Application Project Final Report

Machine Learning Spring 2018

Team: Kimcheese
Members:

Andy Dohoon Kim	435168	dohoon.kim@wustl.edu
Annie Chaehong Lee	444166	annie.lee@wustl.edu
Jaesang Ha	419887	haj@wustl.edu
Ryun Han	429091	r.han@wustl.edu
Nigel Seunghwan Kim	431659	seunghwan.kim@wustl.edu

GitHub: https://github.com/anniechaehonglee/cse517project

Motivation

Coming from a country with the highest alcohol consumption rate, we decided as a whole to experiment and analyze the wine dataset imported from UCI Machine Learning Repository (http://archive.ics.uci.edu/ml/datasets/Wine+Quality).

Dataset
The dataset is separated into white and red dataset, and includes approximately 6500 data points with the following 12 features:

1. fixed acidity
2. volatile acidity
3. citric acid
4. residual sugar
5. chlorides
6. free sulfur dioxide
7. total sulfur dioxide
8. density
9. pH
10. sulfates
11. alcohol
12. quality (score between 0 and 10)

The dataset's outputs—quality scores—are centered around the mean of 5 and does not include values outside of range [3,8]. We suspect this is because sommeliers are more inclined to give scores within the middle range rather than scoring them the extremes.

Goals and application
With the dataset, we proposed to find a model that can reliably predict wine's quality solely from its chemical components. With a best performing algorithm, we can further develop an app that many wine consumers including chefs, restaurant owners, and hotel managers can use to select their choice of wine and evaluate through just taking a picture of a label.

Models

Throughout the semester, we explored total of 8 different models of the following:

1. Logistic Regression
2. AdaBoost
3. Decision Tree
4. Random Forest
5. Gaussian Process
6. Kernel SVM
7. DNN regressor
8. Clustering

The first problem we ran into was the unexpected inconsistency of the output labels. Although the anticipated classes are from 1 to 10, there were only 6 classes, ranging from 3 to 8. Also, the score itself is approximately normally distributed, and there are only a few data points for both ends of the score spectrum. So, when we tried to use multiclass classification, classifying them to their true classes, we got very low accuracies, ranging from around 40% to 60%. To improve the accuracy, we used binary classification that increased the accuracies up to ~95%. However, we realized although this improves the accuracy, it is not optimal for our problem and has no advantages in real life because we are not classifying wine as either good or bad. So, with models explored later on we tune the model into multiclass (1vs1, 1vsAll) and integrate regression and classification.

We note some interesting points with some of our models:

Logistic Regression

We first applied multiclass Logistic regression to classify in 10 categories. We observed the accuracy around 0.5409. We believe that this poor performance in prediction is from of our non-uniformly distributed output labels. So, we tried with binary Logistic regression with generating two classes for outputs in datasets with one class for [0,5] and another class for [6,10]. Then, the accuracy distinctively rose to 0.9612. We observed logistic regression generally applies well to binary classification; however, binary labels were not applicable to our theme of project hence we needed to find other methods to deal with 10 classes.

Decision Tree

Using sci-kit learn API, we implemented Decision Tree model with two different approaches to the dataset. First approach was to implement 11-class classification, which yielded the accuracy around 0.59. We believe this is a decent accuracy for the 11-class classification. Then, using the same method from Logistic Regression model, we changed the model into binary classification, which yielded the accuracy around 0.93. Despite the high accuracy, we did not use the binary classification model for the same reason as Logistic Regression model.
Strength of the decision tree model would be it can even classify highly non-linear model. Also, as the model only thinks of one feature at a time, the model can quite clearly classify the data even if it is centered and not evenly spaced, just as our dataset. Also, efficiency-wise, the model is trained and tested very quickly.
Weakness of the decision tree model would be the overfit. As we did not specifically implemented pruning in our decision tree algorithm, the model is thoroughly fitted to perfectly classify the training data. Thus, in-train accuracy for our decision tree model is always 1. However, this affects

the performance of the model negatively when the model is applied to outside dataset, other than the training data.

Random Forest

Using sci-kit learn API, we implemented Random Forest model into only 11-class classification. For the number of features used in each dataset, the result of the accuracy is like the following:

# features	1	2	3	4	5	6	7	8	9	10
Accuracy	0.577	0.5854	0.6145	0.5895	0.61875	0.633	0.59583	0.64375	0.6375	0.61875

From 6 and below, although the accuracy average is decent, the variance among the accuracies increased significantly, so 8 as a number of parameters in the random forest model showed the best result.
Strength of the random forest model would be that it takes the advantages of the decision tree model and uses ensemble learning method in order to increase accuracy and decrease the dependence between features.
Weakness of the model would be that just like the decision tree, it overfits, if there are too many data points in the training data or if we choose too many features, causing the algorithm to lose the "randomness" in choosing features. However, with our dataset, it seemed that the model did not show huge problems with this possible weakness.

AdaBoost

Adaboost learns a hypothesis function from multiple weak learners. For this model, users can input how many weak learners to learn from. Adequate number of weak learners and boosting can actually enhance decision trees from overfitting too much. However, we observed that too many weak learners can in fact worsen the accuracy. Although this model gives quick results, overall, we learned that this model does not meet our goal accuracy because it is so sensitive to noise and outliers which our dataset is prone to due to its source.

Gaussian Process

In Gaussian Processes, we approached the problem in two ways: 1-vs-1 and 1-vs-all classification. The best GP model was investigated by tuning the kernel and its hyperparameters. The main criterion in evaluating the optimal kernel was by comparing negative log marginal likelihood of each model. RBF Kernel (l=1,λ=1) and Matern Kernel(l=1,λ=1,ν=1.5) yields very similar results, but GP with RBF kernel takes much faster to run. While Gaussian processes yield a high accuracy, it is significantly slow to run than other models. Possible explanation is the joint belief update requires the kernel solver longer to run. For final comparison purpose, RBF kernel was chosen based on its efficiency.

(1-vs-1)	Train Acc.	Test Acc.	Neg-Log Marginal likelihood	CV Mean	Efficiency(sec)
RBF *(l=1, λ=1)*	0.972	0.567	-246.149	0.47 (+/- 0.06)	1.815
Matern *(l=1, λ=1, ν=1.5)*	0.625	0.560	-244.840	0.54 (+/- 0.09)	74.749

Kernel SVM

Kernels	Accuracies										Mean	Variance
polynomial	50.00%	61.49%	55.28%	50.63%	48.13%	62.50%	50.00%	51.88%	52.53%	52.87%	53.53%	0.002136453
sigmoid = 20	42.59%	42.24%	42.24%	42.50%	42.50%	42.50%	42.50%	42.50%	43.04%	43.31%	42.59%	1.01319E-05
sigmoid = 5	46.30%	44.10%	42.24%	43.13%	41.25%	44.38%	41.88%	41.25%	46.84%	42.04%	43.34%	0.000364477
rbf = 128	44.44%	44.72%	42.86%	43.13%	43.13%	43.75%	43.13%	42.50%	43.04%	43.31%	43.40%	4.42743E-05
rbf = 20	46.30%	43.48%	43.48%	43.13%	43.13%	43.75%	43.13%	43.75%	44.94%	42.04%	43.71%	0.000121259
rbf = 10	46.30%	44.10%	42.24%	43.13%	41.88%	41.25%	42.50%	43.13%	46.84%	42.68%	43.40%	0.000304695
rbf = 2	48.77%	45.96%	43.48%	45.63%	40.00%	41.25%	41.88%	45.00%	48.73%	38.85%	43.95%	0.001074307
rbf = 0.1	48.77%	62.73%	53.42%	53.13%	59.38%	72.50%	63.13%	58.75%	62.66%	60.51%	59.50%	0.0044117
linear	50.00%	59.63%	51.55%	51.25%	53.75%	63.13%	50.63%	53.75%	58.23%	50.32%	54.22%	0.00186932

We observed that according to train/test accuracy and cross-validation result, linear, polynomial, and RBF kernels all produces similar results if the parameters are set properly, while sigmoid kernel seemed inefficient even with significant changes to the gamma value. We concluded that RBF kernel with gamma value 0.1 produced the best result.
Our kernel SVM function gives users flexibility to try different parameters, which could potentially allow users to find the best model for our dataset. This allows the algorithm to model non-linear decision boundaries which is exactly what we needed. Also, if we can pick the perfect kernel for our model, SVM can be robust against high-dimensional dataset like ours. However, it is hardly possible to find the perfect kernel and may take a long time picking the right parameters.

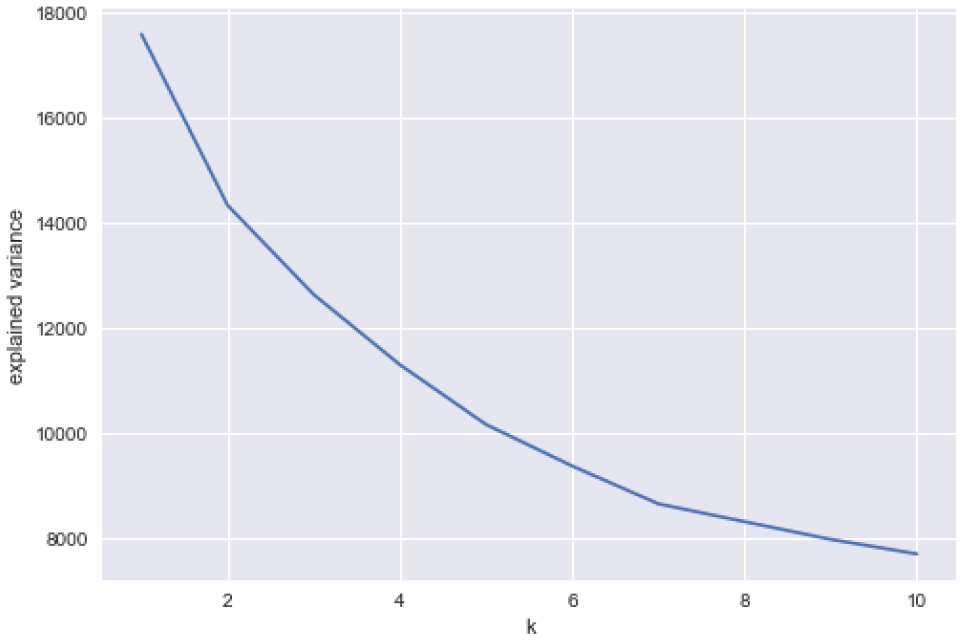
DNN Regressor
For neural network model training, we are not using cross validation for the following reasons:

1. In general, neural network relies on huge dataset and cross validation becomes too expensive, and
2. the efficiency of the model mainly relies on the dataset, number of epochs and the learning rate.

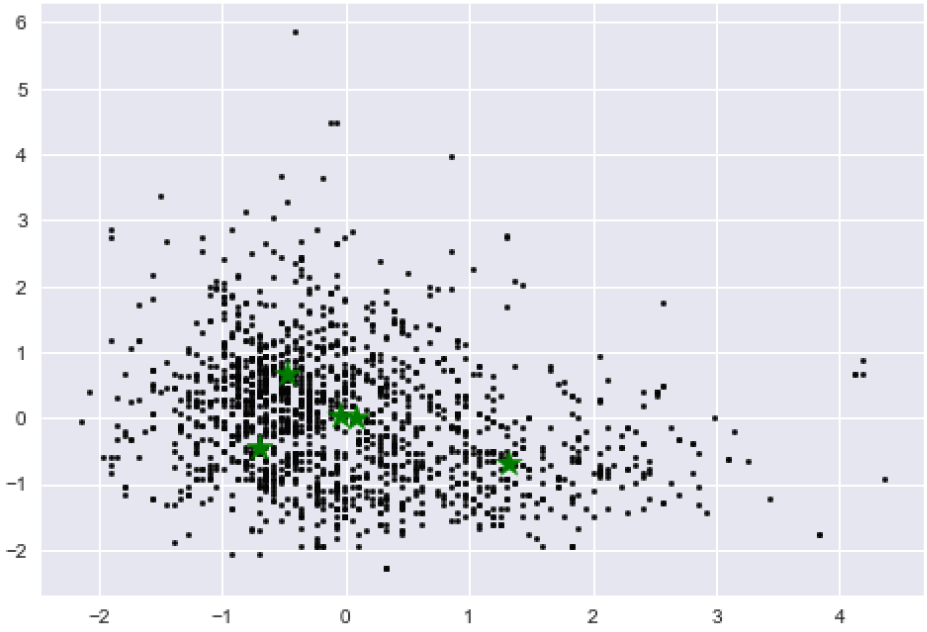
We ran the model with three different activation functions including sigmoid, tanh, and ReLU, and varying hidden layers. We concluded that deep neural network performs the best with sigmoid activation function. We observed that sigmoid and rectified linear unit performs approximately the same, but sometimes relu performs seriously badly with RMSE of up to 150. We found that not necessarily more layers and more nodes doesn't improve the performance. Our results suggest that about 5-7 layers with nodes around ~10 performs the best, with average loss of 7e-07.

Clustering

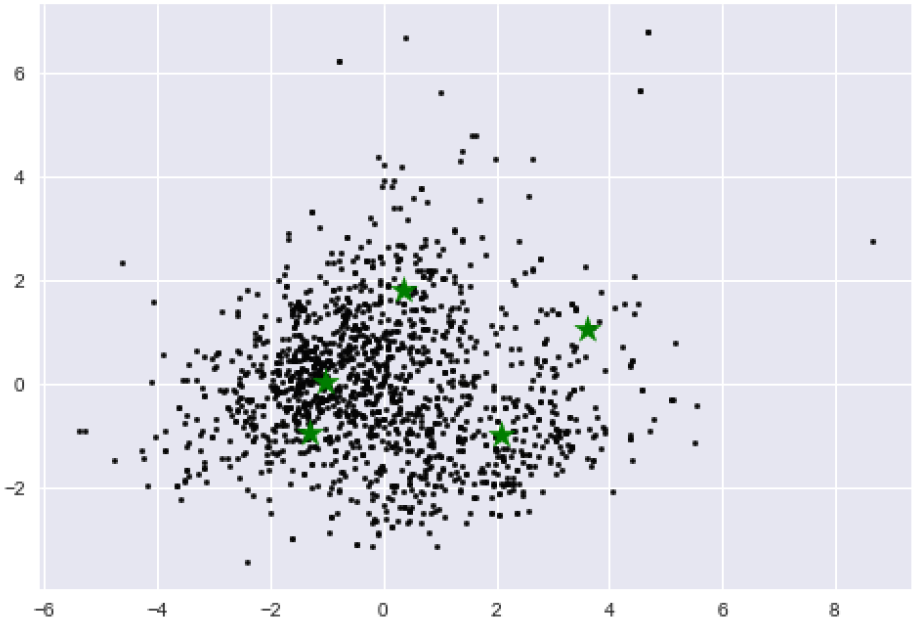
For clustering, we first applied generic K-means clustering, which is known for hard clusters where data points either belongs to a cluster completely or not. The interesting point of enlightenment from k-means clustering on our high dimensional data was that even though we picked 5 clusters through the elbow's method which shows the optimal number comparing through diminishing marginal gains of explained variance, as shown



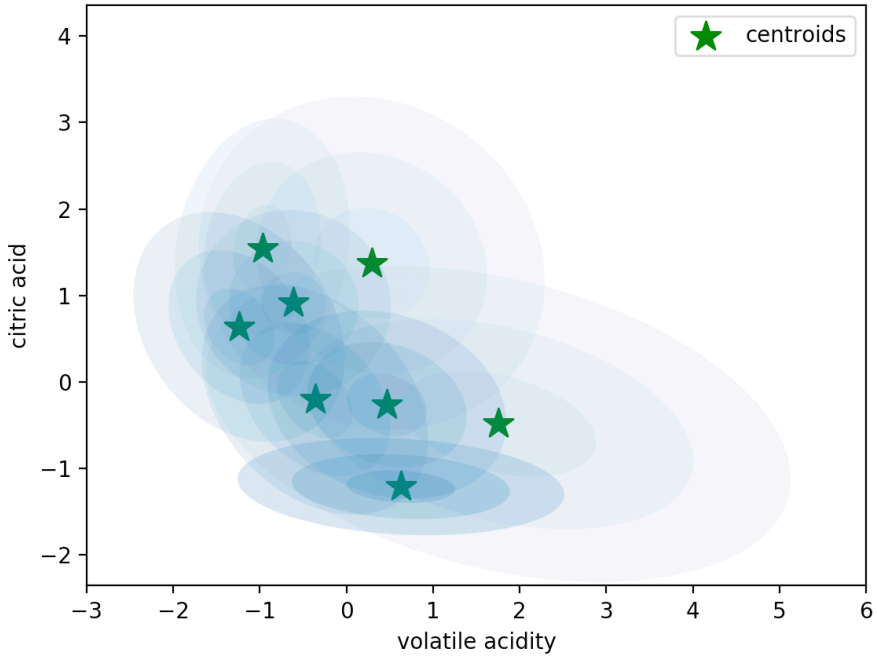
we could not sparsely classify the data points in the clusters as,



5 strict centroids were not sparsely distributed among dataset and reasoning behind this is high dimensionality in feature spaces and data points could not be strictly separable. Thus, we applied PCA which reduces the dimensionality in feature space and GMM method that is soft bound with probabilistic way of classifying the data points. PCA version showed increment in sparsity among the data points, as shown

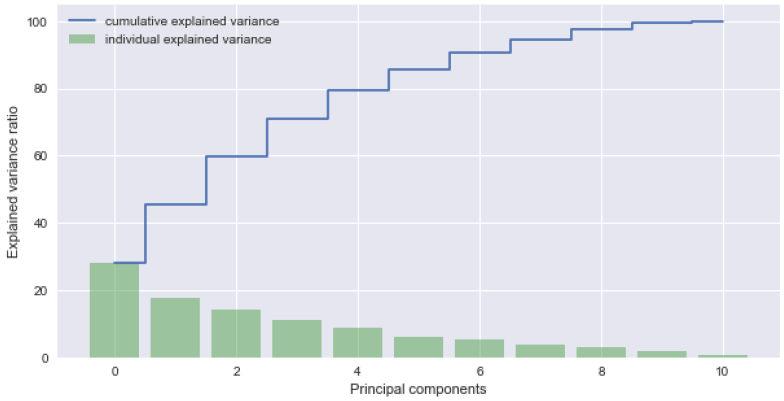


GMM method picked 8 centroids and it was shown,



which shows the probabilistic clusters where data points could be overlapped.

Data Whitening



We also whitened our dataset to improve performance. This was conducted through PCA method. Since we have high-dimensional feature spaces and various ranges of values involved, we normalize the data first. We chose the optimal number of principal components through the explained variance and reduced the feature dimensionality. The number of principal components was attained from the point where the marginal gain in explained variance drops. The dataset with reduced dimension of feature space was applied to all the algorithms we implemented.

Model Comparison

To analyze and evaluate all of our models, we ran 10-reruns of 10-fold cross validations for all 7 models with each other, comparing their 1. Average accuracy measured by test set accuracy based on the model trained upon the separate training set, and 2. runtime for training and testing measured in terms of seconds it took for the given task. In testing both accuracy and efficiency, we used same training set and test set for both models within the single test.

As a suitable statistical test method, we use Independent Sample T test:

1. Student's T test: Two-sample location test—such that means of two populations are equal (variance of the two populations are assumed to be equal)
2. Welch's T test: Two-sample location test—such that means of two populations are equal (Assumption dropped)

We used these and conducted 2-independent sample difference of mean Welch's T-test because of the following three reasons:

1. We are not sure what distribution does the train accuracy and the time takes.
2. We are not sure about the population for both samples.
3. We are not sure about population variances for both samples.
4. We are not sure whether two populations (for respective samples) have the same population variance.

5. We have a large sample size (both models have 100 samples), so technically, we could use Central Limit Theorem and use standard normal distribution as our pivotal statistic. However, we wanted to guarantee the largest confidence interval, so that our p-value could have more significant meaning.

In our test, we are comparing whether one sample is better than the other, so we used one-sided Welch's T-test, and we set our significance level at 0.005. This is less than usual alpha value for the test, which is like 0.1, 0.05, or 0.01 because, again, we would like to say that two models yield statistical difference in accuracy and efficiency when we are certain.

The following is an example out of the model evaluation program:

1. Accuracy

```
Models to Compare:
1. randomForest.py
2. adaBoost.py
New degree of freedom: 194.972350114
Test T-Score: 29.341955932
Comparable T-score: 1.65270642859
Significantly, randomForest.py is better than adaBoost.py / randomForest.py mean accuracy:
0.658026336478 / adaBoost.py mean accuracy: 0.506693003145
```

2. Efficiency

```
Comparing kernelsvm_traintime and gp_traintime
New degree of freedom: 9.002406229198515
Test T-Score: -13.920171190539653
Comparable T-score: 3.2496157006008293
Statistically, no difference detected. But in this sample, kernelsvm_traintime is slightly
better.
kernelsvm_traintime mean train time: 0.0825884103
```

These are the tabularized results:

1. Accuracy

	Adaboost	Decision Tree	DNN regressor	Gaussian Process	Kernel SVM	Logistic regression
Decision Tree	Significantly, decisionTree.py is better than adaBoost.py / adaBoost.py mean accuracy: 0.5095566037735799 / decisionTree.py mean accuracy: 0.6190628930817401					
DNN regressor	Significantly, adaBoost.py is better than DNNregressor.py / adaBoost.py mean accuracy: 0.51305896226416 / DNNregressor.py mean accuracy: 0.36544741660699	Significantly, decisionTree.py is better than DNNregressor.py / decisionTree.py mean accuracy: 0.62388836477986 / DNNregressor.py mean accuracy: 0.36872979858294				
Gaussian Process	Significantly, GPeval.py is better than adaBoost.py / GPeval.py mean accuracy: 0.6282838050314465 / adaBoost.py mean accuracy: 0.5099473270440251	Statistically, no difference detected. But in this sample, GPeval.pyis slightly better. GPeval.py mean accuracy: 0.6325856918238995	Significantly, GPeval.py is better than DNNregressor.py / GPeval.py mean accuracy: 0.6306371855345699 / DNNregressor.py mean accuracy: 0.379097205636552			
Kernel SVM	Significantly, kernelSVM.py is better than adaBoost.py / adaBoost.py mean accuracy: 0.5087594339622601 / kernelSVM.py mean accuracy: 0.60343474842768	Significantly, decisionTree.py is better than kernelSVM.py / kernelSVM.py mean accuracy: 0.600065251572 / decisionTree.py mean accuracy: 0.620192610063	Significantly, kernelSVM.py is better than DNNregressor.py / kernelSVM.py mean accuracy: 0.60606485849058 / DNNregressor.py mean accuracy: 0.36696242337396	Significantly, GPeval.py is better than kernelSVM.py / GPeval.py mean accuracy: 0.6293313679245283 / kernelSVM.py mean accuracy: 0.6044964622641509		

Logistic regression	Significantly, MultinomialLogRegression.py is better than adaBoost.py / adaBoost.py mean accuracy: 0.50723899371069 / MultinomialLogRegression.py mean accuracy: 0.5821650943396101	Significantly, decisionTree.py is better than MultinomialLogRegression.py / decisionTree.py mean accuracy: 0.6215231918238899 / MultinomialLogRegression.py mean accuracy: 0.58255542452831	Significantly, MultinomialLogRegression.py is better than DNNregressor.py / DNNregressor.py mean accuracy: 0.3644407292413399 / MultinomialLogRegression.py mean accuracy: 0.5828580974842898	Significantly, GPeval.py is better than MultinomialLogRegression.py / GPeval.py mean accuracy: 0.6299571540880503 / MultinomialLogRegression.py mean accuracy: 0.581620676100629	Significantly, kernelSVM.py is better than MultinomialLogRegression.py / kernelSVM.py mean accuracy: 0.60455699685534 / MultinomialLogRegression.py mean accuracy: 0.5808463050314498	
Random Forest	Significantly, randomForest.py is better than adaBoost.py / randomForest.py mean accuracy: 0.658026336478 / adaBoost.py mean accuracy: 0.506693003145	Significantly, randomForest.py is better than decisionTree.py / randomForest.py mean accuracy: 0.656595125786 / decisionTree.py mean accuracy: 0.624946147799	Significantly, randomForest.py is better than DNNregressor.py / randomForest.py mean accuracy: 0.6599154874213801 / DNNregressor.py mean accuracy: 0.36703287954784003	Significantly, randomForest.py is better than GPeval.py / GPeval.py mean accuracy: 0.6282688679245283 / randomForest.py mean accuracy: 0.6604795597484275	Significantly, randomForest.py is better than kernelSVM.py / randomForest.py mean accuracy: 0.659723663522 / kernelSVM.py mean accuracy: 0.604745676101	Significantly, randomForest.py is better than MultinomialLogRegression.py / randomForest.py mean accuracy: 0.65872759434 / MultinomialLogRegression.py mean accuracy: 0.581437106918

2. Efficiency

	Adaboost	Decision Tree	DNN regressor	Gaussian Process	Kernel SVM	Logistic regression
Decision Tree	Statistically, no difference detected. But in this sample, dtree_traintimeis slightly better. dtree_traintime mean train time: 0.008533930700000001					
DNN regressor	Statistically, no difference detected. But in this sample, adaboost_traintimeis slightly better. adaboost_traintime mean train time: 0.0733719349	Statistically, no difference detected. But in this sample, dtree_traintimeis slightly better. dtree_traintime mean train time: 0.008533930700000001				
Gaussian Process	Statistically, no difference detected. But in this sample, adaboost_traintimeis slightly better. adaboost_traintime mean train time: 0.0733719349	Statistically, no difference detected. But in this sample, dtree_traintimeis slightly better. dtree_traintime mean train time: 0.008533930700000001	Statistically, no difference detected. But in this sample, gp_traintimeis slightly better. gp_traintime mean train time: 2.2234712123			
Kernel SVM	Statistically, no difference detected. But in this sample, adaboost_traintimeis slightly better. adaboost_traintime mean train time: 0.0733719349	Statistically, no difference detected. But in this sample, dtree_traintimeis slightly better. dtree_traintime mean train time: 0.008533930700000001	Statistically, no difference detected. But in this sample, kernelsvm_traintimeis slightly better. kernelsvm_traintime mean train time: 0.0825884103	Statistically, no difference detected. But in this sample, kernelsvm_traintimeis slightly better. kernelsvm_traintime mean train time: 0.0825884103		
Logistic regression	Significantly, adaboost_traintime is better than lr_traintime / lr_traintime mean train time: 0.38272788519999995 / adaboost_traintime mean train time: 0.0733719349	Significantly, dtree_traintime is better than lr_traintime / lr_traintime mean train time: 0.38272788519999995 / dtree_traintime mean train time: 0.008533930700000001	Statistically, no difference detected. But in this sample, lr_traintimeis slightly better. lr_traintime mean train time: 0.38272788519999995	Statistically, no difference detected. But in this sample, lr_traintimeis slightly better. lr_traintime mean train time: 0.38272788519999995	Significantly, kernelsvm_traintime is better than lr_traintime / lr_traintime mean train time: 0.38272788519999995 / kernelsvm_traintime mean train time: 0.0825884103	
Random Forest	Significantly, rforest_traintime is better than adaboost_traintime / adaboost_traintime mean train time: 0.0733719349 / rforest_traintime mean train time: 0.0404434443	Statistically, no difference detected. But in this sample, dtree_traintimeis slightly better. dtree_traintime mean train time: 0.008533930700000001	Statistically, no difference detected. But in this sample, rforest_traintimeis slightly better. rforest_traintime mean train time: 0.0404434443	Statistically, no difference detected. But in this sample, rforest_traintimeis slightly better. rforest_traintime mean train time: 0.0404434443	Statistically, no difference detected. But in this sample, rforest_traintimeis slightly better. rforest_traintime mean train time: 0.0404434443	Significantly, rforest_traintime is better than lr_traintime / lr_traintime mean train time: 0.38272788519999995 / rforest_traintime mean train time: 0.0404434443

From the results, we can observe the train/test set accuracies and relative efficiency between each pair of models. All the result that showed significantly meaningful difference are colored in red. Here, we have 7 models and we are doing a T test for all possible pairs, both for training and test set. Based on the result, we have the following results:

1. Accuracy (From highest to lowest):

Random Forest > Gaussian Processes ≈ Decision Tree > Kernel SVM > Logistic Regression(Multi) > AdaBoost > Deep Neural Networks

2. Efficiency (From shortest time to longest):
Decision Tree < Random Forest < Adaboost < Kernel SVM < Logistic Regression(Multi) < Gaussian Processes ≈ Neural Networks < Deep Neural Networks

Conclusion / Lessons Learned

Although there were few statistically meaningful differences in efficiency among different models, even the most inefficient model did not take too long to train and test. Thus, we focused more on accuracy of the model. Then, every model can be ranked except for Gaussian Processes and Decision Tree. In this case, Decision Tree showed much higher efficiency than the Gaussian Processes.

Thus, the final model comparison based on the accuracy and efficiency is like the following:
Random Forest > Decision Tree > Gaussian Processes > Kernel SVM > Logistic Regression (Multi) > AdaBoost > Deep Neural Network
when we rank models from best to worst.

During the implementation and model comparisons, we learned following facts:

Some facts to note regarding Accuracy:

1. As our problem is a classification problem, the classification models (Random Forest, Gaussian Processes, Decision Tree, Kernel SVM) show better results than regression models (Logistic Regression, AdaBoost, Deep Neural Networks). This is partly due to the nature of the problem itself (which is classification), and we also lose information from the regression result when we transform the regression problem result to the classification model.
2. Random Forest shows better result than the Decision Tree as random pick of features decreases the innate dependence between different test and train sets in cross validation method. However, although AdaBoost is an improvement upon Decision Tree, it seems that the amount that AdaBoost decreases the overfitting nature of Decision Tree is less than the amount of information we lose from transforming regression nature of AdaBoost result into classification model. (At least with this data set.)
3. Deep Neural Network, especially the regressor model that we chose to implement, usually shows really high accuracy with the given scope of efficiency when there are more than a million samples in the data. Although we could change specifics in creating DNN model so that it becomes more accurate than all the other models we implemented, we could not do so with the minimal knowledge we have for the DNN models.
4. It looks like as our data is not a linearly separable dataset, non-parametric models such as Random Forest and Decision Tree shows the best results in terms of accuracy. Since Gaussian Process (GP) is considered nonparametric because a GP represents a function (i.e. an infinite dimensional vector), we can consider top three models are all non-parametric.
5. Then, for fourth and fifth places, we think Kernel SVM showed better performance than Logistic Regression because Kernel SVM uses kernel, so it is more flexible even with non-linearly-separable dataset such as ours.

6. Although AdaBoost is non-parametric, it is highly sensitive to noises, so it shows low performance with our dataset which contains high level of noise.

Some facts to note regarding Efficiency:

1. Ensemble Learning methods (Decision Tree, Random Forest, Adaboost) are significantly faster, both to train and test, than any other algorithms. Slight differences in time were detected within the 3 ensemble methods, and Decision Tree was the fastest to run.
2. Kernel SVM using RBF kernel (C=1, gamma=0.1) was surprisingly faster than Linear Classifier (Multinomial Logistic Regression). Possible cause is due to the fact that Multinomial Logistic Regression minimizes the multinomial loss fit across the entire probability distribution and uses a gradient descent solver that takes a long time.
3. GP and NN are significantly slower than all other ML algorithms. GP is faster than NN when training, and the opposite when testing. GP integrates a kernel and solving the kernel to make predictions will obviously take a long time. For Neural Networks, training time takes longer than GP due to forward and back propagation.
4. Deep Neural Networks has much more layers and nodes than the default NN model, and therefore it is the slowest in training.

Future Work

Our future work extends to improving our Deep Neural Networks algorithm. By integrating Bayesian Optimization to update our belief of the performance and choosing the next evaluation point with highest utility, we can acquire optimal number of nodes and layers to minimize our loss. Bayesian Optimization can be applied to any of our models that needs hyperparamter optimization, which includes Kernel SVM and Gaussian Processes.

Also, we could build a simple and flexible Graphical User Interface for this specific application so that people can easily use and benefit from this wine-quality application through GUI rather than running each model in terminal. We could also add visual aids for model comparison using GUI and give user better intuition about performance of each model.