Supervised learning.

Kolomiets Dmitriy

kolomietsdv@gatech.edu

***Abstract*—In the report, I present my results of solving two machine learning tasks - regression and classification - using basic machine learning algorithms. The investigation shows the differences in efficiency and ways of tuning supervised learners such as support vector machines, neural networks, boosting ensembles, decision trees, and k nearest neighbors.**

1 CLASSIFICATION

1.1 The dataset

The classification dataset is interesting because it poses a multi label classification problem. This problem can also be represented as a binary classification problem or even as a regression. The dataset contains chemical indicators of different wines along with the appraisal of the quality made by the taster. The main goal is to learn a function that can predict the quality of a wine taking into account only the chemical indicators.

This list includes fixed acidity, volatile acidity, citric acid, residual sugar, chlorides, free sulfur dioxide, total sulfur dioxide, density, pH, sulfates, and alcohol.

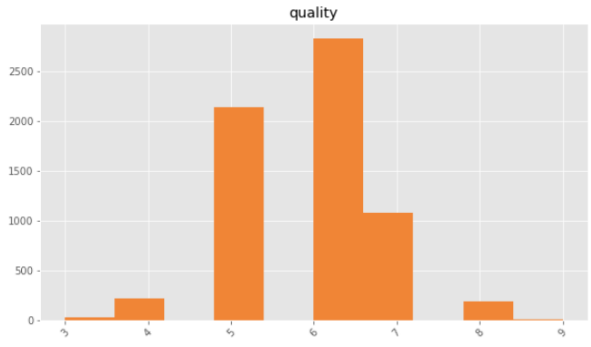


Figure 1—*Quality score distribution.*

I divided the final quality score that originally comes as a number in the range from 0 to 10 into 3 classes based upon the distribution of scores (figure 1):

1. The quality that is worse than average (with the score of less than 5).
2. Average quality (with the score of 5 or 6).
3. The quality that is better than average(with the score of greater than 6).

I made such division because while the original task is trying to look like the regression problem it still stays the classification one with the wide variety of classes that do have the order. In my opinion, the provided 3 classes are more natural for people who make their choice for dinner.

1.2 Validation strategy.

To train the learners I will randomly (to simulate possible distribution in the population) split the data into 2 parts - the bigger part that will contain 70% of the data and will be used for training, and the smaller part that will contain the rest and will be used for the final testing.

To fine-tune the parameters of each algorithm I will use the 3-fold cross-validation scheme - I will split the training data into 3 parts. During the validation, I will train the learner on every single fold and then compute the accuracy on the rest two and then will average the accuracy computed on each pair. I will use such a scheme to avoid over-fitting the training data.

To pick the best solution I will train each fine-tuned learner on the whole training set and will test it on the test set.

For the classification problem, I have to take into account the balance of classes in the data. The class with an average quality score has a frequency of 0.765, so if we will always predict the average score we will be able to reach 76,5% accuracy. It means that our machine learning algorithms should improve this result. It is also important to add that the accuracy is not always the best metric to use - depending on the task it can better to use particular class precision (e.g. if we don't want to overlook the wine that has a bad taste) or it can be based on something more complicated. I use accuracy (versus baseline accuracy) as it is easy to

understand and fits the problem. In most cases, I use the heatmap plot for fine-tuning the hyperparameters of the algorithms instead of lines and curves. I did it because sometimes there is no order in the values of parameters (e.g. trying various distance metrics or split criterion). The heatmap helps to show the local target function optimums.

I want also to notice that I won't be showing the resulting values of the training set - in most cases I will tune the parameters that influence the complexity of the model, it means that the target functions on the train set will grow (e.g. if it represents accuracy, or goes down if it represents an error) gradually and even reach its maximum values.

1.3 The learners.

1.3.1 *Decision Trees.*

The intuition.—One of the simplest types of algorithms is a decision tree. Basically, it splits the values of features into ranges and makes its decision looking to what range of values the object belongs to. The main goal for the tree is to find out (to learn) how to make the correct splits.

The hyper parameters' tuning.—The main characteristics of the tree are its depth and the criterion used for splitting the values of the features. For the classification problem, I tested the Gini index versus entropy along with the depth of the tree.

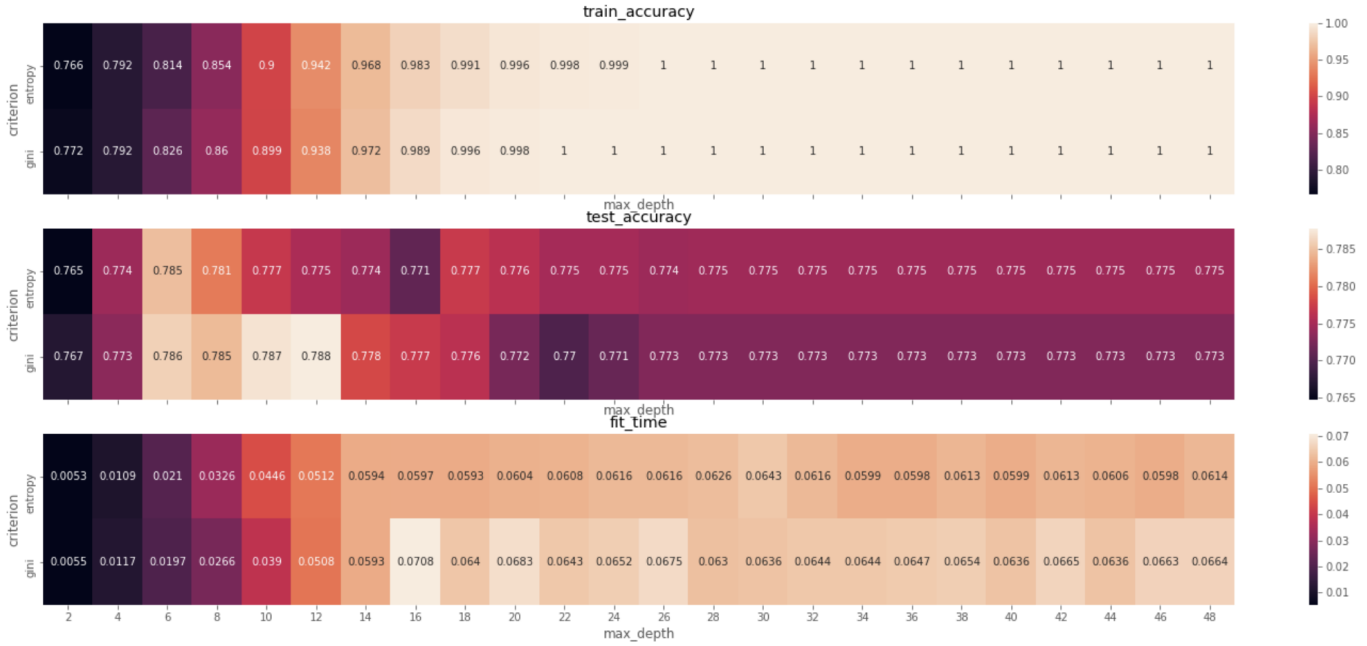


Figure 2—*Decision tree accuracy heatmap.*

I observe that using the Gini index provides slightly more accurate results on the valid set at average and also gets its highest accuracy at a value of 12 max tree depth. As it is shown in the figure 2 growing the tree further makes it more complicated and results in overfitting - the training accuracy reaches 1 (100% accuracy on the train set) but the accuracy on the unseen data goes down.

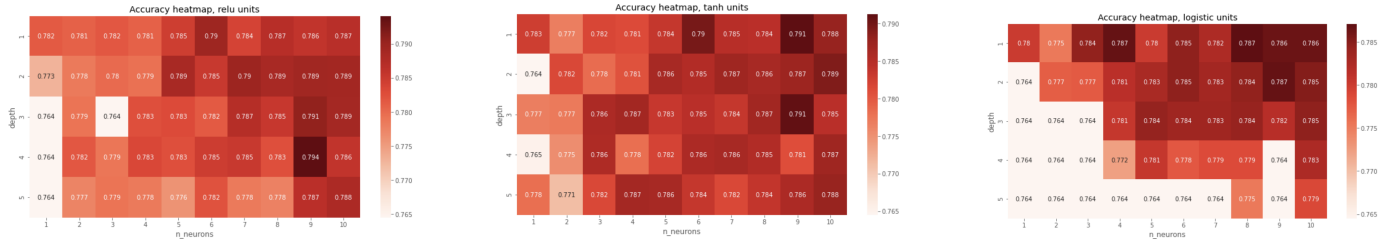
I have also tried to balance the weights of the classes according to the distribution on the train set but it hasn't improved the result on the validation set. However, I was able to get slightly better results pruning the tree with the "cost complexity pruning" technique. Pruning decreases the complexity of the tree but at very low values, it can be effective.

The test accuracy.—The optimal depth of the tree was 12 and 0.0005 for the pruning parameter, the test set accuracy was 0.8 that was better than the accuracy that can be obtained by predicting the most frequent class (0.77).

1.3.2 *Neural network.*

The intuition.—I have tried a simple class of networks - a multilayer perceptron. It is somewhat a complicated combination of logistic regression classifiers with a bit of non-linearity on their edges.

The hyper parameters' tuning.—The most important and basic characteristics of the neural network are its width, depth, and the type of non-linearity of the neurons. So I started to seek the best combination of them using grid search. I have tried three types of activation units, depth up to 5 layers with up to 10 neurons in each layer.



(a) RELU **(b)** Tanh **(c)** Logistic

Figure 3—*Activation units accuracy.*

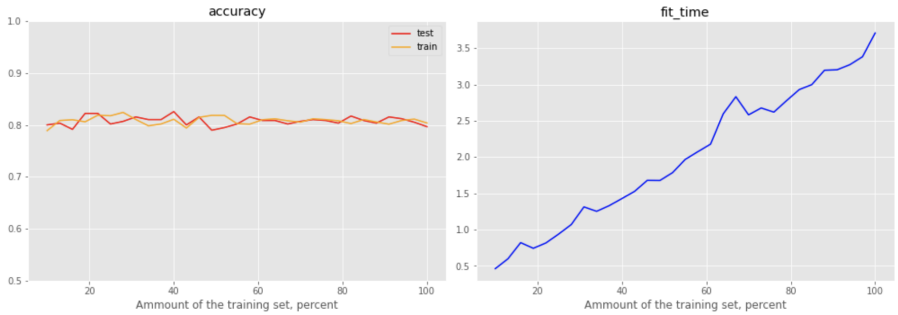


Figure 4—*NN-learning curve.*

The results show that relu units perform slightly better than the others. I also observe that the depth of 4 was enough to catch the underlying dependencies. The best result is obtained with 9 neurons on each layer no matter what type of unit it has.

What was also useful is to scale the values of the features to the standard distribution. Since each neuron learns to weigh the features their different scale causes slowing down the training process and accuracy decline.

Tuning other features such as the learning rate, alpha or the limit on the number of epochs don't result in any significant results. The lbfgs solver was much faster in training but its performance in terms of accuracy was slightly worse.

What is also interesting is the learning curve. The train performance matches the cross-validation performance. On the one hand, it means that it doesn't overfit, on the other hand, the performance itself is not very high.

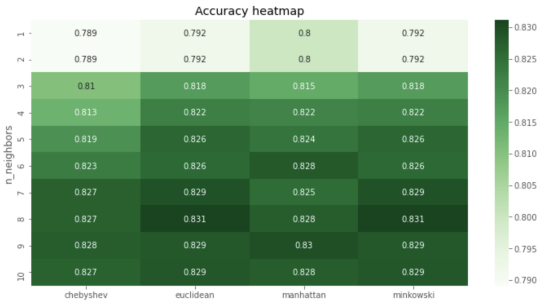
The test accuracy.—The best parameters for a neural net were 9 neurons in 4 layers with relu activation and "Adam" solver. The test set accuracy reached 0.79 which is higher than my baseline of 0.77, but worse than a simple decision tree result.

1.3.3 *K nearest neighbours.*

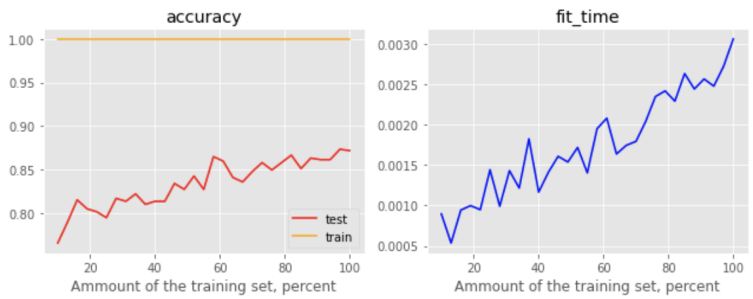
The intuition.—The intuition is very simple - you pick some amount of nearest points in the feature space from the training set and make prediction based on their labels. If you are solving a classification problem you can vote for a class, or take an average of the labels as a solution for a regression problem.

The hyper parameters' tuning.—The final efficiency of the algorithm mostly depends on the number of neighbors you use, the way you measure the distance, and how you combine the labels in the end to make a prediction. I tried to combine searching all

of them together. What is also important is that the result is much better (6.7% improvement) on a scaled to standard distribution feature space.



(a) KNN accuracy heatmap.



(b) KNN learning curve

I found out that I get the best result (0.83 accuracy) when I use 8 nearest neighbors, "euclidean" distance with voting weighted by the distance. What is interesting that the efficiency if you choose some other distance metric (e.g. manhattan or Chebyshev) won't be significantly worse. It is also rather stable no matter how many nearest points you will use starting from 5 to 10. It seems like this algorithm best fits the underlying nature of the dataset. The learning curve shows rather low test accuracy variance and grows with the growth of the training data. What is interesting is that no matter what k we choose on the test set we never make mistakes. It probably says that on the small subsets of the data the points on each class probably form somewhat dense blobs.

The test accuracy.—The test set of the best-tuned knn was even higher - 0.87 of accuracy, which was also much higher than those produced by the decision tree and the neural network.

1.3.4 *Boosting.*

The intuition.—The intuition for boosting is somewhat complicated but in simple words, it as an ensemble technique takes a pack of simpler algorithms (the most popular choice is a decision tree), sets different goals for each of them, fits them, and then combines their predictions. I tried two types of boosting - AdaBoost and gradient boosting..

The hyper parameters' tuning.—As a base, I took a decision tree. I have tried up to 200 trees with different depths for both types of boosting.

The best cross-validation accuracy was pretty similar. The AdaBoost reached 0.84 accuracy with 160 decision trees with a depth of no more than 17, but it was pretty stable in the vicinity. Almost every combination of the number of trees greater than 30 and depth of the trees greater than 7 had the result of at least 0.83. It means that it captures the bulk of the underlying dependencies in the training set. However, tuning pruning and learning rate didn't improve the result.

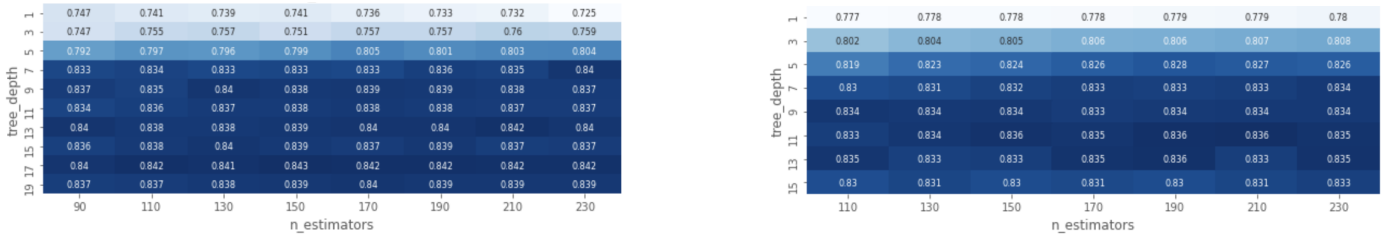
A similar picture was for the gradient boosting - a pretty stable result for the number of trees more than 70 with the depth between 7 and 11 with the peak accuracy of 0.837 with 210 trees of depth 11 without any further improvements on tuning learning rate or pruning.

It is also important to notice that the result on the scaled feature set was slightly better. The learning curve shows variance decline and accuracy growth with the growth of the training data.

The test accuracy.—The test set accuracy for both types of boosting was equal to - 0.86 accuracy. It is the second-best result so far after knn.

1.3.5 *Support Vector Machines.*

The intuition.—The Support Vector Machine resembles the linear classifier, it builds a line, that divides the feature space into 2 areas - the area for the positive class and the negative. The main feature of it is that it makes additional requirements for the line to be right in the middle of the area between the classes. The second key feature of the SVM is that to build the dividing surface it uses the similarity function between two objects in the feature space and it can be somewhat arbitrary - the user can provide



(a) Adaboost **(b)** Gradient Boost

Figure 5—*Boosting types.*

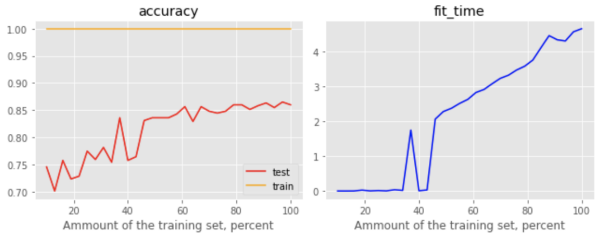
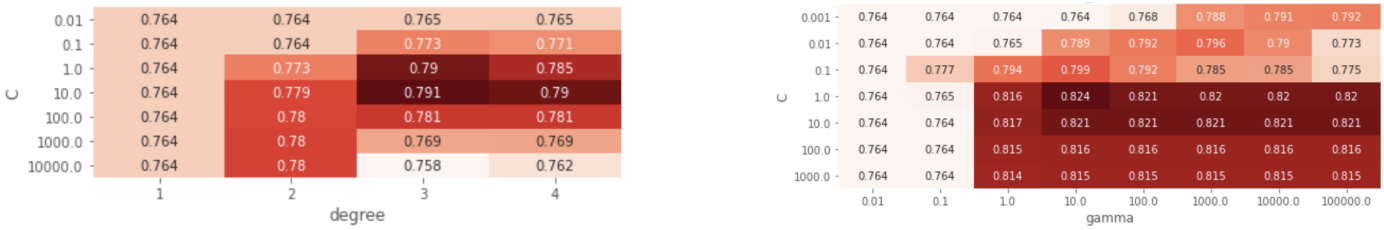


Figure 6—*Adaboost learning curve.*

the similarity function that best captures the domain of the data.



(a) Polynomial **(b)** RBF

Figure 7—*SVM kernels.*

The hyper parameters' tuning.—The similarity function is also called a kernel. I started by searching for the best combination of the kernel function and its main parameters. The difficulty is that each kernel has its own parameters. I tried polynomial kernel with different possible degrees and different values for the regularization parameter "C"; and radial basis kernel with different values for "gamma"(which represents influence distance) and "C" - regularization.

With C greater than 1 and gamma greater than 1 the SVM with the rbf kernel becomes stable with accuracy greater than 0.8 with the maximum 0.824 (C=1, gamma=10) on the scaled feature space. The polynomial kernel's efficiency for degrees less than 4 and a wide range of C wasn't been able to achieve a result better than 0.791.

The learning curve shows low variance and accuracy slow growth with the growth of the training data, as well as the perfect train scores.

The test accuracy.—The best result on cross-validation was for the SVM with rbf kernel which was performed on the test set with an accuracy of 0.85.

1.4 The best solution.

While the best score on the test set (0.87 accuracy) was achieved by the KNN algorithm I would consider the first three algorithms depending on what other features of the ML algorithm matter for the task (eg. SVM would be more computationally efficient and fast in predicting, KNN sometimes can be more interpretable).

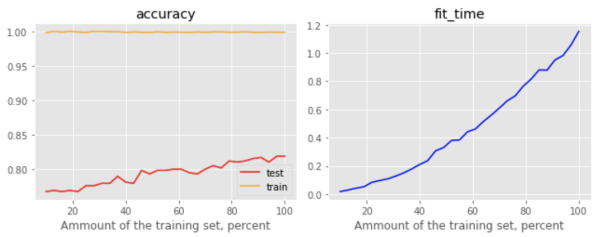


Figure 8—*SVM learning curve.*

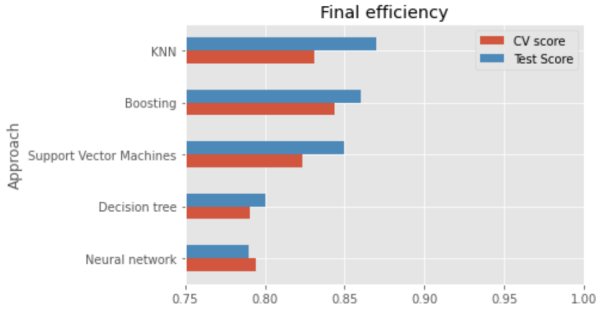


Figure 9—*Final scores classification.*

2 REGRESSION.

2.1 The dataset

The regression dataset contains day weather information along with the number of bikes rented that day by the bike rental company. It is interesting because its dependencies are more complicated than those from the classification dataset it it shows what algorithms are able to deal with and who retreats.

The indicators include humidity, temperature and feeling temperature, wind speed, and weather. They also include whether the day is a holiday or working day, hour, year, and month.

The main goal for the problem is to predict the number of bikes that would be rent from such features.

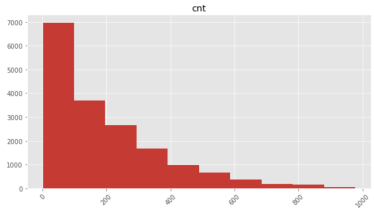


Figure 10—*Rented bikes number distribution.*

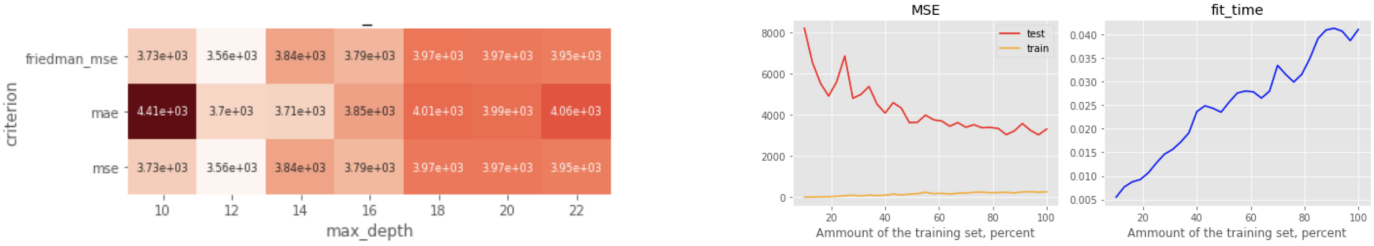
2.2 Validation strategy.

I will follow the same 3-fold cross-validation strategy as I did for the classification problem with measuring test set performance in the end. As the pivotal metric I will use mean squared error because it's differentiable, hence in simple words often requires less time to be optimized by algorithms. As a first baseline, I will take 32442 - the mean squared error that can be reached if I will always predict 189.56 - the mean value across the training set. It will help me to understand whether there is any use of ML algorithms.

2.3 The learners.

2.3.1 *Decision Trees.*

Regression nuances.—In opposite to the classification version, regression decision trees have different optimal split criterions - mean squared error and mean absolute error. Though mse criterion fully matches my final efficiency measure I iterated over all options to find the best combination of the metric and depth of the tree as its primary parameters. ***The hyper parameters' tuning.***—



(a) Depth and criterion tuning (cv performance)

(b) Learning Curve

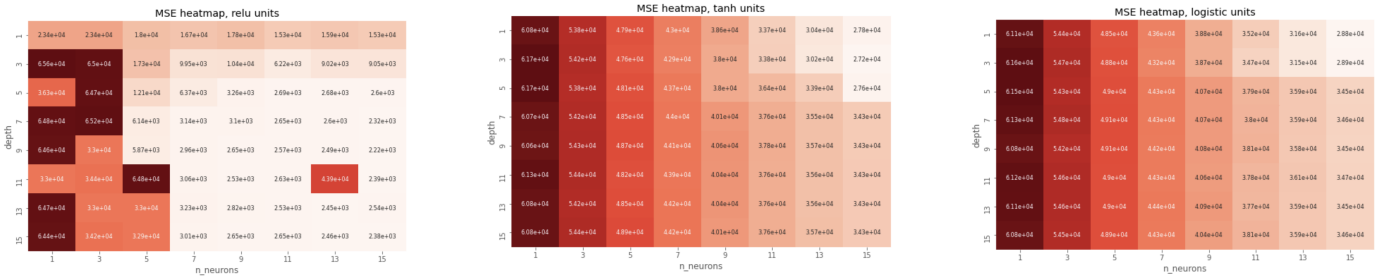
Figure 11—*Decision Tree Regressor.*

As expected the least cross-validation mean squared error was reached with the MSE criterion (and tree depth of 12) - 3560. I was able to slightly improve this result down to 3526 with a bit of cost complexity pruning.

The learning curve shows variance decline and MSE reduction with the growth of the training data. ***The test accuracy.***—The test set performance was even better - 3072. It is 10 better than our primitive baseline, consequently, it makes sense to prefer at least a simple tree for this problem.

2.3.2 *Neural network.*

Regression nuances.—The only difference between classification multilayer perceptron and regression one is the activation function in the output layer - instead of using a sigmoid function which values can represent the probability of the class it uses the identity function. ***The hyper parameters' tuning.***—The parameters to tune are still the same, I tested a wide variety of depth and number of neurons and the type of the hidden layer activation functions on the original and scaled feature space. The learning curve shows variance decline and MSE reduction with the growth of the training data. What is also interesting is that while the cross-validation performance goes down the training error goes up. It is so probably because the tuned alpha parameter that is responsible for regularization struggles to prevent overfitting to the training set.



(a) RELU

(b) Tanh

(c) Logistic

Figure 12—*Activation units accuracy.*

Again, relu activation and the scaled feature space were the best with the result of 2219 MSE, however, this time the neural network needed to be much deeper (9 depth) and wider (15 neurons). Tuning alpha and learning rate cut this cross-validation error down to 2126 and the lbfgs solver cut down the fit time from 8 seconds to 1.5 seconds.

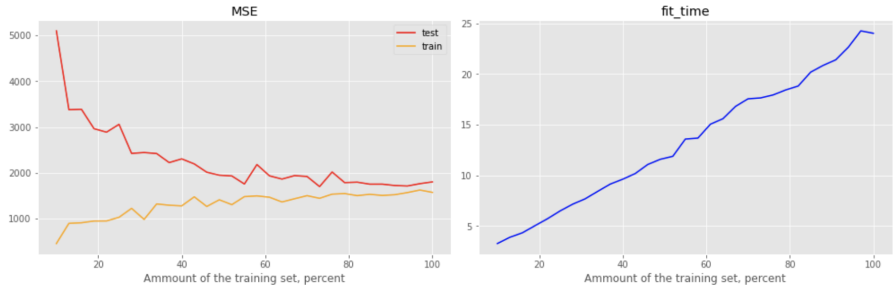


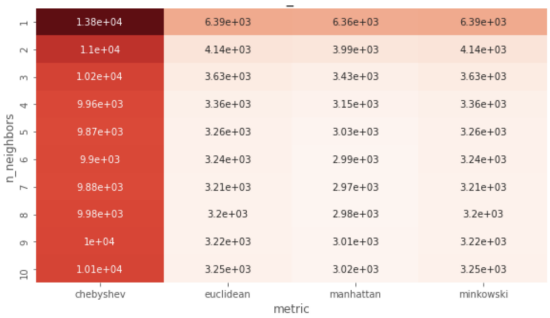
Figure 13—*NN-learning curve.*

The test accuracy.—The test set performance was even better - 1802 MSE.

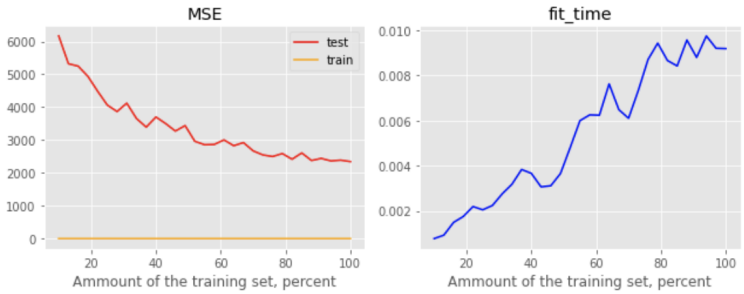
2.3.3 *K nearest neighbours.*

Regression nuances.—It's pretty much the same algorithm as it was for the classification problem but it averages the result in the end instead of voting for the class.

The hyper parameters' tuning.—The parameters to tune are the same - firstly, we have to find the best combination of the distance metric and the number of neighbors, then we can fine-tune the combining function. Moreover, we should try original and scaled feature space.



(a) KNN MSE heatmap.



(b) KNN learning curve

What is interesting is that for this particular regression problem the best result - 3247 - was reached on the original feature space (with k=7). The result on the scaled version was much worse (the MSE error 3 times higher) Weighting average as it was in classification problem made a significant improvement - cut down the error to 2967. The learning curve shows low cross-validation MSE variance and reduction with the growth of the training data.

The test accuracy.—It also shows decent efficiency with 2586 MSE.

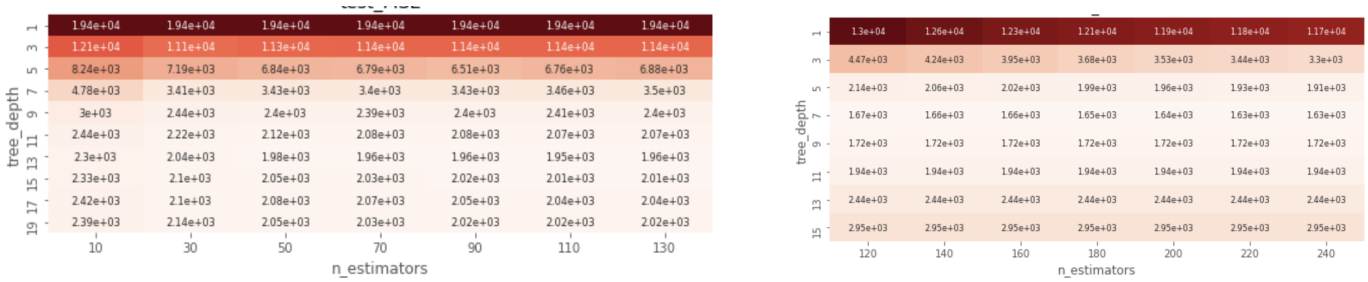
2.3.4 *Boosting.*

Regression nuances.—The only difference in boosting approach from the version used in the classification problem was the change in the base learner - instead of using classification decision trees, I used regression trees.

The hyper parameters' tuning.—Like it was in the classification, there was no significant difference between using scaled or original feature space for the boosting approach but the result was slightly better on the scaled one. The best cross-validation performance was 1952 with 110 boosted trees with a depth of 13. The result was stable with low variance for the arbitrary number of trees greater than 40 (<= 2000 MSE) of depth 13.

The best cross-validation performance for gradient boosting was 1631 on the scaled feature space with 240 boosted trees with a depth of 7. The result was also stable for the arbitrary number of trees greater than 100 (<= 1700 MSE) of depth 7.

In both cases pruning the trees and tuning the learning rate didn't improve the cross-validation efficiency.



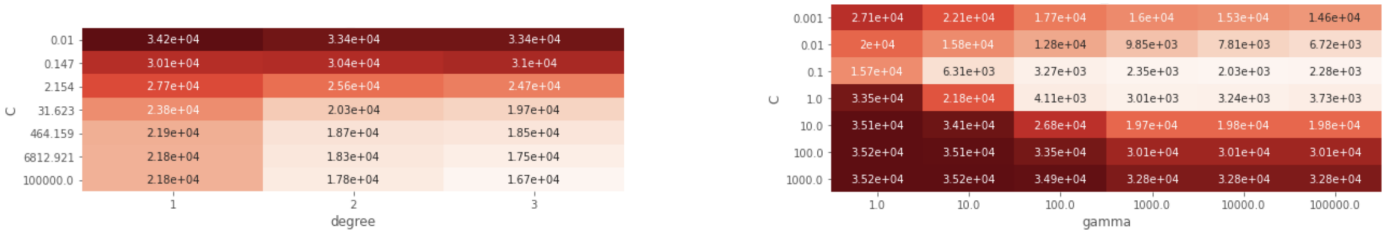
(a) Adaboost **(b)** Gboost

Figure 14—*Boosting types.*

The learning curve shows low cross-validation MSE variance and reduction with the growth of the training data.

The test accuracy.—The test set performance for the AdaBoost was better than that on the cross-validation stage - 1832, the gradient boosting approach showed was even better - 1479 MSE.

2.3.5 *Support Vector Machines.*



(a) Polynomial **(b)** RBF

Figure 15—*SVM kernels.*

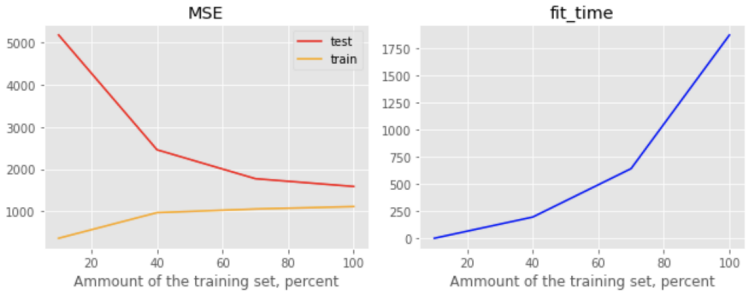


Figure 16—*SVM learning curve.*

For this particular regression problem, I had to test very high values for the regularization parameter C to achieve the result comparable to those achieved by other ML algorithms. For the RBF kernel, the best result of 2026 was achieved with a value of gamma=10000 and C=0.1 on the original feature space.

The lowest achieved error for the polynomial kernel was 8 times higher, however, I should add that it was achieved at the edge of my parameters' grid, hence the error could possibly be lower for the higher degrees (> 3) or for the C > 100000, but the training time grows up exponentially if I grow the parameters further and it becomes unreasonable in many cases to continue searching. The training time for this combination was more than 10 minutes for training. What is interesting is that on the scaled feature space the best result was much worse for both types of kernels. ***The test accuracy.***—The test set MSE error was even lower - 1855.

2.4 The best solution.

The best solution for the regression problem was the gradient boosting algorithm. It had the lowest both cross-validation error and test error. The second best was deep neural network and support vector machines with rbf kernel.

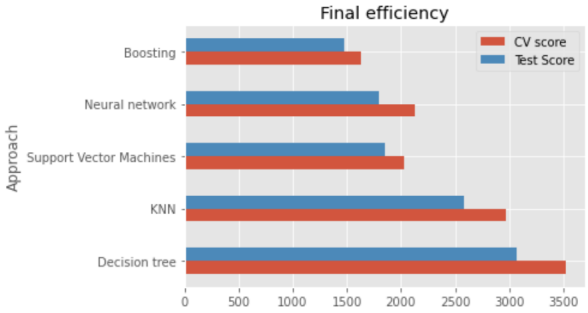


Figure 17—*Final scores classification.*

Being the best solution for the classification problem KNN was much worse for this particular regression problem.

I suppose, the reason is that the dependencies in the Bikes dataset were more complicated. It can be seen from the optimal values of the parameters which represent the complexity - the number of the base learners and the depth of the base trees for boosting; or the depth and width of the neural network. Probably, the KNN algorithm cannot scale its complexity with its parameters, it relies on the feature space and on the fact that the distance between objects matters. I'm sure that thorough tuning the feature space would improve the KNN, but sometimes it's more convenient and less time consumable to take the boosting or the neural network which apparently deal with this problem on itself.

3 CONCLUSION.

To successfully solve the machine learning problem you have to be acquainted with the wide variety of machine learning algorithms, and understand the pros and cons of each one. Furthermore, often the safest way is to try all possible solutions and pick the best according to its efficiency and computational resources. Sometimes it is better to pick a more simple and fast solution trading the accuracy, other times it is worth finding the most precise estimator no matter how much time it requires to train and predict.

I showed some basic algorithms in the field and approaches to tune their base parameters. Each of the techniques can be tuned more thoroughly, especially in terms of the feature space and bringing the domain knowledge to help the learner to catch the underlying dependencies.

What is also important is the amount of data you have to do your experiment. In most cases the more data you have, the less error it contains, the better solution you will be able to find, and the more computation time resources you will need to have.

REFERENCES

[1] 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, 2009.

[2] Fanaee-T, Hadi, and Gama, Joao, 'Event labeling combining ensemble detectors and background knowledge', Progress in Artificial Intelligence (2013): pp. 1-15, Springer Berlin Heidelberg.

[3] Scikit-learn: Machine Learning in Python, Pedregosa et al., JMLR 12, pp. 2825-2830, 2011.