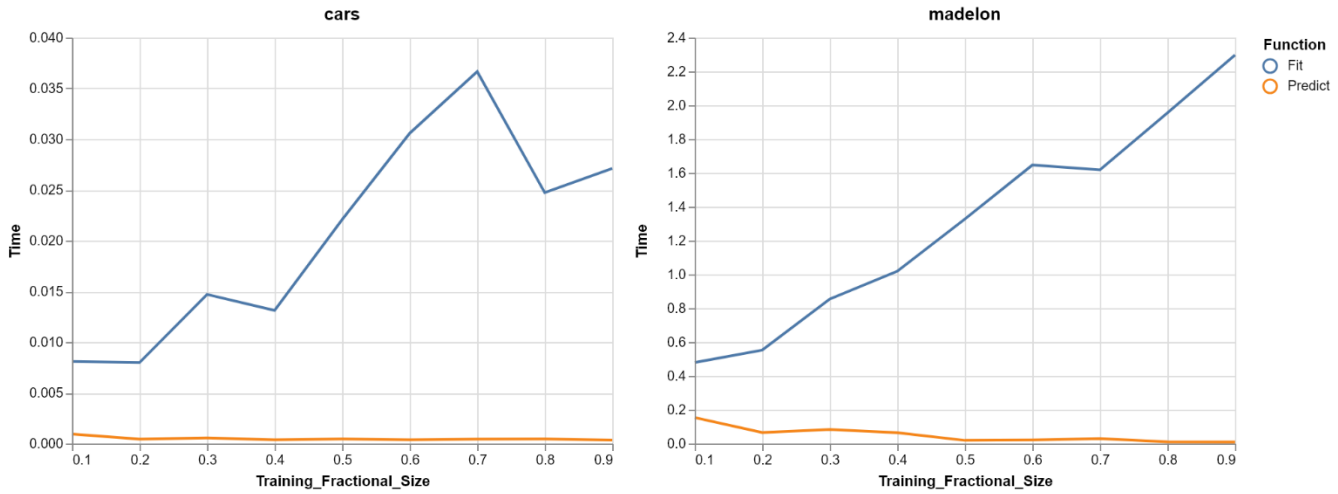
For the decision tree, pruning was implemented at varying levels of alpha. It defaulted to removing nodes that did not drop training accuracy, with higher values more aggressively pruning the tree and low values leaving more decision nodes. The splitting criteria was close to equal among gini and entropy, which makes sense since they are very similar in nature. The alpha level was the key parameter here. As could be expected, the effect of alpha on the test score reflected a bell curve with extreme values resulting in a much lower score. The sweet spot appeared to be anywhere +-.01, indicating that standard pruning was the key factor with a tiny amount of increased pruning preferred with the Madelon set. Even with the increased pruning, the Madelon decision tree had ~1.5x as many decision nodes as the Cars tree. The ratio of nodes to dimensions was larger for the Cars tree however: 19:30 vs 31:49. This reflects the higher level of signal in the Cars set vs the noisy features of Madelon, many of which would not be useful to split on.

Learning Curve



The decision tree learner modeled the car dataset well, which is understandable given that it reflects the interactions between feature levels well while ignoring irrelevant info afterwards. After only 1/3 of the data the decision tree was able to achieve a 90% balanced accuracy test score, a testament to the high signal of the dataset. The Madelon set proved more of a challenge, but it appears to slowly be converging. At first the 'overfitting' gap between the train and test split is rather large until roughly 1/3 of the data is seen.

Timing Curve



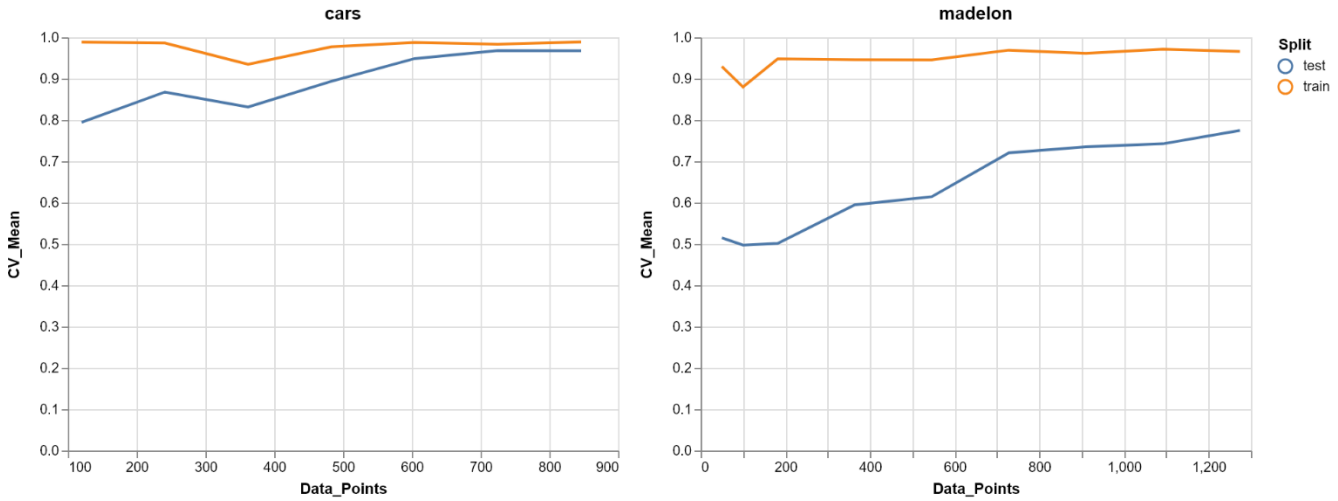
The decision tree trains incredibly quickly compared to some of the more computationally expensive models, fitting the Cars dataset in less than 1/20th of a second and the Madelon dataset in under 2.5 seconds. This was the second fastest of all models tested. It's a bit curious that the prediction time appear to decrease as the size of the data goes up, but the predict time is still so small this could be an anomaly.

Boosting

Dataset	**5 Fold CV Score**	**Base Estimator (Decision Tree) Learning Rate**	**Number of Estimators**
Cars	.9649	-.001	5
Madelon	.8269	.0316	20

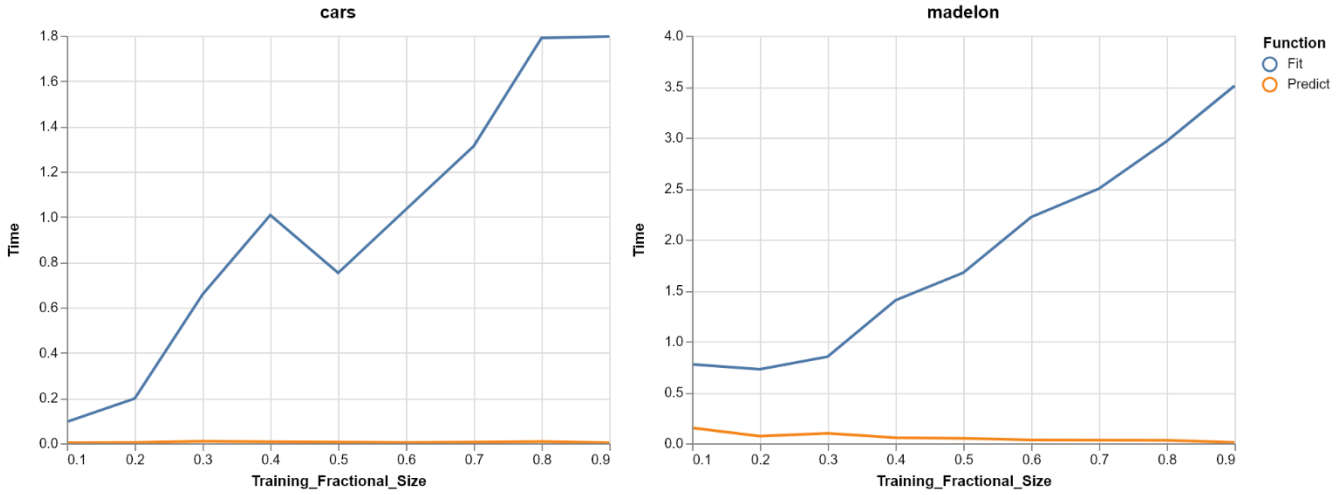
One would assume that if the decision tree performed well, a boosted tree model would do even better. This proved to be the case with both datasets seeing a small improvement here (+.015). The level of pruning for the base estimator decision tree proved to be much more important to test performance than the number of estimators. Scoring across the alpha pruning level varied much more than across the number of estimators. Performance increased slightly (usually around .01) up until about 10 estimators, at which point it negligibly increased or remained constant. This holds to the constraints of boosting, as more estimators are not added unless they qualify as weak learners. Of more importance is the level of pruning applied to the base estimator. Interestingly, the Cars model less aggressively pruned while the Madelon pruned more aggressively, which once again reflects the differing signal to noise ratio in the data. The Cars set with high signal was modelled more effectively with deeper trees while the Madelon set benefited from shorter trees focusing on the select informative features. This is further evidenced in the number of estimators. Cars had more expressive trees (less pruning) but with only 5 trees, while Madelon had less expressive trees (more pruning) but with more total trees —20.

Learning Curve



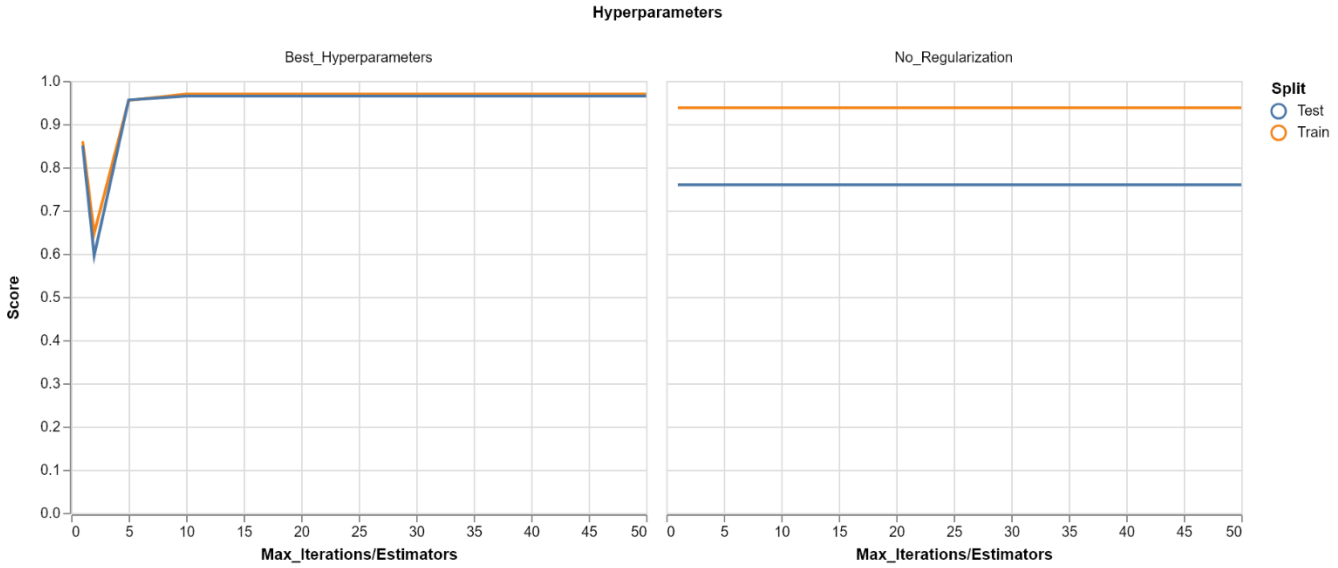
The hyperparameter selections are mirrored by the learning curve. The Cars set trains to a nearly perfect score after about half the data while the Madelon dataset has a large gulf between the train and test sets. The Cars set can be approximated well with little data, reflecting that only a few trees with less pruning can approximate the function well. Madelon on the other hand is a more difficult function to approximate, requiring a larger number of weak learners in ensemble to perform well.

Timing Curve



Both models trained and predict fairly quickly, as the smaller Cars set under 2 seconds while the Madelon set took longer but was still relatively quick (> 4s). The relative time increase for the smaller Cars set was much larger than the Madelon set: while the data increased ten fold across training size, the train time increased 18x which is worst than Madelon's 4x at the lower bound compared to upper bound. Perhaps this has to do with the deeper trees of the Cars model vs the shorter but more for the Madelon model.

'Expressiveness' Curve (Cars)



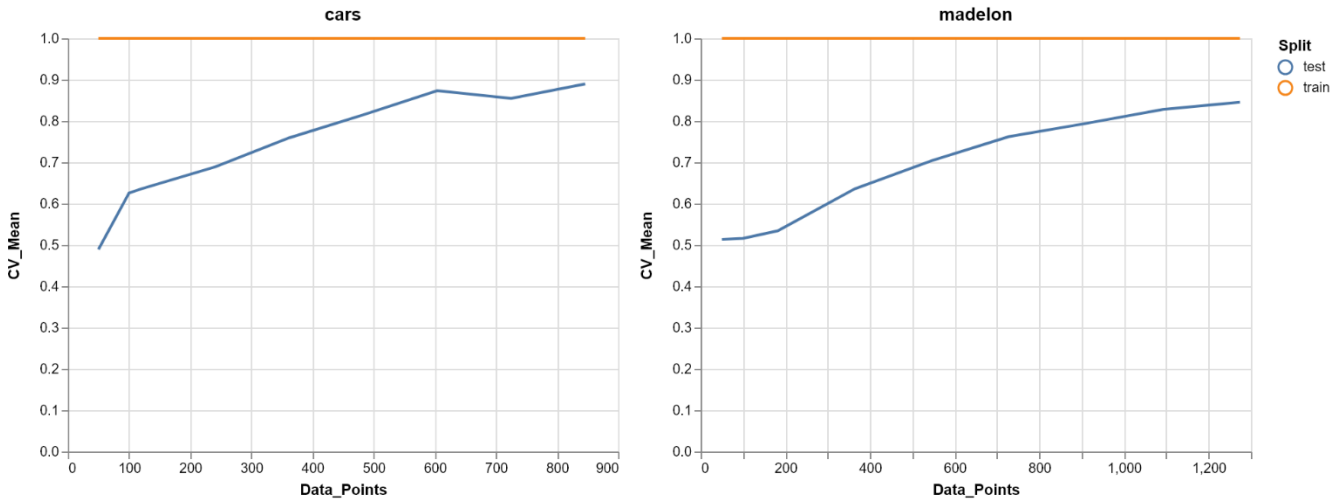
Of interest is the difference between the best hyper parameters for the Cars dataset vs no regularization set. The best hyperparameters appear to reflect the test set nearly perfectly while the non regularization parameters fail to generalize well at all. This illustrates that without regularization the variance of the learner is high while the bias is low.

KNN

Dataset	**5 Fold CV Score**	**Distance Metric**	**Number of Neighbours**	**Neighbours Weight**
Cars	.9540	Manhattan	10	Distance
Madelon	.8641	Manhattan	10	Distance

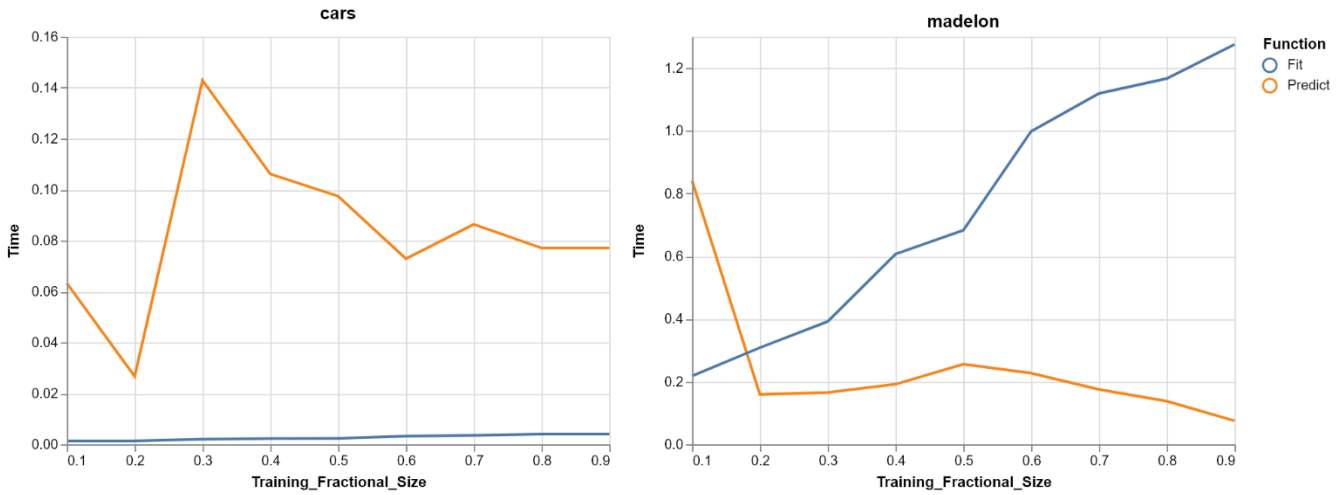
For both datasets, the Chebyshev distance metric performed poorly as did using a uniform neighbours weighting. The key parameters here were the choice between manhattan and Euclidean distance and the number of neighbors to use (K). For the distance metric, in higher dimensions manhattan distance is often preferred due to Euclidean's tendency for one dimension with large values to overwhelm all others when the data is sparse. This is the case for Cars as much of the data is sparse, while Madelon had 30+ features being considered. For the number of neighbors, both learners performed best with K between 7-16, and including more neighbours than this only added uninformative noise even when weighting by distance. The Cars dataset performed similar using Euclidean distance with 7 neighbors and manhattan distance with 10. If one particular feature was more important than all others, this could be used due to the tendency of Euclidean distance to have one dimension overwhelm the others. The Madelon KNN model was the best performing out of all the learners, and this makes perfect sense given the nature of the problem. The classes came from points on a five dimensional hypercube, so it follows that measuring the distance of an instance in ~30 dimensional space and weighting by the distance would result in the best performing model. The solution follows the form of the problem.

Learning Curve



The KNN algorithm is dependent on the quantity and quality of the data for measuring the similarity of instances. Since the quality of the 'neighbors' change with the quantity of the data and the best parameters were learned with the most data these learning curves illustrate a type of future leakage. If the best parameters were found on the smaller datasets (with possibly worst instances of 'neighbours'), it would likely be different and most likely ones with higher bias and lower variance to generalize better. Higher quality neighbors (i.e. all data) allow for better parameters to be found, and these curves illustrate how KNN is highly dependent on finding the right parameters. With the right parameters, the training scores are nearly perfect. Even with these parameters, the model overfits with the gap between train and test scores.

Timing Curve



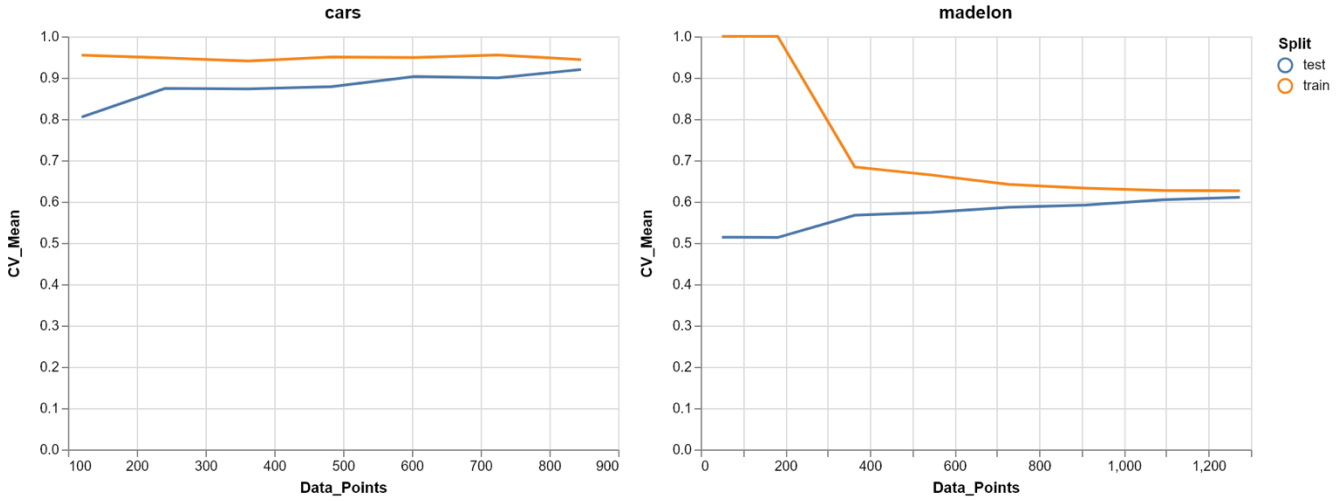
Here the lazy nature of KNN is revealed. This is the only model with predict times larger than fit times (at some levels of training size). In low dimensions (Cars set) this is the case as it takes very little time to compute the distance between the 16 dimensions, but the Madelon set increases in train time linearly as the dataset size increases . This shows that the computational expense of KNN is a function number of dimensions and number of instances. Higher dimensions and neighbours requires computing the distances and therefore has a higher computational expense. This is still a very fast learner, as it had the best speed for Madelon.

SVM (Linear Kernel)

Dataset	5 Fold CV Score	Alpha (Regularization)	Number of Iterations
Cars	.9007	.1	678
Madelon	.6012	.0001	1034

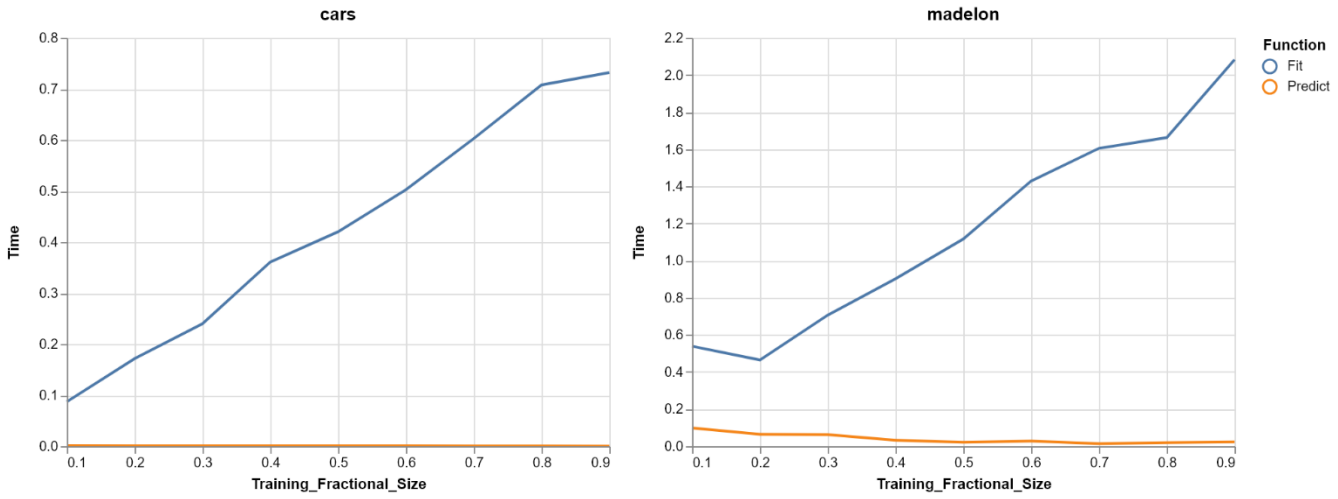
Compared to the other learners, the linear SVM performed poorly, as the 2nd worst on the Cars set and the worst on Madelon. This makes perfect sense since both problems required nonlinear reasoning to approximate the true functions. The SVM learners were implemented using a gradient descent with a hinge loss and L2norm penalty to guarantee convergence if possible (these parameters mimic SVM with the penalty as L2 distance if misclassifying the instance). The parameters selected illustrate the relative complexity of the datasets. Madelon was incredibly nonlinear with many dimensions, reflected by taking 1.5x as many iterations to reach convergence. The alpha regularization term for the SVM Linear contrasts the Boosting model, which had deeper trees (less regularization) for Cars and shallower trees (more regularization) for Madelon.

Learning Curve



It's impressive how well the SVM score on very little data from Cars, achieving .8 with only 100 data points. This would suggest the dataset has high signal to noise rather, which is the opposite of the Madelon set that shows massive amounts of overfitting with small amounts of data before converging near a score of .6

Timing Curve



The SVM models far and away took the longest to train. The fit time increased linearly with data but had decent scaling to larger data with Madelon. The time to fit was ~1 second for Cars and ~2 for Madelon, which is reasonable until the matter of iterations comes into play. For Cars the optimal iterations was >600 which resulted in a train time of over 5 minutes while Madelon was far worse with over 1000 iterations and a total train time of ~35 minutes. This far exceeds any other algorithm, but is in line with the nature of quadratic programming.

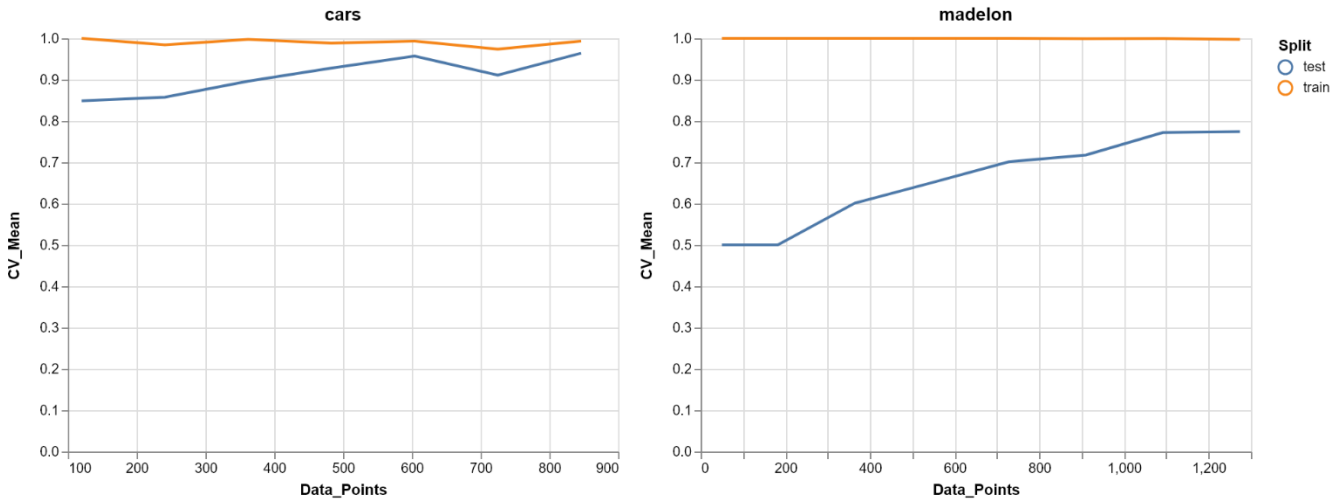
SVM (RBF Kernel)

Dataset	5 Fold CV Score	Alpha (Regularization)	Number of Iterations	Gamma
Cars	.9856	.0001	1034	.8

Madelon	.8371	0.00031622776601683794	687	.15

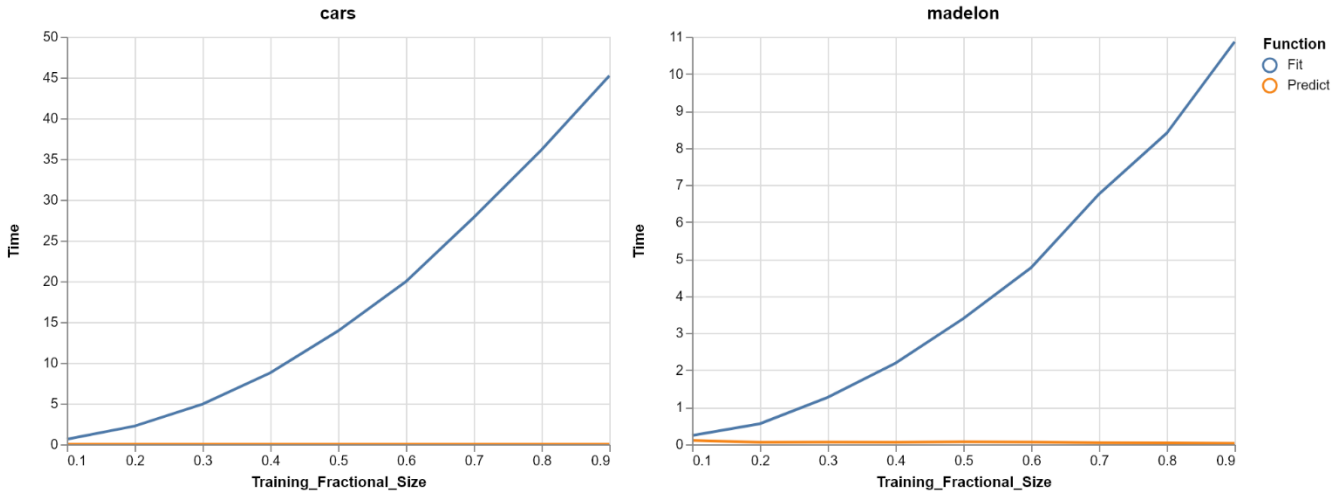
This SVM performed well on both sets due to its strength in approximating nonlinear relationships by projecting into higher dimensions using the RBF kernel. The RBF kernel added the gamma parameter which can control the influence of instances on the support vectors based on their distance in the RBF projected space. A low gamma value indicates points that are farther away can still influence the support vector while a higher gamma means the instances closer in the projected space influence the support vector but distant instances do not. This is reflected in the datasets, where Cars has a high gamma which indicates other similar Cars determine the support vector more than dissimilar Cars. Madelon has a low gamma which indicates that all points can influence the support vector, which makes sense considering the classes were generated from points on a five dimensional hypercube and randomly assigned a class. The regularization terms are both low, likely as a tradeoff with how much gamma determines the influence of instances in the projected space. The number of iterations has switched from the linear SVM, with Cars requiring more iterations than Madelon.

Learning Curve



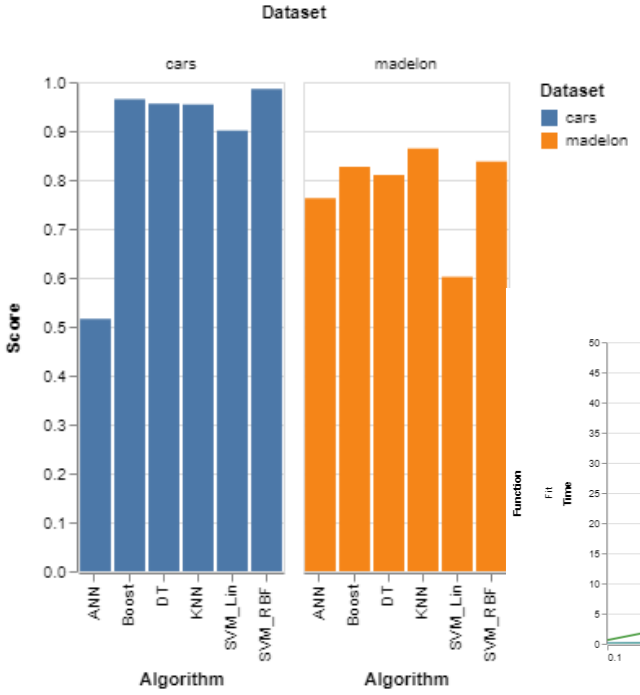
The learning curves appear similar to the linear SVM, albeit with better performance. Madelon in particular is interesting as instead of converging the learning curve appears to have a large gulf between the train test split. Before the linear SVM could not score well on the train set, while here it can overfit to maximum score. The test score has a positive slope, which indicates with enough data the learner could continue to improve its performance and perhaps eventually perfectly model the function used to create the classes.

Timing Curve



The general shape imitates the linear SVM, but the RBF kernel adds a large computational expense: ~ 50x for Cars and 11x for Madelon. Perhaps the sparsity of the Cars dataset does not play well with the RBF curve, as Madelon (a dense dataset) had a much smaller relative time increase. This learner was far and away the slowest of all the algorithms.

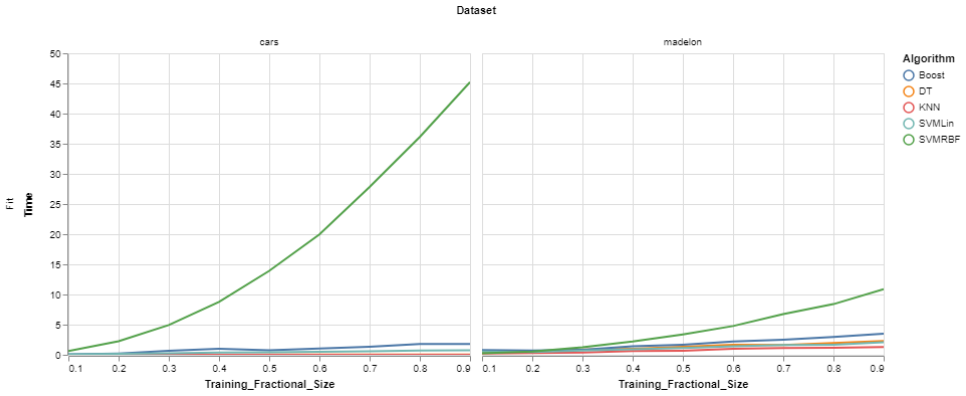
Concluding Remarks



Left: 5 Fold CV Score

Bottom: Training Fit Times

(ANN not included as it skewed the y axis as to make comparison visually irrelevant)



The Cars and Madelon datasets provided contrasting problems for the supervised learners examined: one a small, high signal, hierarchical and sparse dataset (Cars) with the other a large (5000x500), high noise, non-linear dense dataset. The SVM with an RBF kernel performed the best overall with the highest score on Madelon and 2nd highest on Cars. Boosting did well as an ensemble method scoring 2nd best Cars and 3rd best on Madelon. The KNN proved to work best with the Madelon scoring a .86 overall (1st) but struggled on the Cars set (4th).

These problems and subsequent analysis further hammer home the necessity of aligning the problem space with the strengths of a learner and careful hyperparameter tuning to aid the learner with domain knowledge. Each of these learners possesses contrasting strengths and weaknesses- *this is not a bad thing* but rather an opportunity to craft a tailored solution to any problem. One size fits all solutions are never optimal, and such is the case here as evidenced by the datasets and the problems they pose. Before any model fitting takes place, it pays dividends to diagnose the unique aspects of the problem space and apply learners best suited to find a solution.

Take the Cars and Madelon dataset as examples. It would make the most sense that a hierarchical problem would lend itself well to a decision tree. Lo and behold a boosted trees model scored very well on it, and with tuning performed very well with minimal computational expense. Similarly, for the Madelon data that was artificially created using points on a 5 dimensional hypercube was best approximated by KNN. KNN classified instances based on a weighted similarity score that came from a distance measure that lends itself well to high dimensions (manhatttan). The RBF SVM outperformed its linear counterpart due to the nonlinear nature of both datasets and its ability to project into higher dimensions.

Of course there are always more considerations when searching for the optimal solution, such as computational complexity. Although the RBF SVM performed best on the Cars dataset, it requires much more computation and time for convergence. If the use case involved massive amounts of data it would make far more sense to use a fast method such as a decision tree or boosted trees method which lends well to scale.

Machines learning models are fast, accurate, and stupid. This of course is the case without careful analysis and thoughtful application, as this paper's analysis illustrated the core importance of proper algorithm application and tuning involved in solving supervised learning problems.