Application Project Final Report

Machine Learning Spring 2018

Team: Kimcheese
Members:

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

GitHub: https://github.com/NigelKim/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

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--which we realize this is not optimal for our multiclass data--that increased the accuracies up to ~95%. However, we realized although this improves the accuracy, it 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. However, the accuracy was around 0.5409 and the possible reasoning behind this is due to our non-uniformly distributed output labels. Then, 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 was distinctly high with 0.9612. We observed logistic regression is generally well applied 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.

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 $(l = 1, \lambda = 1)$ and Matern Kernel$(l = 1, \lambda = 1, \nu = 1.5)$ yields very similar results, and for the final comparison with other ML models, RBF Kernel with 1-vs-1 multiclass GP classification algorithm was chosen.

(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.

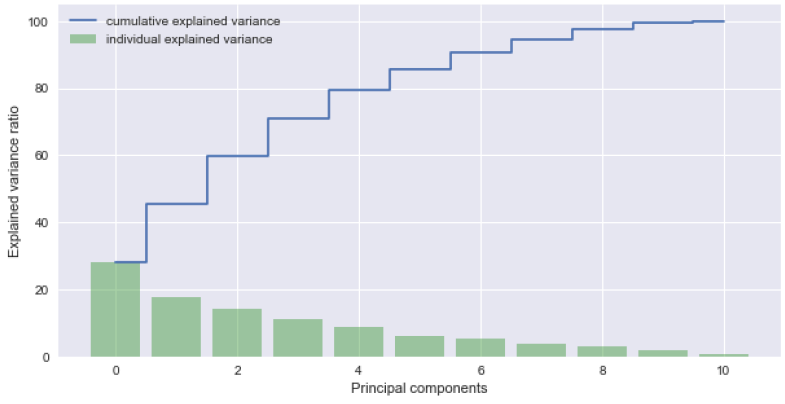
DNN Regressor

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.

Data Whitening



We also whitened our dataset to improve performance. This was conducted through k-means clustering. Since we have high-dimensional feature spaces and various ranges of values involved, we

normalize the data first. We chose the optimal k from the point where the marginal gain in explained variance drops. For extension of k-means clustering, we used GMM for in-depth analysis. Although it is simple and relatively easy to understand, the non-probabilistic nature of k-means and its use of simple distance-from-cluster-center to assign cluster membership leads to poor performance for many real-world situations. These disadvantages of k-means—its lack of flexibility in cluster shape and lack of probabilistic cluster assignment—can be generalized with GMM. It attempts to find a mixture of multi-dimensional Gaussian probability distributions that best model any input dataset. Trough experimenting with different number of clusters, we concluded that partitioning the dataset into 8 clusters can improve the performance the best. This data whitening was conducted to all of our dataset for all the models later on.

Comparison

To analyze and evaluate all of our models, we ran 10-reruns, calculated the average runtime for train time and test time, and compared each of the results to each other. 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 about the population for both samples.
2. We are not sure about population variances for both samples.
3. We are not sure whether two populations (for respective samples) have the same population variance.

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

```
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:

	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 relative efficiency between each pair of models. 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 conclude that strictly comparing the runtime of each algorithm, models can be ordered from least time to longer time:

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

Some facts to note from the analysis:

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.

Conclusion

In conclusion, by strictly comparing the models with respect to the efficiency of training and test time, ensemble learning is the most efficient followed by Kernel SVM with a small difference. Logistic Regression, Gaussian Processes, and Neural Network is relatively inefficient by purely comparing the runtimes.

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 hyperparameters to minimize our loss.