Analysis and Comparison of Supervised Learning Algorithms

Yunhan Zou

Submitted: February 10, 2019

1 Introduction

In this report, I will present 5 supervised learning algorithms: decision trees with pruning, adaptive boosting with decision trees, neural networks, support vector machines, and k-nearest neighbors. Each algorithm was implemented using Scikit-Learn, a machine learning library for Python and run in Python 2.7. For the purpose of analysis and comparison, the learning curve for each algorithm was drawn. Train/test splits were used for testing and hyperparameter tuning. Data set was divided into training set (0.75) and test set (0.25) randomly, and the test accuracy was obtained by averaging the test accuracy of 10 runs of train/test splits.

2 Classification Problems

Data lie at the heart of machine learning problems. For the purpose of this assignment, the two classification problems I chose are the car data set, and the pen digits data set.

2.1 Car data set

The car data set was derived from a simple hierarchical decision model that evaluates cars with the following attributes: purchase price, maintenance price, number of doors, capacity, size of luggage boot, and estimated safety.

There are 1728 instances in the data set. The 6 attributes above all have categorical values. There are 4 possible classes: unacceptable, acceptable, good and very good. I chose this data set because with supervised learning model, it can help more people make purchase decision based on the criteria described above. In addition, all attribute values are categorical and discrete. To make supervised learning algorithms work, certain type of pre-processing and encoding of data is necessary.

2.2 Pen digits data set

The pen digit data set was created by collecting 250 samples from 44 writers. The writers were asked to write 250 digits between 0 and 9 in random order on a pressure sensitive tablet. The features were derived from pen trajectories arising from handwritten digits. 10992 instances are present in the data set. There are 16 features, each of which is continuous in the range between 0 and 100. Features consist of (x, y) coordinate pair corresponding to pen trajectory. 8 such pairs were selected as feature vectors. There are 10 possible classes, indicating the written digit between 0 and 9. This data set was chosen mainly because unlike car data set, the pen digit data set enable me to test the performance of different learning algorithms on continuous feature vectors. In addition, the motivation for using this data set is to be able to classify handwritten digit based on pen trajectory.

3 Decision Tree

Decision tree is one of the most powerful and useful tools for classification. Decision trees classify instances by sorting them down the tree from the root to some leaf node, which provides the classification of the instance. DecisionTreeClassifer from Scikit-Learn library was used to implement my decision tree. There are 2 hyperparameters worth discussing: splitting metrics and max depth of the tree for pruning.

3.1 Gini impurity vs Entropy

There are 2 metrics to split an attribute: information gain and Gini impurity. They both measure how heterogeneous the data is distributed over a subset. To compare their performance, the decision tree was built with each metric. The pruning was not implemented at this stage, and will be discussed later. Their accuracy on the test set is shown below, which is obtained by averaging the test accuracy of 10 runs of train/test split:

	Gini	Entropy
Car	0.973	0.978
Pen digits	0.96	0.963

Table 1: Comparison of Gini impurity and entropy

It can be seen that without pruning, the decision tree achieves similar test accuracy on both data sets, regardless of splitting metrics. Theoretically, in most cases, their performance on classification should be similar. Because the calculation of entropy is logarithmic and thus is slower to compute, in the analysis that follows, the Gini impurity will be used as the splitting metric.

3.2 Pruning

The pre-pruning was implemented by limiting the maximum depth of the tree. Table 2 shows that as the maximum depth increases, the test accuracy rises. It's interesting to notice that in the end, the test accuracy is even slightly higher than that of the tree with no pruning applied. This is an evidence that pre-pruning also helps resist the overfitting. Figure 1 and Figure 2 are visualization of decision tree with and without pruning on the car data set generated using export_graphviz. It's obvious to note the decrease in the number of nodes present in the tree with pruning.

Max depth	3	6	9	12	15	No pruning
Car	0.764	0.882	0.956	0.975	0.976	0.973
Pen digits	0.598	0.897	0.959	0.961	0.963	0.96

Table 2: Classification rate vs. max depth for decision tree

Table 3 shows that as the maximum depth increases, the number of nodes in the tree increases as well and eventually will converge to that of the tree with no pruning.

Max depth	3	6	9	12	15	No pruning
Car	9	39	109	161	171	173
Pen digits	15	111	345	535	587	597

Table 3: Number of nodes vs. max depth for decision tree

With the help of 10 runs of train/test split, the max depth was set to 10 for the rest of the discussion.

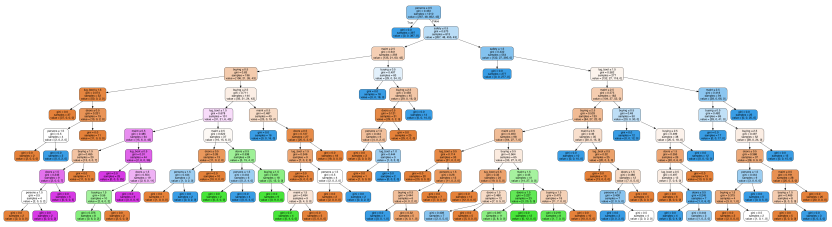


Figure 1: Decision tree visualization with pruning (max_depth = 10)

3.3 Learning curve

Figure 4(a) shows the learning curve for the decision tree. It shows how training and testing errors change as the training size increases. 25% of the data set was reserved as the test set. The size of the actual training set varies between 5% to 100% of the other 75% of the data set. When the size of training

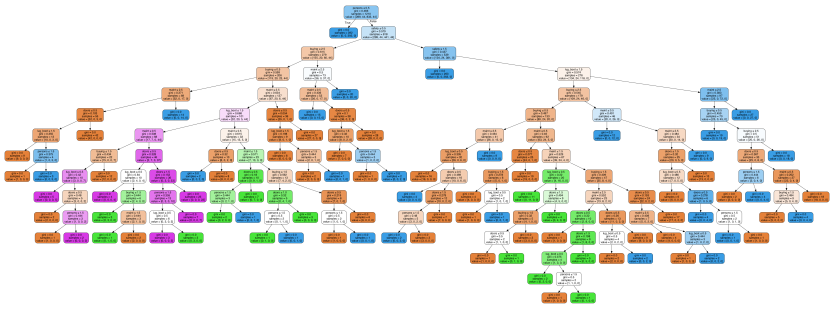


Figure 2: Decision tree visualization without pruning

data is small, the training error is negligible because data is rather simple to fit while the testing error is significant because of underfitting. However, as the size of training data increases, the training error starts to increase while the testing error starts to fall. This happens because more training data result in more complexity in the model, which can perform better for unseen data. Due to space limitation, the learning curve for the pen digits data set is not shown but should display the same information.

4 Boosting

I used the AdaBoostClassifier from Scikit-Learn library to implement my boosted version of decision tree. There are 2 hyperparameters worth discussing here and they are the max depth for pruning the decision tree, and the number of estimators.

4.1 Max depth for decision tree

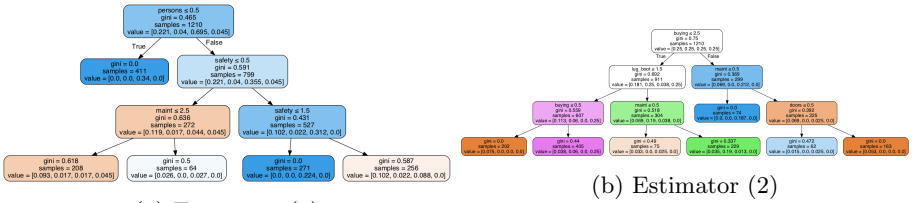
To test different maximum depth for pruning decision tree, the boosting algorithm was run with number of estimators set to its default value (= 50) on both data sets. 10 runs of test/train splits were used to generate the table below showing the test accuracy at different values of maximum depth,

Max depth	1	2	3	4	5	6	7	8
Car	0.771	0.902	0.924	0.933	0.957	0.975	0.98	0.981
Pen digits	0.335	0.689	0.879	0.959	0.979	0.989	0.992	0.993

Table 4: Test accuracy vs. max depth for base estimator

Compared with the decision tree algorithm, it's clear to notice that with boosting, the decision tree can be more aggressive in pruning because the boosting algorithm is an ensemble algorithm that combines a series of low performing classifiers to form an overall improved classifier, and thus should be more powerful than a single decision tree classifier.

Figure 3 are the visualization of 2 estimators on the car data set when maximum depth is 3. It's clear that they represent different simple rules. The final estimator is formed by combining those 50 estimators with weights.



(a) Estimator (1)

(b) Estimator (2)

Figure 3: Visualization of 2 base estimators on car data set with max depth = 3

4.2 Number of estimators

For the following discussion, a maximum depth of 4 will be used in pruning the decision tree on the car and pen digits data set, respectively, because it doesn't seem to either overfit or underfit the training data from section 4.1.

Table 5 shows the test accuracy on two data sets as number of estimators increases from 5 to 200. As the number of estimators increases, the final estimator tends to achieve a higher test accuracy because as N increases, more simple rules can be generated and combined to result in a more complex estimator that can classify those "hard" samples with higher accuracy than simple estimators.

N	5	10	15	20	30	40	50	100	200
Car	0.898	0.92	0.915	0.931	0.926	0.933	0.929	0.928	0.884
Pen digits	0.751	0.822	0.843	0.884	0.9	0.911	0.922	0.951	0.867

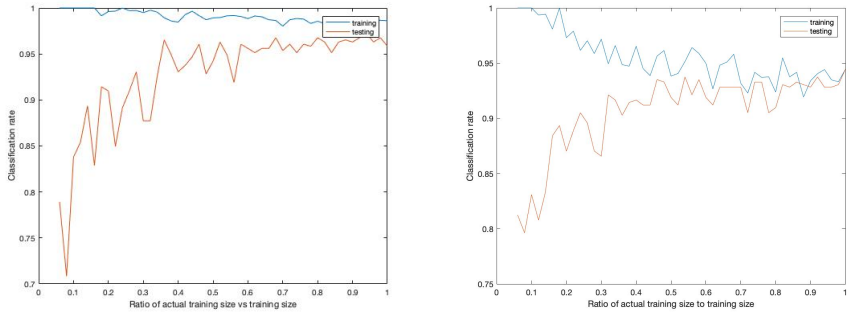
Table 5: Test accuracy vs. number of estimators (N) for boosting

It's also worth mentioning that as N becomes too large, the model tends to overfit the training set and doesn't perform as well on the test set as smaller N. This problem can be alleviated by decreasing the learning rate, as lower learning rate forces the model to learn longer before overfitting, or by pruning the base estimators even more so that it takes more base estimators before overfitting.

4.3 Learning curve

For the following discussion, the number of estimators of 50 will be used on the car data set. This hyperparameter was tuned by averaging 10 runs of train/test split as discussed in the previous section. Figure 4(b) shows the learning curve of the boosted version of decision tree. The x axis denotes the size ratio of the training set actually used to the entire training set. When

the size of training set is small, the training error is negligible because the model is relatively simple to fit small set of data, while the testing error is significant because of underfitting. However, as the size of training set increases, the training error starts to increase while the testing error starts to fall. Due to space limitation, the learning curve for the pen digits data set is not shown here but should display the same information.



(a) Decision tree (b) Boosted version of decision tree

Figure 4: Learning curve for decision tree and boosted version of decision tree

5 Neural Networks

Neural networks are a set of algorithms, modeled loosely after the human brain, that are designed to recognize patterns. The MLPClassifier in Scikit-Learn was used to implement the algorithm. The inner working of neural network is rather complex to analyze. There are 3 major hyperparameters that contribute to this complexity: number of hidden layers, number of neurons on each layer, and maximum number of iterations. In the following subsections, I'll discuss their difference and similarities.

5.1 Model complexity

Table 6 shows the test accuracy of the neural network with 1 hidden layer of 1, 3, 6, 10, 20 and 30 neurons on two data sets, with maximum iteration set to default value (= 200). The test accuracy was obtained by averaging the test accuracy of 10 runs of train/test splits on each data set. It clearly shows that as the number of neurons in the hidden layer increases, more complex model can be learned and thus result in a higher test accuracy. However, one can notice that after 10 neurons, the classification rate doesn't increase significantly by increasing the number of neurons. There is a trade-off between complexity and accuracy. We want our model to learn well while at the same time not being too complex.

Number of neurons	1	3	6	10	20	30
Car	0.696	0.712	0.762	0.796	0.876	0.887
Pen digits	0.368	0.799	0.98	0.98	0.99	0.993

Table 6: Test accuracy vs. number of neurons in a neural net with 1 hidden layer

Table 7 shows the classification rate with 1, 2, 4, 10, 15, and 20 hidden layers, each with 10 neurons, with maximum iteration set to its default value (= 200). It's quite apparent from the table that as the number of layers exceeds 4, the propensity of the model to overfit the training data increases. We never want to design a model that's too complex because of the risk of overfitting training data while not performing as well on unseen data. Based on the observation, our neural network achieves high test accuracy without overfitting with 2, and 4 hidden layers, each with 10 neurons, on the car and pen digits data set, respectively.

Number of layers	1	2	4	10	15	20
Car	0.797	0.902	0.957	0.909	0.877	0.879
Pen digits	0.983	0.984	0.981	0.969	0.942	0.914

Table 7: Test accuracy vs. number of layers in a neural net

Table 8 shows the test accuracy with 10, 50, 100, 200, 500, and 1000 iterations of forward/backward propagation. It is true that with more iterations, the model can obtain a more complex structure. However, on the other hand, the computation becomes more expensive. At the same time, when the number of iterations exceed 200, the model doesn't achieve a significantly higher test accuracy. For the following discussion, the maximum iteration will thus be set to 200.

Number of iterations	10	50	100	200	500	1000
Car	0.619	0.751	0.866	0.944	0.968	0.965
Pen digits	0.844	0.971	0.98	0.983	0.986	0.984

Table 8: Test accuracy vs. number of iterations

5.2 Learning curve

The Figure 5(b) shows the learning curve of the neural network with 2 hidden layers, each of which has 10 neurons, maximum iteration of 200, run on the pen digits data set. It's interesting to note that unlike the decision tree or the boosted decision tree, as the training size increases, the training accuracy doesn't drop noticeably. This is partly because the neural network is able to

learn a rather complex model while at the same time doesn't overfit the training data. Their ability to learn by instance makes them very flexible and powerful.

6 Support Vector Machine

Support vector machine tries to find the best hyperplane to separate the different classes by maximizing the distance between sample points and hyperplane. The algorithm was implemented using SVC from the Scikit-Learn library. There are 2 hyperparameters that are worth discussing: kernel and gamma.

6.1 Kernel

Kernel parameter selects the type of hyperplane used to separate the data. Two types of kernel are being compared here: linear kernel, and Gaussian RBF kernel which is a non-linear kernel.

Table 9 shows the comparison of running two types of kernels on each of the data set, with every other parameters set to default values. Again, the accuracy was obtained by averaging the test accuracy of 10 runs using train/test splits.

Kernel	Linear	Gaussian RBF
Car	0.745	0.894
Pen digits	0.888	0.996

Table 9: Test accuracy vs. kernels

It's clear to see that on both data sets, Gaussian RBF kernel achieves a much higher accuracy. Both data sets have more than two feature vectors, and the result suggests that the training instances are not linearly separable. However, if two kernels work equally well on a data set, then the linear kernel should be preferred, because the linear kernel is a parametric model, but RBF isn't. In addition, the complexity of the RBF kernel grows with the size of the training set. Furthermore, more hyperparameters in RBF kernel need to be tuned, so the model selection is more expensive.

6.2 Gamma

In this subsection, I'll discuss the hyperparameter gamma: a parameter for non-linear hyperplanes. The higher the gamma value, more propensity there is for the model to exactly fit the data. Table 10 shows the comparison of running Gaussian RBF kernel with different values of gamma on two data sets.

It's clear to see that with higher value of gamma, the model has a higher tendency of overfitting the training data, resulting in lower test accuracy. High value of gamma means higher complexity of the model. For the following discussion using RBF kernel, gamma will be set to 1/n_features.

Gamma	0.1	0.5	1	5	10
Car	0.856	0.706	0.951	0.706	0.697
Pen digits	0.996	0.701	0.973	0.701	0.448

Table 10: Test accuracy vs. gamma

6.3 C

C is the penalty parameter of the error term. It controls the trade-off between smooth decision boundary and classifying the training points correctly. Thus, similar to the ridge parameter in ridge regression, C is a regularization parameter that requires careful tuning to avoid overfitting. Table 11 shows the comparison of running Gaussian RBF kernel with different values of C on two data sets.

Gamma	0.01	0.1	0.5	1	5	10	100
Car	0.676	0.715	0.817	0.905	0.979	0.981	0.984
Pen digits	0.9	0.983	0.992	0.994	0.995	0.997	0.996

Table 11: Test accuracy vs. C

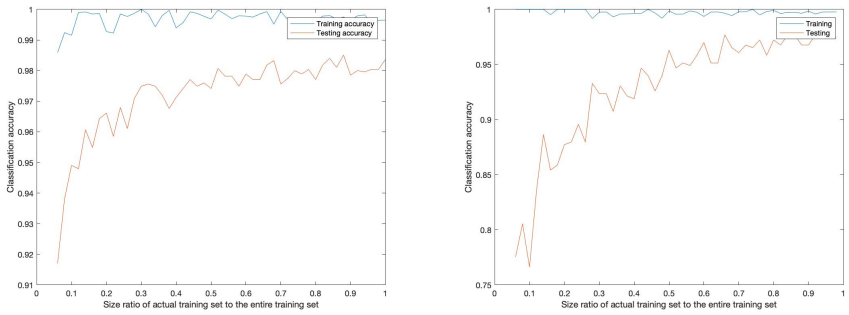
One can tell that increasing the value of C increases the test accuracy because theoretically larger C tells the model to try to minimize its misclassification of training data, resulting in a more complex model. A trade-off has to be made when picking this value. For the rest of the discussion, C will have a value of 10 and 5 for the car and pen digits data set, respectively.

6.4 Learning curve

The Figure 5(a) shows the learning curve of SVM using Gaussian RBF kernel, trained and tested on the car data set. Again, similar to aforementioned neural network, as the training size increases, the training accuracy keeps at a high level and doesn't vary by much, while the test accuracy increases as model becomes more complex with more training data.

7 K-Nearest Neighbors

k-NN is a non-parametric, lazy learning algorithm that doesn't make any assumption on the underlying data distribution. In other words, the model structure is determined from data. My k-NN algorithm was implemented using KNeighborsClassifier from the Scikit-Learn library. I will discuss two hyper-parameters in the following section: K, the number of nearest neighbors to compare with, and weights, the weight function used in prediction.



(a) Neural network (b) SVM

Figure 5: Learning curve for neural network and SVM

7.1 K

The Table 12 shows the test accuracy with different values of K on two data sets. When $K = 1$, we have the most complex model because the model tries too hard to classify, and as a result, the ourliers or noise affect the decision process. Similarly, when K is small, the model tends to overfit the training data with low bias but high variance. On the other hand, as K increases, the model becomes less complex, and it starts to underfit the data with high bias but low variance. A good starting point for K lies near the square root of n_training_data, in my case, $K = 40$ and $K = 100$ for the car and pen digits data set, respectively.

K	1	3	9	15	25	35	45	55
Car	0.879	0.915	0.911	0.881	0.844	0.805	0.788	0.76
Pen digits	0.994	0.994	0.99	0.987	0.984	0.979	0.974	0.97

Table 12: Test accuracy vs. K in k-NN

7.2 Weights

A refinement of the kNN classification algorithm is to weigh the contribution of each of the k neighbors according to their distance to the query point, giving greater weight to the neighbor closer to the query point. Table 13 shows the results. Compared to Table 12, weighted k-NN tends to be less overfitting with smaller values of K, and less underfit with larger values of K. As a result, the weighted k-NN should be preferred over the classical k-NN. The major disadvantage is longer computation time.

K	1	3	9	15	25	35	45	55
Car	0.818	0.892	0.905	0.88	0.876	0.826	0.807	0.775
Pen digits	0.993	0.993	0.991	0.988	0.985	0.983	0.979	0.977

Table 13: Test accuracy vs. K in weighted k-NN

7.3 Learning curve

The learning curve for the weighted k-NN, tested and trained on the car data set with $k = 40$ is shown in Figure 6. It's interesting to note that the training accuracy is always 1.0 regardless of the training size. This is true because the weighted k-NN is used and the query point itself dominates the classification. Like any other learning algorithms, as training size increases, the testing accuracy increases as well.

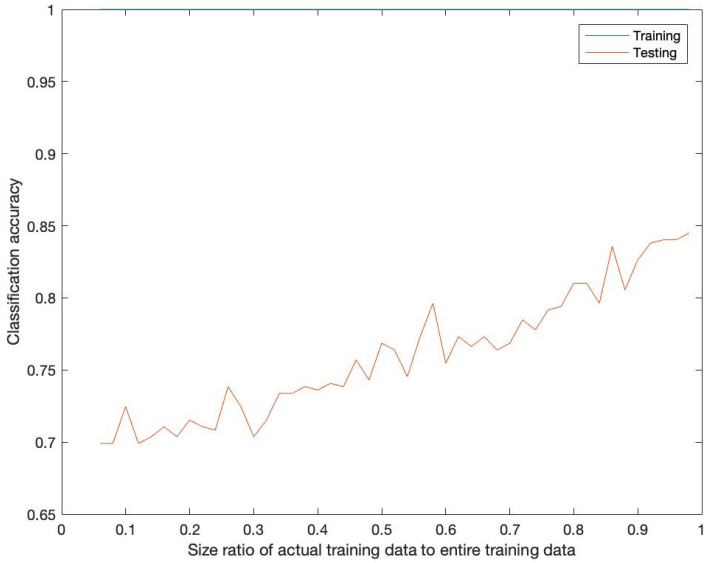


Figure 6: Learning curve for weighted k-NN

8 Comparison of SL Algorithms

In this section, the running time and test accuracy of 5 supervised learning algorithms will be compared and discussed. The hyperparameter for each algorithm was set to ideal values as discussed in the previous sections. Again, the test accuracy was obtained by averaging the test accuracy of 10 runs using

train/test splits. Table 14 and Table 15 show the comparison of test accuracy and running time, respectively.

Algorithm	Decision tree	Boosting	Neural network	SVM	k-NN
Car	0.965	0.935	0.944	0.993	0.829
Pen digits	0.958	0.928	0.985	0.996	0.962

Table 14: Test accuracy of 5 SL algorithms

Algorithm	Decision tree	Boosting	Neural network	SVM	k-NN
Car	0.121s	1.766s	8.209s	0.409s	0.23s
Pen digits	4.386s	14.764s	42.361s	2.795s	8.594s

Table 15: Running time of 5 SL algorithms

For both data sets, the SVM model achieves the highest test accuracy, with 3rd and 1st shortest running time. With high accuracy, nice theoretical guarantees regarding overfitting, and an appropriate kernel, SVM can work well even if the data isn't linearly separable. The major disadvantages of SMV is memory-intensive, and hard to interpret. Boosted decision tree and neural network take a much longer running time because the number of estimators in boosting and hidden layer structure in neural network greatly impact the model complexity. With both models, extremely complex models can be trained and they can be utilized as a kind of black box. However, it's very hard to clarify both model structures and parameterization is rather hard to understand. Decision trees can easily handle feature interactions and they're non-parametric, so we don't need to worry about encoding the feature vector space or normalization, or if the data is linearly separable. The downside of using a decision tree is it can be memory-intensive when memorizing a huge tree structure. In addition, it can easily overfit the training data. This is where ensenble algorithms, such as boosted decision trees, come into play. k-NN, a non-parametric, lazy learner, is one of the simplest while most used algorithms for classification. It makes no assumption about the underlying data distribution but it is computationally expensive because of the amount of data it needs to store and look up.

9 Conclusion

In practice, each algorithm has its advantages and disadvantages. When it comes to model selection and hyperparameter tuning, certain procedures need to be followed. There exist a lot more hyperparameters than the ones discussed in this report. Cross validation and multiple runs of train/test split can help one decide which model selection is preferred for specific problem data set. I hope the guidelines presented in this report can help other people in choosing their supervised learning algorithms and model tuning.