EE660 Final Project: Wine Quality Prediction

Wenzhen Gong wenzheng@usc.edu

12/02/2014

1. Abstract

Wine's quality is essential in the process of brewing and selling. However, it is time-consuming and costly to have every bottle of wine's quality assessed by human. So it is meaningful to develop a way that can predict the wine quality based on some of its chemical features. The features used in this problem are fixed acidity, volatile acidity, citric acid, residual sugar, chlorides, free sulfur dioxide, total sulfur dioxide, density, pH, sulphates, alcohol. All of them are real-valued. We have 1599 red wine samples and 4898 white wine samples. The value need to be predicted is wine quality, which is evaluated from 0 (worst) to 10 (best). Although the samples have all the quality as integers, it is wiser to treat the learning process as a regression problem since the predicted value might be hard to classify when it is around X.5.
My ways of training the model is to use :

(1) Basic linear regression
(2) Linear regression with regularizers (L1, L2, Elastic Net)
(3) Linear regression with regularizers over selected features
(4) Linear regression with regularizers over selected features using Recursive Feature Elimination
(5) Nonlinear regression (Logistic regression with L1 and L2 regularizers and Bayesian Ridge)
(6) Support vector machine

And my ways of measure the performance of each algorithm is to check the mean absolute deviation (MAD), mean squared error (MSE) and out of sample error (E_{out}). The final result turned out to be acceptable with different tolerance of error. Under some certain conditions, E_{out} can be lower than 10%.

2. Problem Statement and Goals

The problem here is to use the 11 chemical related features of the wine samples to predict their quality. And we have different choices of training algorithms (see part 1). Our goal is to get the best model among all of them. Our measurement of the performance of each model are basically MAD, MSE and E_{out}, which means that these three values will be calculated for each algorithm and be compared after the learning process is done.
This problem is interesting because firstly, itself has real life meaning. It can be adopted immediately by the wine producers so that their cost will be greatly lowered. Secondly, the data given here is not trivial. The 11-D feature vectors can be performed a feature reduction. And 1599 and 4898 data samples are enough to distinguish the performances among different algorithms. Most importantly, the data samples have nonlinear behavior as we can see in the implementation part.

3. Literature Review

Modeling wine preferences by data mining from physicochemical properties by Paulo

Cortez, António Cerdeira, Fernando Almeida, Telmo Matos, José Reis is the paper related to this problem. In the paper, authors talk about the ways of training data samples and their final results. They used linear regression (with regularizers), neural network and SVM to training the model and finally got some good results. However, they did not talk about the details of how they performed the learning process. In their paper, they claimed that for tolerance T = 0.25, 0.50 and 1.00, SVM has the best performance based on both MAD and accuracy (E_{out}).

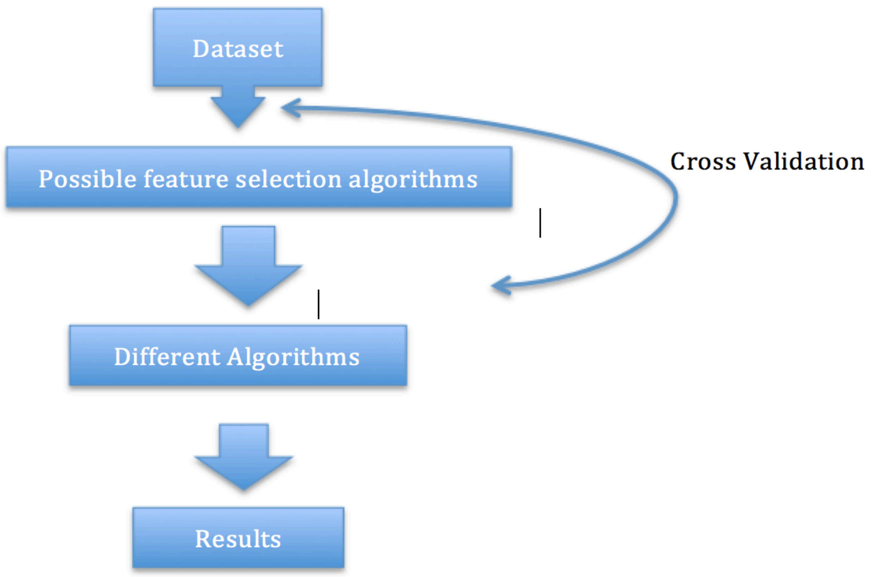
4. Prior and Related Work

Personally I do not have any Prior or related work for this problem.

5. Project Formulation and Setup

The algorithms are listed in Part 1. I am just changing some parameters and doing comparisons among different models, thus everything will be explained in the implementation part.

6. Methodology



The above is my flowchart for the training and prediction process. Firstly, I split the dataset into 3 sub-datasets with equal number of samples. Then I use these sub-datasets for cross validation in every algorithm. Finally, I will get the best model witch is the model with least out-of-sample MAD or MSE.

7. Implementation

7.1 Feature Space

Let us take a sample from training set to illustrate the feature space:

Example: $\underline{x}$ = {7.4; 0.7; 0; 1.9; 0.076; 11; 34; 0.9978; 3.51; 0.56; 9.4} y = 5

The feature space is created by the first to the 11th number (here 7.4 0.7 0 1.9 … 0.56 9.4). They are respectively
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 - sulphates
11 - alcohol
And the last number is wine quality (here 5), which is what we are predicting in this problem. All the feature values are real-valued. The wine qualities in the dataset are all integer numbers. However, since we are treating the problem as a regression problem, we allow the predicted value to be a real number.

7.2 Pre-processing and Feature Extraction
Since the data in this problem are all synthesized, I did not do any pre-processing or feature extraction on the data. However, two ways of feature dimensionality reduction (or feature selection) were performed in my project. The first one is called "SelectKBest" and the second one is called "Recursive Feature Elimination". SelectKBest basically use a score to evaluate each feature and then select the desired number of features with highest scores. Here we use chi-square as the score. Recursive Feature Elimination (RFE) is the process of assigning weights to each feature recursively and then eliminating the features with least importance.

7.3 Training Process
I used:

1. linear regression (with no regularizer, with L1, L2 and Elastic Net regularizers). The α for them are 0.2, 0.8 and 0.9 respectively.
2. Logistic regression (with L1 and L2 regularizers). The C for them are 0.8 and 0.8. Bayesian ridge, n_iter is 300.
3. Support vector machine. The kernels are linear and RBF.

The complexity for the hypothesis set is measured by its growth function $m_H(11) = 12$. Altogether I have 1599 red wine samples and 4898 white wine samples. I use 2/3 of each dataset, which is 1066 red wine samples and 3265 white wine samples as training set and the rest as test set.

With the VC generalization bound $E_{out} \le E_{in} + \sqrt{\frac{1}{2N}\ln\frac{2m_H(N)}{\delta}}$, we want to make sure that E_{out} and E_{in} do not differ too much so that there is no overfitting. So I chose the parameters of the regression methods so that $\sqrt{\frac{1}{2N}\ln\frac{2m_H(N)}{\delta}}$ is small with a certain and acceptable tolerance δ.

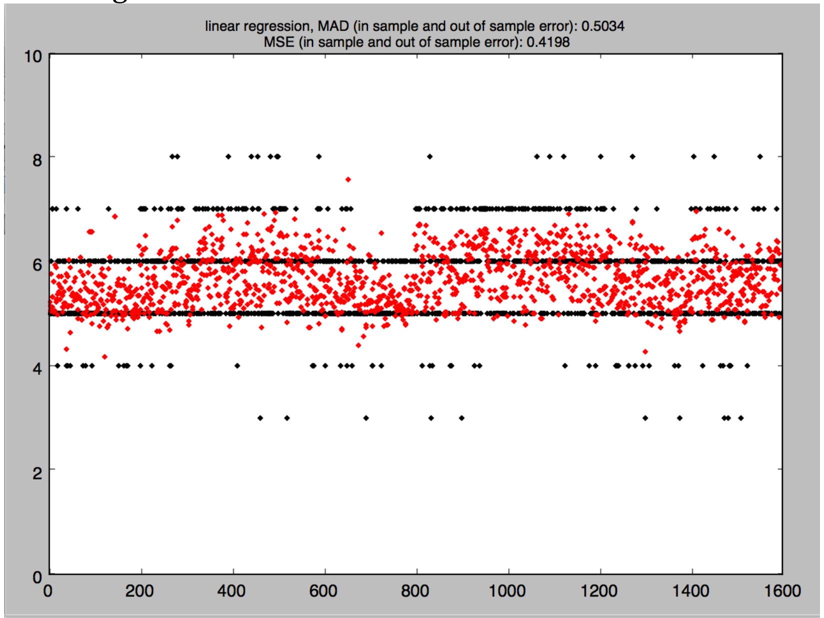
7.4 Testing, Validation and Model Selection

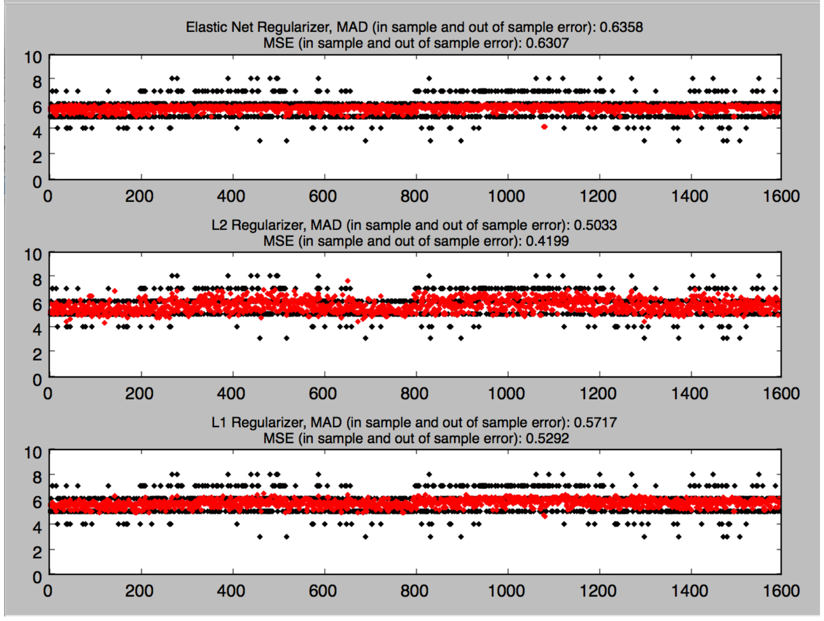
Since I have 1599 samples for red wine and 4898 samples for white wine, I am using a 3-fold cross validation to tune my model to achieve the best performance.

The following are the test results: (Black dots stands for real y, red dots stands for predicted y_hat)

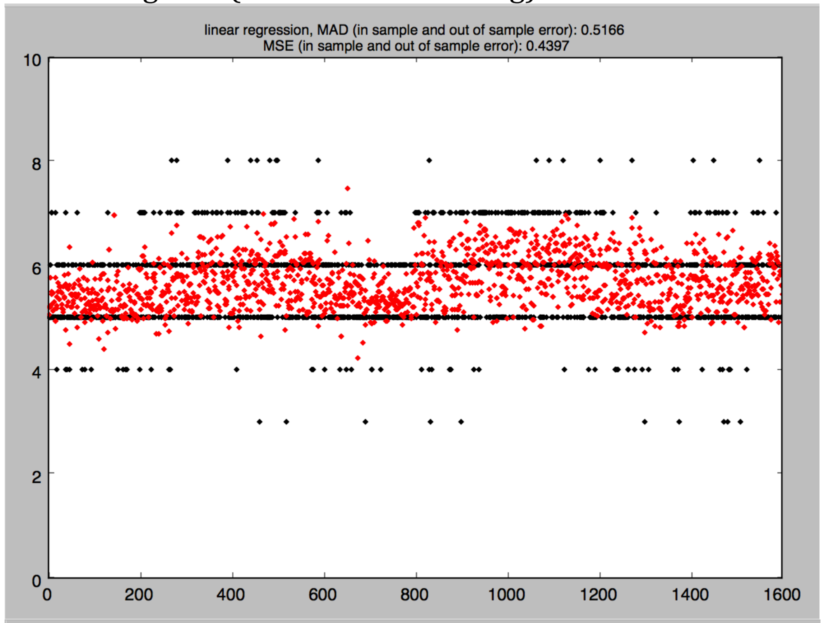
1.red wine different algorithms test results:

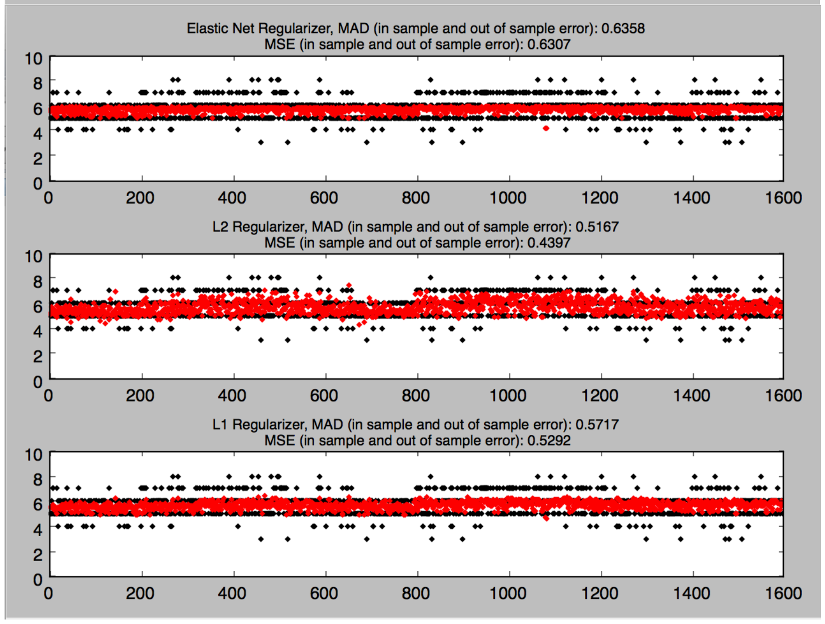
Basic regressions



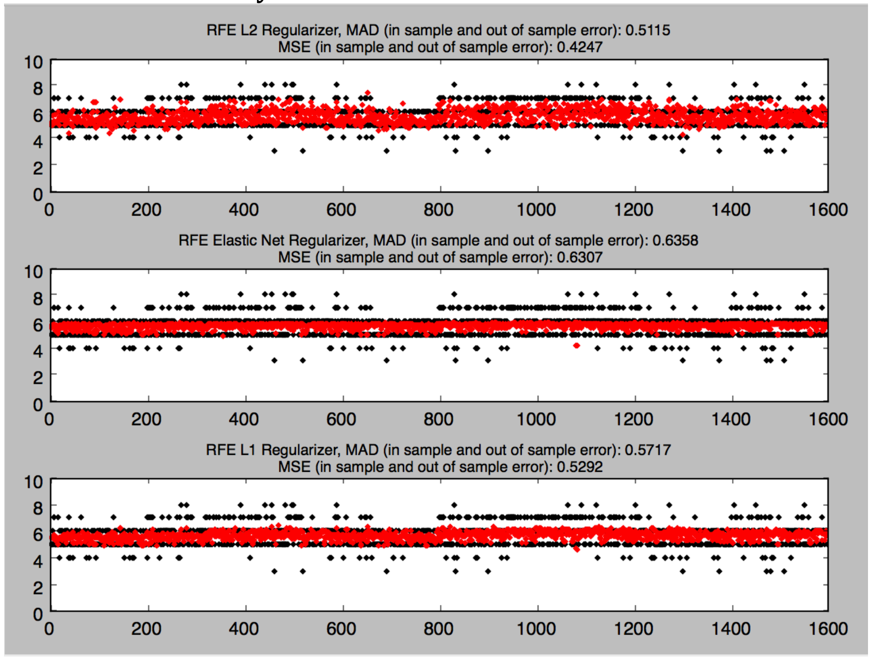


After doing PCA (6 features remaining)

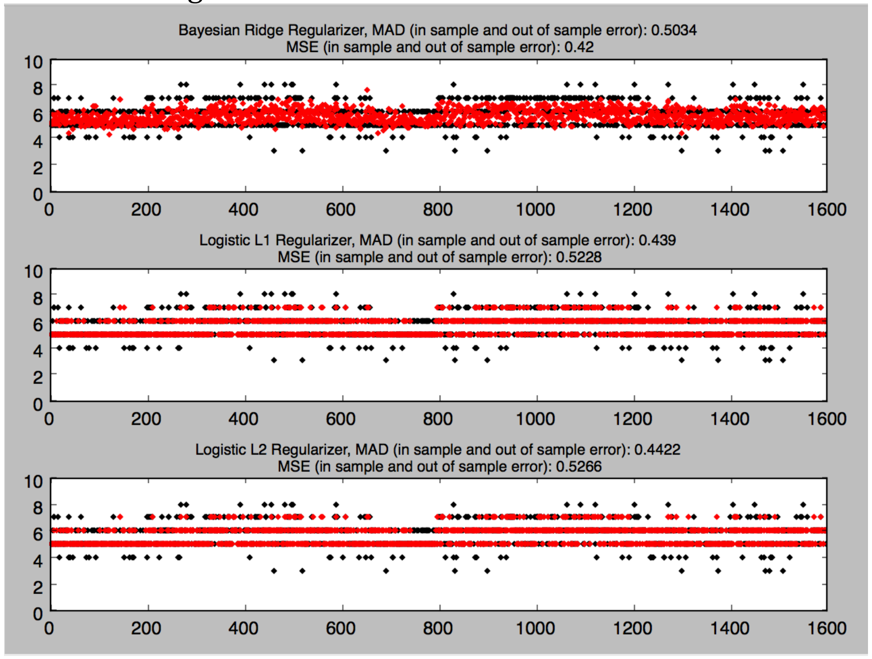




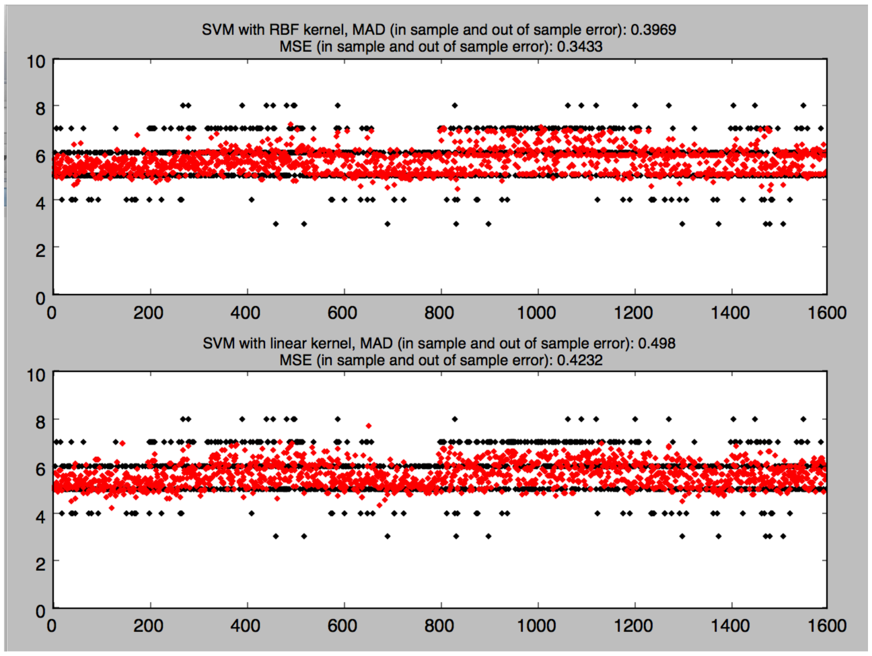
feature selection by Recursive Feature Elimination



Non-linear regressions

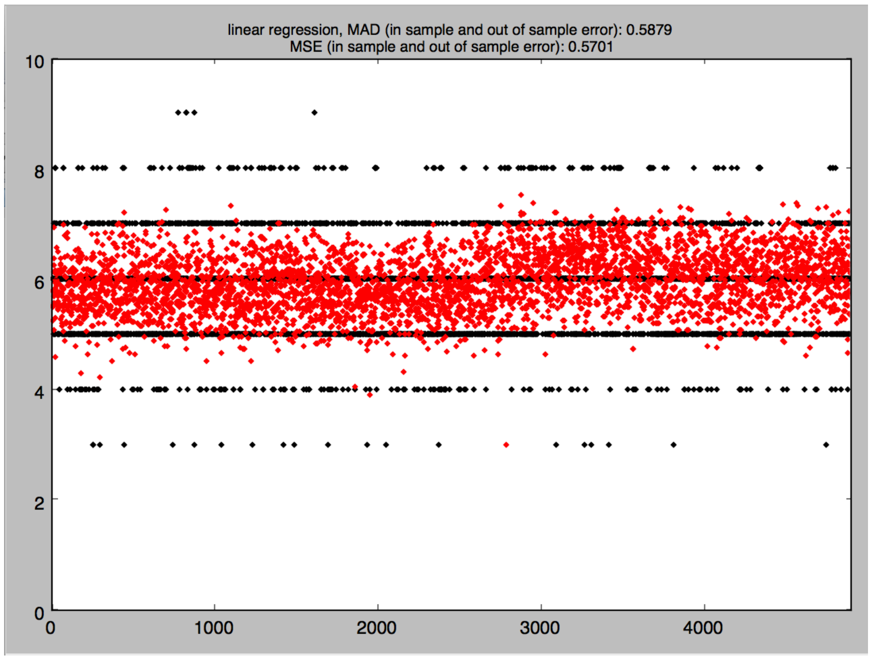


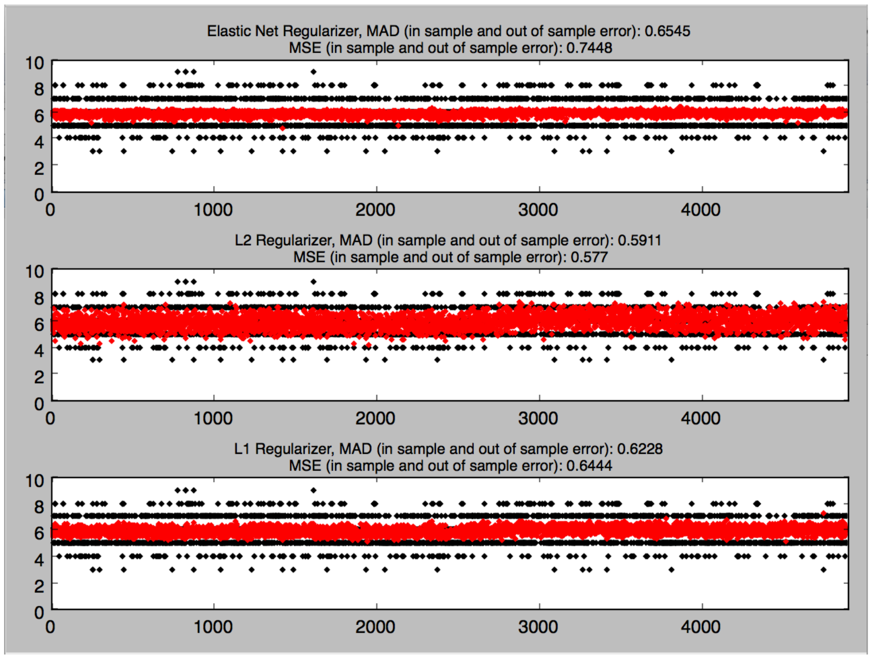
SVM



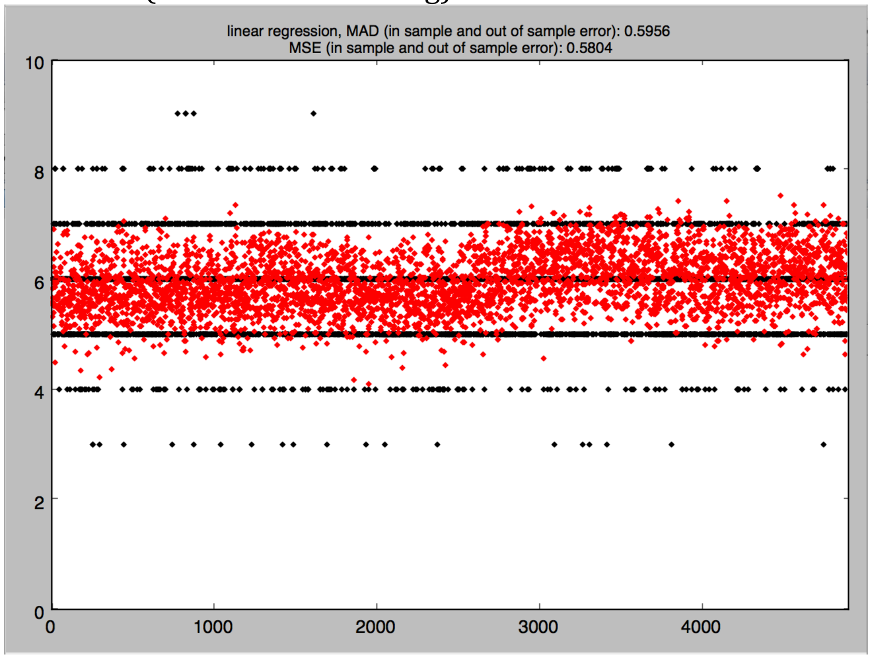
2.white wine different algorithms test results:

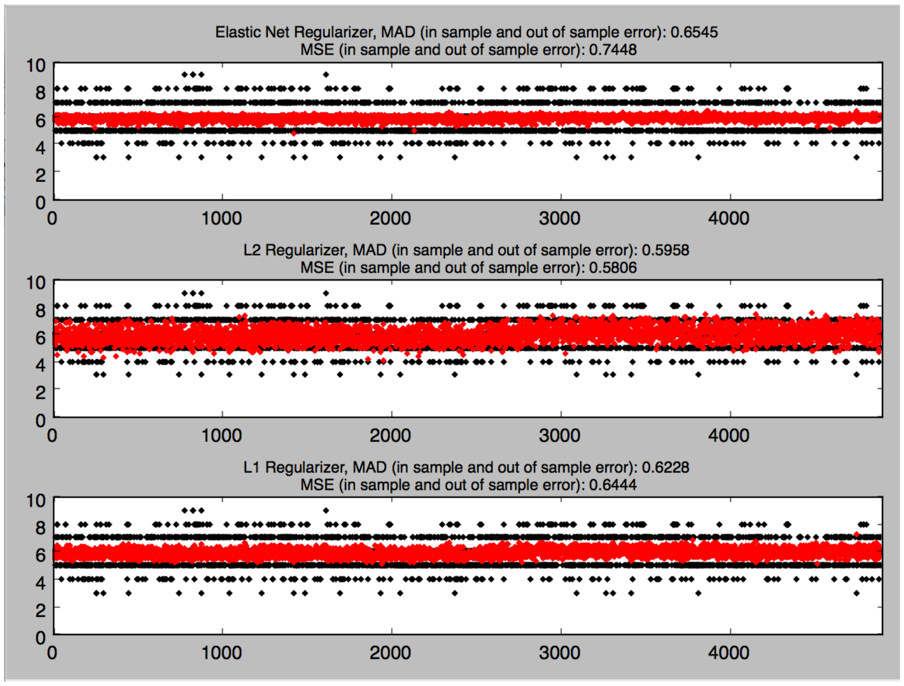
Basic regressions



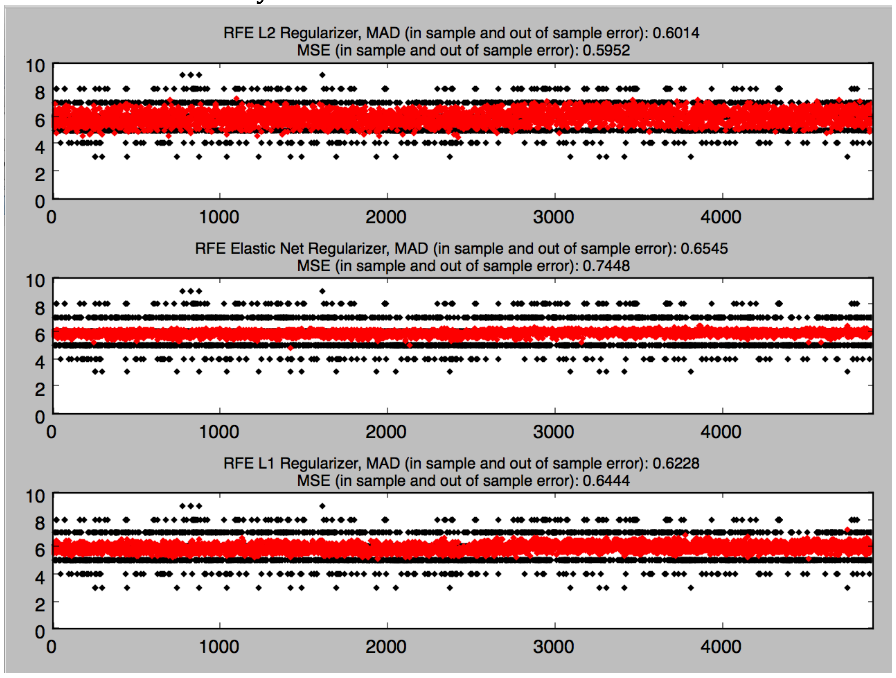


After PCA (6 features remaining)

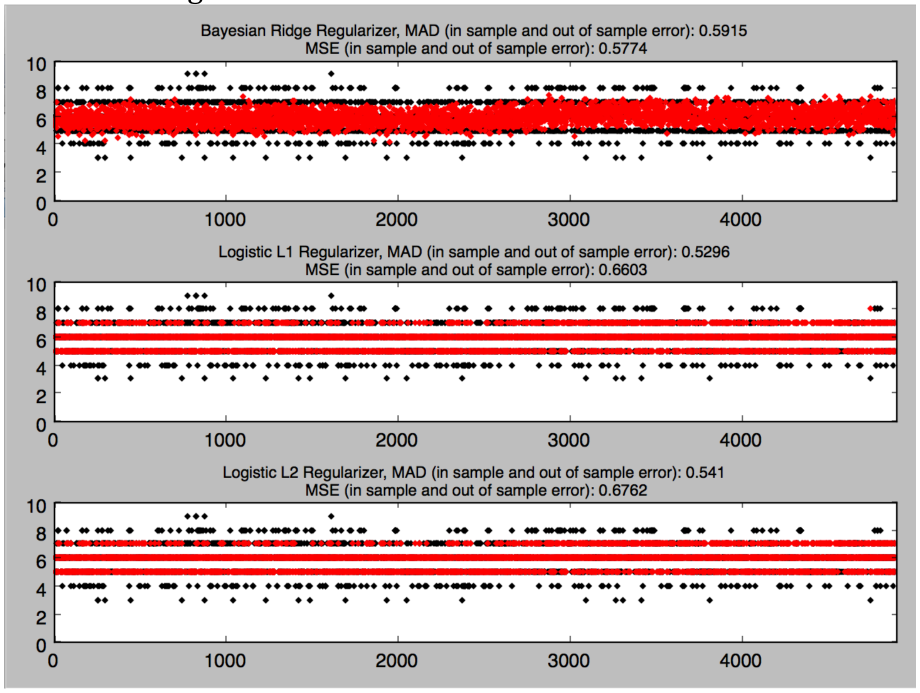




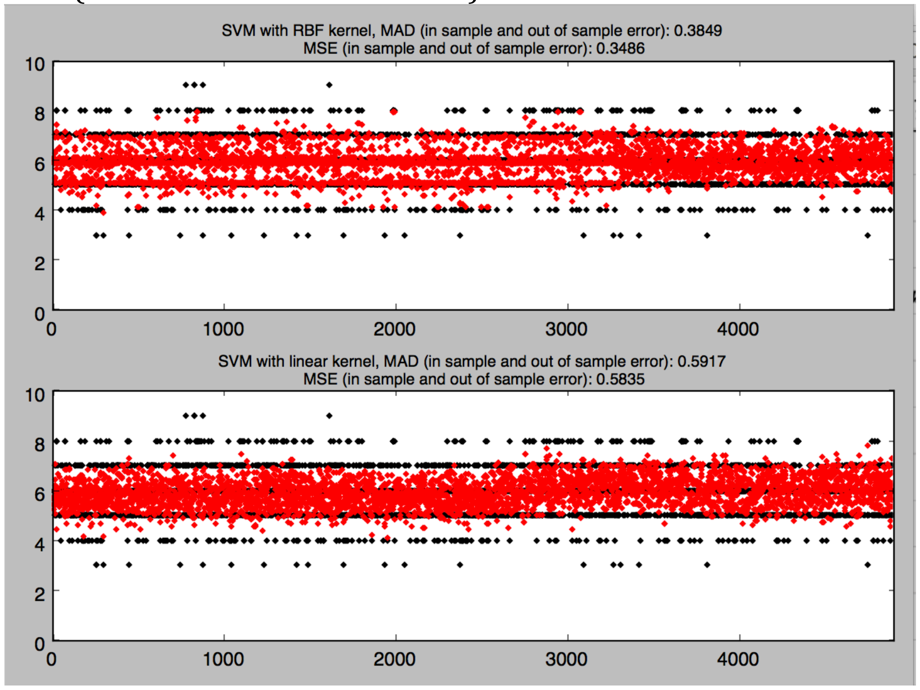
feature selection by Recursive Feature Elimination



Non-linear regressions



SVM (take more than 5 min to run)



The above are the results of comparison between raw value and predicted value. Though they seem like disorganized, we can still learn something from the whole chart. For example, when we do logistic regression with L1 or L2 regularizer, we can clearly see that some predicted values nearly perfectly hit the real values. However, the MAD or MSE of these two algorithms are not greatly smaller than other algorithms'. This fact is telling us that "if we care more about the percentage of accurately predicted value than the deviation when a wrong prediction happens, these algorithms could be better than others. But if the reverse is true, we should consider another algorithm maybe."

8. Final Results

Besides MAD and MSE, E_{out} is also an important indicator for the performance of an algorithm. E_{out} is very sensitive to tolerance. If the tolerance is too high, we cannot believe the prediction made by our models; If the tolerance is too low, it might be too strict with our models so that they can only give us a low accuracy. The following table gives us a clear evaluation at the performance by each model:

1. red wine

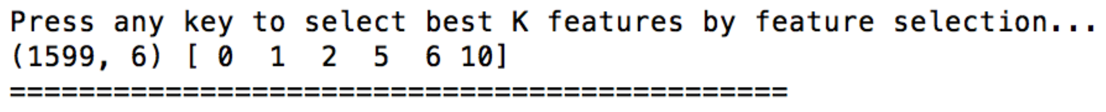
Model	MAD	MSE	E_{out}(tolerance=0.5)	E_{out}(tolerance=0.8)
LR (Linear Regression)	0.5034	0.4198	39.96%	**19.51%**
LR with L1	0.5717	0.5292	47.84%	22.70%
LR with L2	0.5033	0.4199	40.90%	**19.51%**
LR with Elastic Net	0.6358	0.6307	50.84%	22.33%
LR (6 features)	0.5166	0.4397	**37.90%**	22.33%
LR with L1 (6 features)	0.5717	0.5292	47.84%	22.70%
LR with L2 (6 features)	0.5167	0.4397	**37.90%**	22.33%
LR with Elastic Net (6 features)	0.6358	0.6307	50.84%	22.33%
RFE LR with L1	0.5717	0.5292	47.84%	22.70%
RFE LR with L2	0.5115	0.4247	41.65%	20.08%
RFE LR with Elastic Net	0.6358	0.6307	50.84%	22.33%
Logistic Regression with L1	0.4390	0.5228	39.59%	39.59%
Logistic Regression with L2	0.4422	0.5266	39.59%	39.59%
Bayesian Regression with Ridge	0.5034	0.4200	41.09%	**19.51%**
SVM with Linear Kernel	0.4980	0.4232	41.09%	20.26%
SVM with RBF Kernel	**0.3969**	**0.3433**	50.66%	24.77%

2. white wine

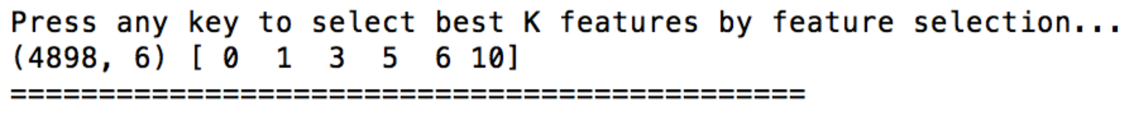
Model	MAD	MSE	E_{out}(tolerance=0.5)	E_{out}(tolerance=0.8)
LR (Linear Regression)	0.5879	0.5701	47.49%	**22.30%**
LR with L1	0.6228	0.6444	50.00%	25.25%
LR with L2	0.5911	0.5770	47.06%	22.98%
LR with Elastic Net	0.6545	0.7448	50.06%	39.64%
LR (6 features)	0.5956	0.5804	47.00%	23.65%
LR with L1 (6 features)	0.6228	0.6444	50.00%	25.25%
LR with L2 (6 features)	0.5958	0.5806	47.18%	23.41%
LR with Elastic Net (6 features)	0.6545	0.7448	50.06%	39.64%
RFE LR with L1	0.6228	0.6444	50.00%	56.06%
RFE LR with L2	0.6014	0.5952	46.32%	52.70%
RFE LR with Elastic Net	0.5917	0.5835	50.06%	64.48%
Logistic Regression with L1	0.5300	0.6652	**46.14%**	45.96%

Logistic Regression with L2	0.5414	0.6766	47.79%	47.92%
Bayesian Regression with Ridge	0.5915	0.5774	47.06%	23.10%
SVM with Linear Kernel	0.5917	0.5835	46.51%	22.61 %
SVM with RBF Kernel	**0.3849**	**0.3486**	51.59%	29.84%

Moreover, for red wine, feature number 0, 1, 2, 5, 6, 10 are the best 6 features. However, for white wine, feature number 0, 1, 3, 5, 6, 10 are the best 6 features. There is a tiny difference worth noticing.



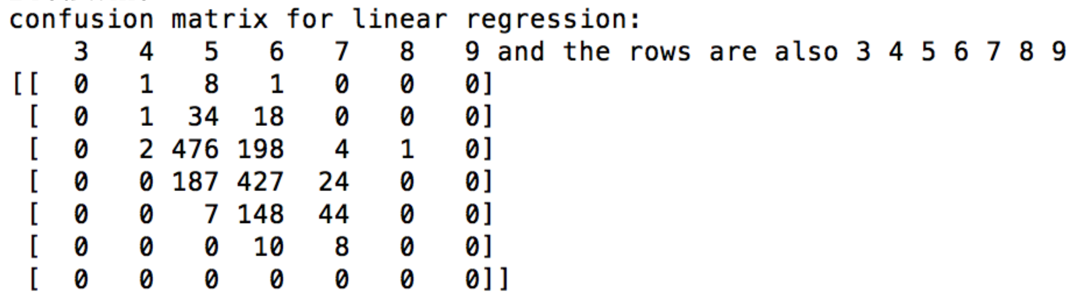
```
Press any key to select best K features by feature selection...
(1599, 6) [ 0  1  2  5  6 10]
============================================
```



```
Press any key to select best K features by feature selection...
(4898, 6) [ 0  1  3  5  6 10]
============================================
```

Here are the confusion matrices:

1.red wine



```
confusion matrix for linear regression:
    3   4   5   6   7   8   9 and the rows are also 3 4 5 6 7 8 9
[[  0   1   8   1   0   0   0]
 [  0   1  34  18   0   0   0]
 [  0   2 476 198   4   1   0]
 [  0   0 187 427  24   0   0]
 [  0   0   7 148  44   0   0]
 [  0   0   0  10   8   0   0]
 [  0   0   0   0   0   0   0]]
```

```
confusion matrix for L2 Regularizer:
    3   4   5   6   7   8   9 and the rows are also 3 4 5 6 7 8 9
[[  0   1   8   1   0   0   0]
 [  0   1  33  19   0   0   0]
 [  0   2 472 204   2   1   0]
 [  0   0 189 425  24   0   0]
 [  0   0   7 152  40   0   0]
 [  0   0   0  10   8   0   0]
 [  0   0   0   0   0   0   0]]
confusion matrix for L1 Regularizer:
    3   4   5   6   7   8   9 and the rows are also 3 4 5 6 7 8 9
[[  0   0   2   8   0   0   0]
 [  0   0  11  42   0   0   0]
 [  0   0 287 394   0   0   0]
 [  0   0  96 542   0   0   0]
 [  0   0   3 196   0   0   0]
 [  0   0   0  18   0   0   0]
 [  0   0   0   0   0   0   0]]
confusion matrix for Elastic Net Regularizer:
    3   4   5   6   7   8   9 and the rows are also 3 4 5 6 7 8 9
[[  0   0   0  10   0   0   0]
 [  0   0   7  46   0   0   0]
 [  0   0 210 471   0   0   0]
 [  0   0  76 562   0   0   0]
 [  0   2  14 183   0   0   0]
 [  0   0   2  16   0   0   0]
 [  0   0   0   0   0   0   0]]
```

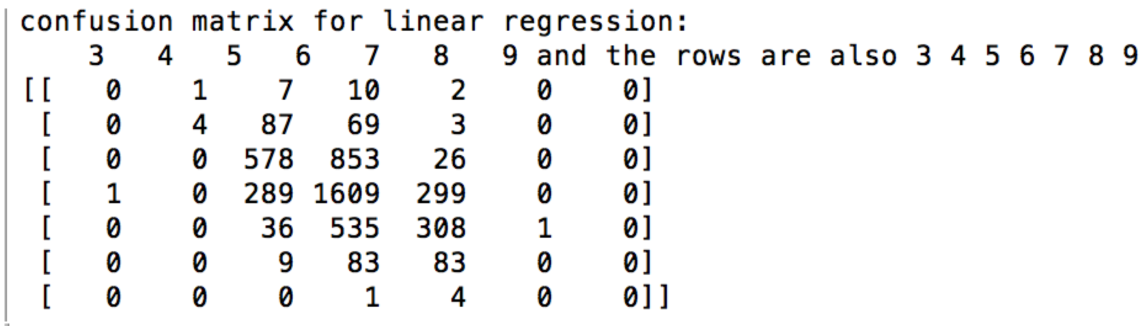
after feature selection

```
confusion matrix for linear regression:
    3   4   5   6   7   8   9 and the rows are also 3 4 5 6 7 8 9
[[  0   0   9   1   0   0   0]
 [  0   0  33  20   0   0   0]
 [  0   4 464 210   3   0   0]
 [  0   0 176 429  33   0   0]
 [  0   0   7 153  39   0   0]
 [  0   0   0  11   7   0   0]
 [  0   0   0   0   0   0   0]]
```

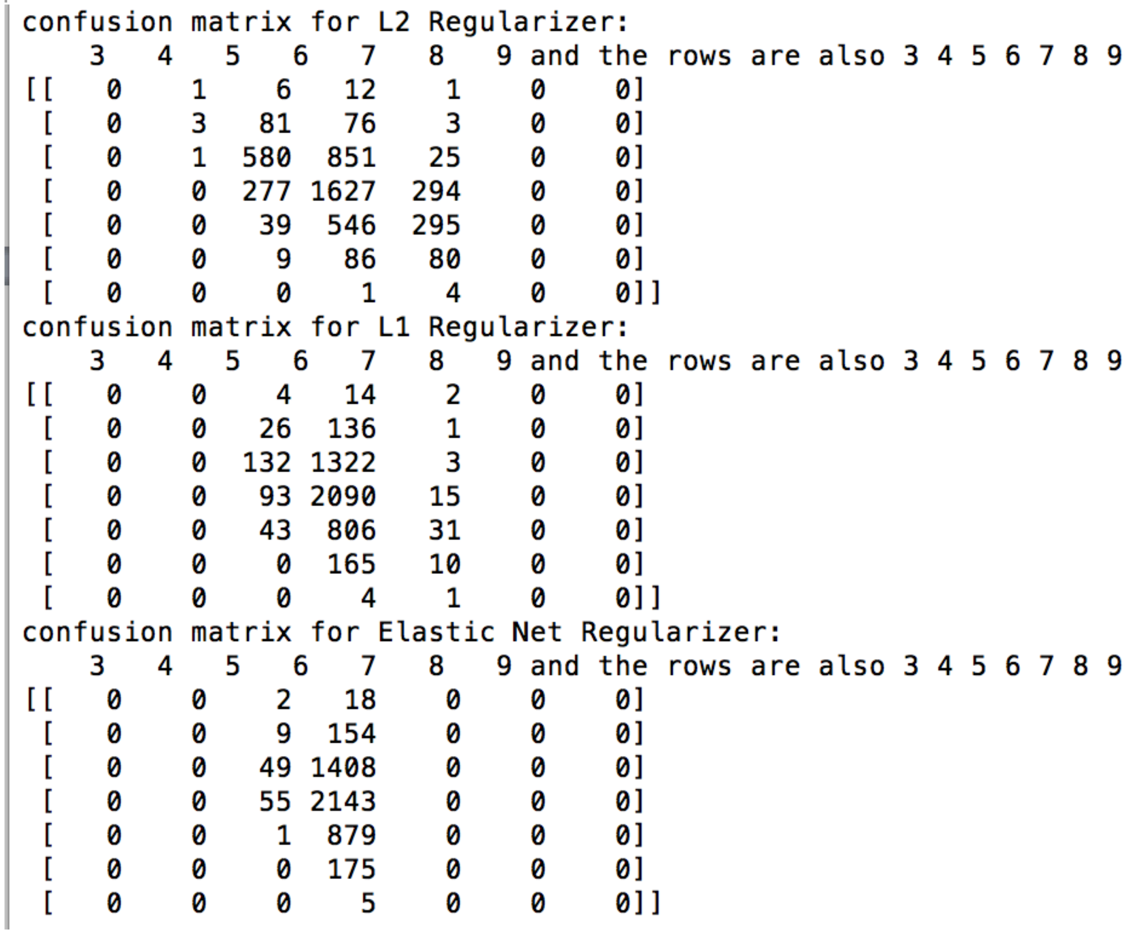
```
confusion matrix for L2 Regularizer:
    3   4   5   6   7   8   9 and the rows are also 3 4 5 6 7 8 9
[[  0   0   9   1   0   0   0]
 [  0   0  32  21   0   0   0]
 [  0   2 466 211   2   0   0]
 [  0   0 175 431  32   0   0]
 [  0   0   5 155  39   0   0]
 [  0   0   0  11   7   0   0]
 [  0   0   0   0   0   0   0]]
confusion matrix for L1 Regularizer:
    3   4   5   6   7   8   9 and the rows are also 3 4 5 6 7 8 9
[[  0   0   2   8   0   0   0]
 [  0   0  11  42   0   0   0]
 [  0   0 287 394   0   0   0]
 [  0   0  96 542   0   0   0]
 [  0   0   3 196   0   0   0]
 [  0   0   0  18   0   0   0]
 [  0   0   0   0   0   0   0]]
confusion matrix for Elastic Net Regularizer:
    3   4   5   6   7   8   9 and the rows are also 3 4 5 6 7 8 9
[[  0   0   0  10   0   0   0]
 [  0   0   7  46   0   0   0]
 [  0   0 210 471   0   0   0]
 [  0   0  76 562   0   0   0]
 [  0   2  14 183   0   0   0]
 [  0   0   2  16   0   0   0]
 [  0   0   0   0   0   0   0]]
---------------------------------------------
```

```
confusion matrix for RFE Elastic Net Regularizer:
    3   4   5   6   7   8   9 and the rows are also 3 4 5 6 7 8 9
[[  0   0   0  10   0   0   0]
 [  0   0   7  46   0   0   0]
 [  0   0 210 471   0   0   0]
 [  0   0  76 562   0   0   0]
 [  0   2  14 183   0   0   0]
 [  0   0   2  16   0   0   0]
 [  0   0   0   0   0   0   0]]
confusion matrix for RFE L1 Regularizer:
    3   4   5   6   7   8   9 and the rows are also 3 4 5 6 7 8 9
[[  0   0   2   8   0   0   0]
 [  0   0  11  42   0   0   0]
 [  0   0 287 394   0   0   0]
 [  0   0  96 542   0   0   0]
 [  0   0   3 196   0   0   0]
 [  0   0   0  18   0   0   0]
 [  0   0   0   0   0   0   0]]
confusion matrix for RFE L2 Regularizer:
    3   4   5   6   7   8   9 and the rows are also 3 4 5 6 7 8 9
[[  0   1   8   1   0   0   0]
 [  0   1  33  19   0   0   0]
 [  0   1 490 186   4   0   0]
 [  0   0 201 410  27   0   0]
 [  0   0   7 160  32   0   0]
 [  0   0   0  12   6   0   0]
 [  0   0   0   0   0   0   0]]
```

```
confusion matrix for Logistic L2 Regularizer:
    3   4   5   6   7   8   9 and the rows are also 3 4 5 6 7 8 9
[[  0   0   9   1   0   0   0]
 [  0   0  36  16   1   0   0]
 [  0   0 526 148   7   0   0]
 [  0   0 223 376  39   0   0]
 [  0   0  15 132  52   0   0]
 [  0   0   1   9   8   0   0]
 [  0   0   0   0   0   0   0]]
confusion matrix for Bayesian Ridge Regularizer:
    3   4   5   6   7   8   9 and the rows are also 3 4 5 6 7 8 9
[[  0   1   8   1   0   0   0]
 [  0   1  33  19   0   0   0]
 [  0   2 471 205   2   1   0]
 [  0   0 190 424  24   0   0]
 [  0   0   7 152  40   0   0]
 [  0   0   0  10   8   0   0]
 [  0   0   0   0   0   0   0]]
confusion matrix for Logistic L1 Regularizer:
    3   4   5   6   7   8   9 and the rows are also 3 4 5 6 7 8 9
[[  0   0   9   1   0   0   0]
 [  0   0  36  15   2   0   0]
 [  0   0 526 149   6   0   0]
 [  0   0 219 377  42   0   0]
 [  0   0  16 126  57   0   0]
 [  0   0   1   9   8   0   0]
 [  0   0   0   0   0   0   0]]
confusion matrix for SVM with linear kernel:
    3   4   5   6   7   8   9 and the rows are also 3 4 5 6 7 8 9
[[  0   0   9   1   0   0   0]
 [  0   1  34  18   0   0   0]
 [  0   2 495 182   1   1   0]
 [  0   0 212 405  21   0   0]
 [  0   0   7 157  35   0   0]
 [  0   0   0  11   7   0   0]
 [  0   0   0   0   0   0   0]]
confusion matrix for SVM with RBF kernel:
    3   4   5   6   7   8   9 and the rows are also 3 4 5 6 7 8 9
[[  0   2   5   3   0   0   0]
 [  0   1  36  16   0   0   0]
 [  0   0 535 145   1   0   0]
 [  0   0 130 499   9   0   0]
 [  0   0   8 104  87   0   0]
 [  0   0   0  13   5   0   0]
 [  0   0   0   0   0   0   0]]
```
2. white wine

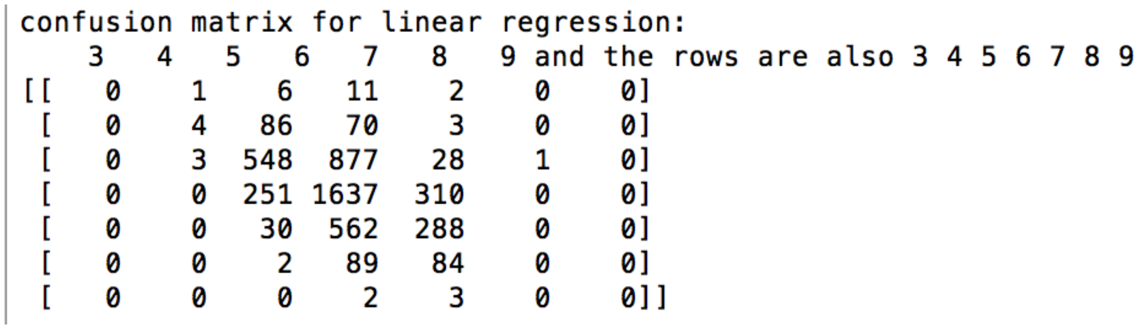


```
confusion matrix for linear regression:
    3    4    5    6    7    8    9 and the rows are also 3 4 5 6 7 8 9
[[   0    1    7   10    2    0    0]
 [   0    4   87   69    3    0    0]
 [   0    0  578  853   26    0    0]
 [   1    0  289 1609  299    0    0]
 [   0    0   36  535  308    1    0]
 [   0    0    9   83   83    0    0]
 [   0    0    0    1    4    0    0]]
```

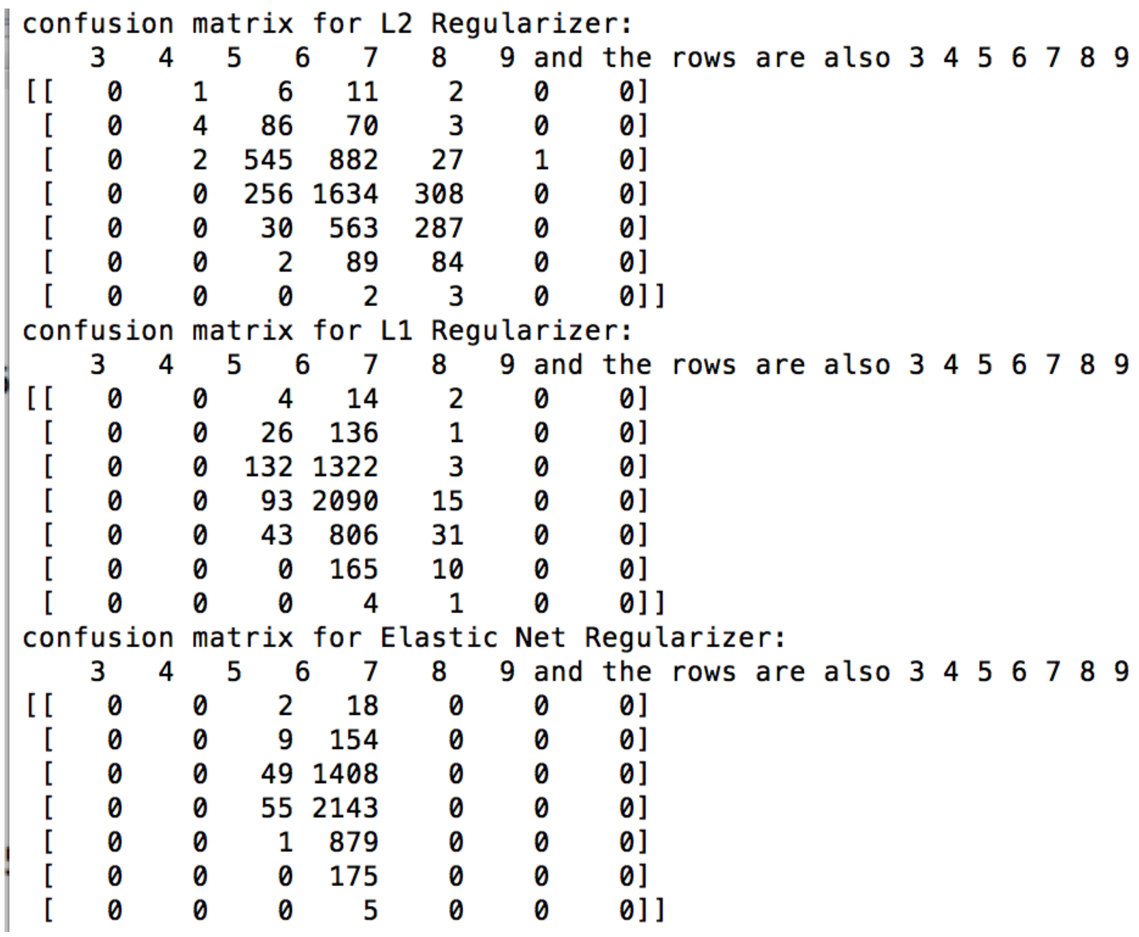


```
confusion matrix for L2 Regularizer:
    3    4    5    6    7    8    9 and the rows are also 3 4 5 6 7 8 9
[[   0    1    6   12    1    0    0]
 [   0    3   81   76    3    0    0]
 [   0    1  580  851   25    0    0]
 [   0    0  277 1627  294    0    0]
 [   0    0   39  546  295    0    0]
 [   0    0    9   86   80    0    0]
 [   0    0    0    1    4    0    0]]
confusion matrix for L1 Regularizer:
    3    4    5    6    7    8    9 and the rows are also 3 4 5 6 7 8 9
[[   0    0    4   14    2    0    0]
 [   0    0   26  136    1    0    0]
 [   0    0  132 1322    3    0    0]
 [   0    0   93 2090   15    0    0]
 [   0    0   43  806   31    0    0]
 [   0    0    0  165   10    0    0]
 [   0    0    0    4    1    0    0]]
confusion matrix for Elastic Net Regularizer:
    3    4    5    6    7    8    9 and the rows are also 3 4 5 6 7 8 9
[[   0    0    2   18    0    0    0]
 [   0    0    9  154    0    0    0]
 [   0    0   49 1408    0    0    0]
 [   0    0   55 2143    0    0    0]
 [   0    0    1  879    0    0    0]
 [   0    0    0  175    0    0    0]
 [   0    0    0    5    0    0    0]]
```

after feature selection

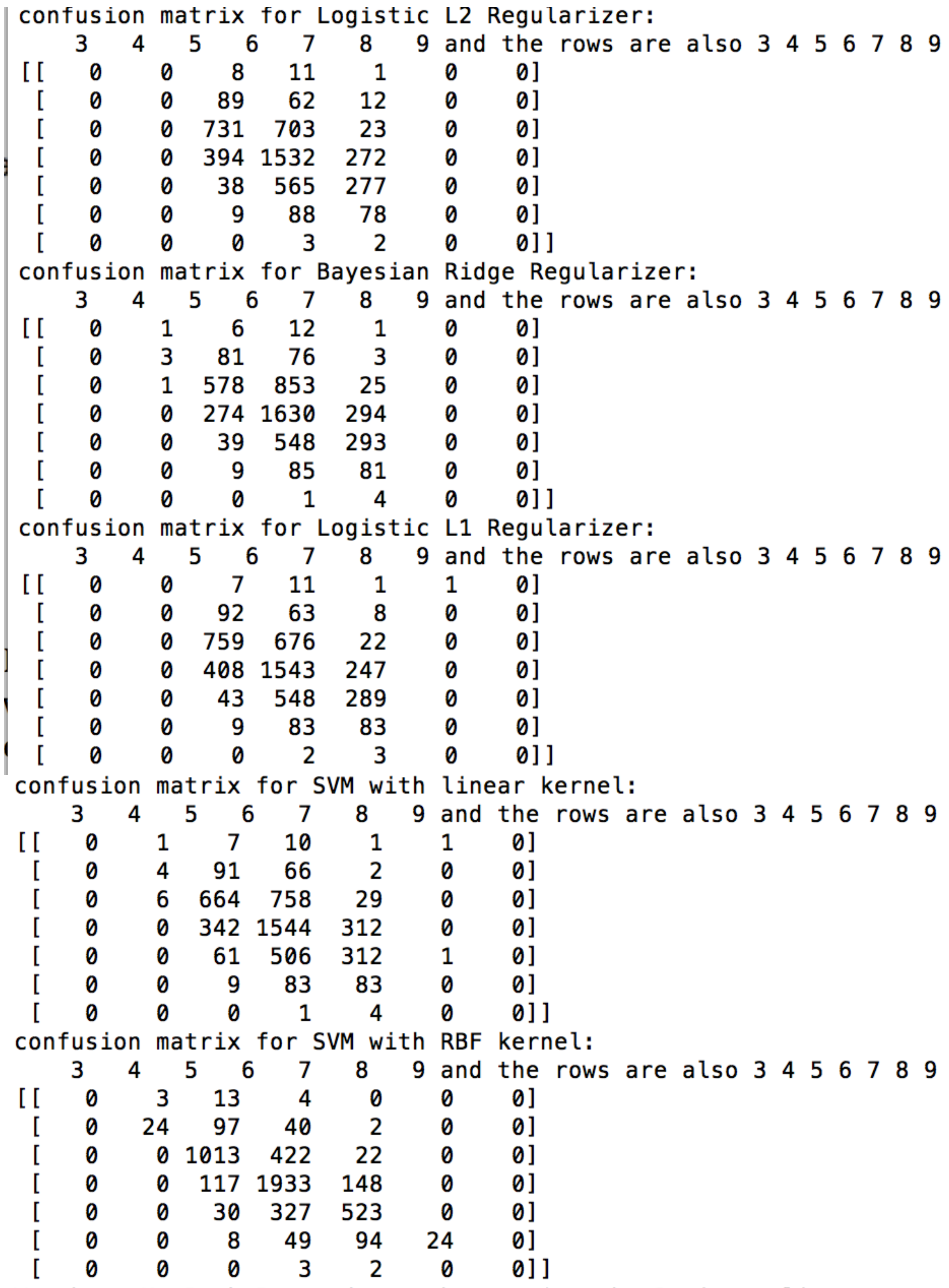


```
confusion matrix for linear regression:
    3    4    5    6    7    8    9 and the rows are also 3 4 5 6 7 8 9
[[   0    1    6   11    2    0    0]
 [   0    4   86   70    3    0    0]
 [   0    3  548  877   28    1    0]
 [   0    0  251 1637  310    0    0]
 [   0    0   30  562  288    0    0]
 [   0    0    2   89   84    0    0]
 [   0    0    0    2    3    0    0]]
```



```
confusion matrix for L2 Regularizer:
    3   4   5   6   7   8   9 and the rows are also 3 4 5 6 7 8 9
[[   0    1    6   11    2    0    0]
 [   0    4   86   70    3    0    0]
 [   0    2  545  882   27    1    0]
 [   0    0  256 1634  308    0    0]
 [   0    0   30  563  287    0    0]
 [   0    0    2   89   84    0    0]
 [   0    0    0    2    3    0    0]]
confusion matrix for L1 Regularizer:
    3   4   5   6   7   8   9 and the rows are also 3 4 5 6 7 8 9
[[   0    0    4   14    2    0    0]
 [   0    0   26  136    1    0    0]
 [   0    0  132 1322    3    0    0]
 [   0    0   93 2090   15    0    0]
 [   0    0   43  806   31    0    0]
 [   0    0    0  165   10    0    0]
 [   0    0    0    4    1    0    0]]
confusion matrix for Elastic Net Regularizer:
    3   4   5   6   7   8   9 and the rows are also 3 4 5 6 7 8 9
[[   0    0    2   18    0    0    0]
 [   0    0    9  154    0    0    0]
 [   0    0   49 1408    0    0    0]
 [   0    0   55 2143    0    0    0]
 [   0    0    1  879    0    0    0]
 [   0    0    0  175    0    0    0]
 [   0    0    0    5    0    0    0]]
```

```
confusion matrix for RFE Elastic Net Regularizer:
    3   4   5   6   7   8   9 and the rows are also 3 4 5 6 7 8 9
[[    0     0     2    18     0     0     0]
 [    0     0     9   154     0     0     0]
 [    0     0    49  1408     0     0     0]
 [    0     0    55  2143     0     0     0]
 [    0     0     1   879     0     0     0]
 [    0     0     0   175     0     0     0]
 [    0     0     0     5     0     0     0]]
confusion matrix for RFE L1 Regularizer:
    3   4   5   6   7   8   9 and the rows are also 3 4 5 6 7 8 9
[[    0     0     4    14     2     0     0]
 [    0     0    26   136     1     0     0]
 [    0     0   132  1322     3     0     0]
 [    0     0    93  2090    15     0     0]
 [    0     0    43   806    31     0     0]
 [    0     0     0   165    10     0     0]
 [    0     0     0     4     1     0     0]]
confusion matrix for RFE L2 Regularizer:
    3   4   5   6   7   8   9 and the rows are also 3 4 5 6 7 8 9
[[    0     0     4    16     0     0     0]
 [    0     0    72    89     2     0     0]
 [    0     1   645   793    18     0     0]
 [    0     0   357  1608   233     0     0]
 [    0     0    62   590   228     0     0]
 [    0     0    15    97    63     0     0]
 [    0     0     0     1     4     0     0]]
```



```
confusion matrix for Logistic L2 Regularizer:
    3    4    5    6    7    8    9 and the rows are also 3 4 5 6 7 8 9
[[   0    0    8   11    1    0    0]
 [   0    0   89   62   12    0    0]
 [   0    0  731  703   23    0    0]
 [   0    0  394 1532  272    0    0]
 [   0    0   38  565  277    0    0]
 [   0    0    9   88   78    0    0]
 [   0    0    0    3    2    0    0]]
confusion matrix for Bayesian Ridge Regularizer:
    3    4    5    6    7    8    9 and the rows are also 3 4 5 6 7 8 9
[[   0    1    6   12    1    0    0]
 [   0    3   81   76    3    0    0]
 [   0    1  578  853   25    0    0]
 [   0    0  274 1630  294    0    0]
 [   0    0   39  548  293    0    0]
 [   0    0    9   85   81    0    0]
 [   0    0    0    1    4    0    0]]
confusion matrix for Logistic L1 Regularizer:
    3    4    5    6    7    8    9 and the rows are also 3 4 5 6 7 8 9
[[   0    0    7   11    1    1    0]
 [   0    0   92   63    8    0    0]
 [   0    0  759  676   22    0    0]
 [   0    0  408 1543  247    0    0]
 [   0    0   43  548  289    0    0]
 [   0    0    9   83   83    0    0]
 [   0    0    0    2    3    0    0]]
confusion matrix for SVM with linear kernel:
    3    4    5    6    7    8    9 and the rows are also 3 4 5 6 7 8 9
[[   0    1    7   10    1    1    0]
 [   0    4   91   66    2    0    0]
 [   0    6  664  758   29    0    0]
 [   0    0  342 1544  312    0    0]
 [   0    0   61  506  312    1    0]
 [   0    0    9   83   83    0    0]
 [   0    0    0    1    4    0    0]]
confusion matrix for SVM with RBF kernel:
    3    4    5    6    7    8    9 and the rows are also 3 4 5 6 7 8 9
[[   0    3   13    4    0    0    0]
 [   0   24   97   40    2    0    0]
 [   0    0 1013  422   22    0    0]
 [   0    0  117 1933  148    0    0]
 [   0    0   30  327  523    0    0]
 [   0    0    8   49   94   24    0]
 [   0    0    0    3    2    0    0]]
```

9. Interpretation

As we can see from the data above:

For red wine, if we care most about MAD or MSE, SVM with RBF kernel will be the best choice to train the data. However, if we care about E_{out}, with tolerance = 0.5, Linear Regression with reduced feature and Linear Regression with reduced feature and with L2 regularizer will be the best choices. When tolerance = 0.8, Linear Regression, Linear Regression with L2 regularizer and Bayesian Regression with Ridge regularizer are the best choices. However, I would say that considering both tolerances = 0.5 or 0.8, Linear Regression will be the best choice because when tolerance = 0.5, its E_{out} = 39.96%, which is not far away from the smallest value 37.90%. But for LR with L2 and Bayesian with Ridge, both E_{out} when tolerance = 0.5 are much larger than 37.90%. If we look at the data from E_{out} when tolerance = 0.5, same conclusion will be drawn since when tolerance = 0.8, E_{out} for LR (6 features) and LR with L2 (6 features) will be a little bit larger than (39.96-37.90)% = 2.06%.

For red wine, if we care most about MAD or MSE, SVM with RBF kernel will be the best choice to train the data. However, if we care about E_{out}, with tolerance = 0.5, Logistic Regression with L1 regularizer will be the best choices. When tolerance = 0.8, Linear Regression is the best choices. However, I would say that considering both tolerances = 0.5 or 0.8, Linear Regression will be the best choice because when tolerance = 0.5, its E_{out} = 47.49%, which is not far away from the smallest value 46.14%. If we look at the data from E_{out} when tolerance = 0.5, same conclusion will be drawn since when tolerance = 0.8, E_{out} for Logistic Regression with L1 45.96%, which is much higher than the smallest value 22.30%.

Then let us talk about the best features.

For red wine, the best features are 1 - fixed acidity, 2 - volatile acidity, 3 - citric acid, 6 - free sulfur dioxide, 7 - total sulfur dioxide, 11 – alcohol.

For white wine, the best features are 1 - fixed acidity, 2 - volatile acidity, 4 - residual sugar , 6 - free sulfur dioxide, 7 - total sulfur dioxide, 11 – alcohol.

10. Summary and Conclusions

Several conclusions can be inferred from the training and testing process:

1. The best model can be different when we have different requirements on the performance. In this problem, if we only care about the deviation or squared error, SVM with RBF kernel will definitely be the best model since both MSE and MAD are very small. However, if we only care about the out-of-sample error, the classic Linear Regression works best. Both red wine and white wine have the same situation here.
2. Red wine and white wine have different attributes. Basically the best chemical features are the same, however, citric acid is more important for red wine while residual sugar is more important for white wine. Luckily, the difference of best features dose not affect our final best models since the best model does not drop in the category of any regression after feature selection from both criteria mentioned in 1.

3. My findings are consistent with the findings in the paper, since they only used MAD as error metric and they also got SVM regression as best model, though they did not mention which kernel they used.

There is something I should talk about. At the first glance of the problem people might think it is a classification problem. However, I treated it as a regression problem because I think if we can get a float number as a predicted value, it will be much closer to the real case. For example, if I got a predicted value as 7.9, it is reasonable to classify it into 8; However, if I got a predicted value as 7.6, it is awkward to classify it into 8 since it has a comparatively large deviation with 8. So I think regression is better for this problem because it reflect a more authentic, reliable and objective situation. However, the confusion matrices have to be performed as a classification problem, there is simply no other choice.

11. Reference

1. Paulo Cortez, António Cerdeira, Fernando Almeida, Telmo Matos, José Reis. *Modeling wine preferences by data mining from physicochemical properties*. Department of Information Systems/R&D Centre Algoritmi, University of Minho, 2009.

12. Code

Note: Download winequality-red.csv and winequality-white.csv from https://archive.ics.uci.edu/ml/datasets/Wine+Quality. Put both files and the code together. Make sure you are using python 3.4.2. Run python3 winequality.py. Hit enter when you are asked to move on. Close the pop window to continue.

```
import matplotlib.pyplot as plt #for simplicity

import numpy

import csv

import copy

from sklearn.metrics import confusion_matrix, mean_absolute_error,
mean_squared_error

from sklearn.cross_validation import KFold

from sklearn.linear_model import LinearRegression # basic linear regression

from sklearn.linear_model import Lasso # L1 regularizer

from sklearn.linear_model import Ridge # L2 regularizer

from sklearn.linear_model import ElasticNet # Linear regression with combined L1
```

```
and L2 priors as regularizer

from sklearn.linear_model import LogisticRegression # nonlinear regression

from sklearn.linear_model import BayesianRidge

from sklearn.svm import SVR  # SVM regression

from sklearn.feature_selection import SelectKBest, chi2, RFE


# Read data from csv

# Parameter: csv filename

# Returns: feature matrix (X in class notation) and quality vector (y in class
notation)

def read_data(filename):

    csvfile = open(filename, 'r')

    spamreader = csv.reader(csvfile, delimiter=';')

    D = []

    for row in spamreader:

        D.append(row)

    csvfile.close()

    return ((numpy.array(D)[1:,:-1]).astype(float), (numpy.array(D)[1:, -
1]).astype(float))

    # 1: means 1st row to last row, :-1 means 0th col to second last col, which are
features, -1 means last col, which is quality


# K-fold cross validation

# Paramter: X: feature matrix, y: quatity vector, k: number of fold, let default value =
3

# Returns: training set and test set

def get_kfold_train_test(X, y, k = 3):
    # len(X) is number of samples

    kf = KFold(len(X), k)

    for train, test in kf:

        # training sets and test sets

        yield X[train], y[train], X[test], y[test]


# mean absolute deviation

# Parameter: pred: prediction value(y_hat in class notation), actual: actual
value(y_test in class notation)

# Returns: mean absolute deviation(a float value)

def mean_absolute_dev(pred, actual):

    return mean_absolute_error(actual, pred)


# mean squared error

# Parameter: pred: prediction value(y_hat in class notation), actual: actual
value(y_test in class notation)

# Returns: mean squared error(a float value)

def mean_squared_err(pred, actual):

    return mean_squared_error(actual, pred)


# print confusion matrix

# Parameter: pred: prediction vector(y_hat in class notation), actual: actual
vector(y_test in class notation), these vector should be int type

def print_confusion_matrix(actual, pred):

    # get confusion matrix (we dont have sample whose quality is <=3 or >=9, but we
still keep 3 and 9 but neglect 0, 1, 2 and 10 to save memory)

    lb = range(3, 10)
    cm = confusion_matrix(actual, numpy.round(pred), lb)

    print('    3   4   5   6   7   8   9 and the rows are also 3 4 5 6 7 8 9')

    print(cm)


# plot result

# actual_list: actual value lists, preds_list: predicted value lists, name_list: names of the lists

def plot_result(actual_list, preds_list, name_list):

    # number of plots in a window(typically 1, sometimes 2 or 3 when we want to compare)

    pn = len(actual_list)

    # number of values in a list

    x = range(len(actual_list[0]))

    fig = plt.figure()

    for i in range(pn):

        plt.subplot(pn, 1, i+1)

        plt.title(name_list[i], fontsize = 10)

        plt.plot(x[:len(actual_list[0])], actual_list[i][:len(actual_list[0])], 'k.')

        plt.plot(x[:len(actual_list[0])], preds_list[i][:len(actual_list[0])], 'r.')

        plt.axis([0, len(actual_list[0]), 0, 10])

    plt.tight_layout()

    plt.show()


# select K best features(PCA)

# Returns: new X and the best features index

def select_KBest(X, y, k = 6):
    selector = SelectKBest(chi2, k) # use chi squared statistic for each class/feature
combination to select 6 best features

    X_new = selector.fit_transform(X, y)

    index = selector.get_support(True) # we want to show which features are left

    return X_new, index



# K-fold cross validation regression prototype

# Parameters: estimators: a vector of tuple(s) whose struction is/are
[('name_of_predictor1', predictor1),('name_of_predictor2', predictor2),...], X: feature
matrix, y: quatity vector

# Returns: a dictionary with keys as the names of estimators, values as
(prediction_vector_of_all_samples, corresponding_MAD)

def make_KfoldCV_regression(estimators, X, y):

    # return value, a dictionary

    ets = {}

    for name, estimator in estimators:

        # select best model with minimum error (et1, er1 for MAD, et2, er2 for MSE)

        et1 = None

        et2 = None

        er1 = None

        er2 = None

        finaly_test = None #used to calculate E_out

        finalX_test = None

        for X_train, y_train, X_test, y_test in get_kfold_train_test(X, y):

            # do training

            estimator.fit(X_train, y_train)

            # get prediction vector
            preds = estimator.predict(X_test)

            # calculate MAD, MSE (out of sample)

            error1 = mean_absolute_dev(preds, y_test)

            error2 = mean_squared_err(preds, y_test)

            # select the best training set and test set, i.e select the best estimator, and the
corresponding MAD and MSE (out of sample)

            if er1 is None:

                et1 = copy.deepcopy(estimator)

                er1 = error1

                finaly_test = copy.deepcopy(y_test)

                finalX_test = copy.deepcopy(X_test)

            else:

                if error1 < er1:

                    et1 = copy.deepcopy(estimator)

                    er1 = error1

                    finaly_test = copy.deepcopy(y_test)

                    finalX_test = copy.deepcopy(X_test)

            if er2 is None:

                et2 = copy.deepcopy(estimator)

                er2 = error2

            else:

                if error2 < er2:

                    et2 = copy.deepcopy(estimator)

                    er2 = error2

        print(name, ':\n', 'MAD (out of sample):', '%.4f' % er1, '; MSE (out of sample):',
'%.4f' % er2)
```

```
        # use the best estimator(respectively based on MAD and MSE) to predict all
        y_preds1 = et1.predict(X)
        y_preds2 = et2.predict(X)
        # calculate E_out
        finaly_preds = et1.predict(finalX_test)
        count = 0
        tol = 0.8
        for i in range(len(finaly_test)):
            if abs(finaly_preds[i]-finaly_test[i]) <= tol:
                count += 1
        print('With tolerance ', '%.4f' % tol, ', E_out is ', '%.4f' %(1 -
count/len(finaly_test)))
        # put corresponding vectors of prediction of all samples and MAD, MSE into a
dictionary with keys as names of the estimators
        ets[name] = (y_preds1, mean_absolute_dev(y_preds1, y), y_preds2,
mean_squared_err(y_preds2, y))
    return ets
# basic linear regression
# Parameters: X: feature matrix, y: quatity vector
# Returns: a dictionary created by make_KfoldCV_regression() method
def basic_linear_regression(X, y):
    # define estimator
    estimator = LinearRegression()
    # create the vector of tuples required as parameter of make_KfoldCV_regression(),
here we have 1 estimator
    estimators = [('linear regression', estimator)]
    return make_KfoldCV_regression(estimators, X, y)
```

```
# Linear regression with L1, L2, ElasticNet regularizer

# Parameters: X: feature matrix, y: quatity vector

# Returns: a dictionary created by make_KfoldCV_regression() method

def regularizer_linear_regression(X, y):

    # define estimators

    estimator1 = Lasso(alpha = 0.2)

    estimator2 = Ridge(alpha = 0.8)

    estimator3 = ElasticNet(alpha = 0.9)

    # create the vector of tuples required as parameter of make_KfoldCV_regression(),
here we have 3 estimators

    estimators = [('L1 Regularizer', estimator1), ('L2 Regularizer', estimator2),
('Elastic Net Regularizer', estimator3)]

    return make_KfoldCV_regression(estimators, X, y)


# Linear regression with RFE method, using L1, L2 and Elastic Net Regularizer

# Parameters: X: feature matrix, y: quatity vector

# Returns: a dictionary created by make_KfoldCV_regression() method

def RFE_linear_regression(X, y, n_features = 6):

    # define estimators

    estimator1 = Lasso(alpha = 0.2)

    estimator2 = Ridge(alpha = 0.8)

    estimator3 = ElasticNet(alpha = 0.9)

    # create the vector of tuples required as parameter of make_KfoldCV_regression(),
here we have 3 estimators

    estimators = [('RFE L1 Regularizer', estimator1), ('RFE L2 Regularizer',
estimator2), ('RFE Elastic Net Regularizer', estimator3)]
```

```
ets = {}

for name, estimator in estimators:

    # select best model with minimum error (et1, er1 for MAD, et2, er2 for MSE)

    et1 = None

    et2 = None

    er1 = None

    er2 = None

    finaly_test = None #used to calculate E_out

    finalX_test = None

    for X_train, y_train, X_test, y_test in get_kfold_train_test(X, y):

        # do feature selection

        selector = RFE(estimator, n_features, step=1)

        # do training

        selector.fit(X_train, y_train)

        # get prediction vector

        preds = selector.predict(X_test)

        # calculate MAD, MSE (out of sample)

        error1 = mean_absolute_dev(preds, y_test)

        error2 = mean_squared_err(preds, y_test)

        # select the best training set and test set, i.e select the best estimator, and the
corresponding MAD and MSE (out of sample)

        if er1 is None:

            et1 = copy.deepcopy(selector)

            er1 = error1

            finaly_test = copy.deepcopy(y_test)
            finalX_test = copy.deepcopy(X_test)
        else:
            if error1 < er1:
                et1 = copy.deepcopy(selector)
                er1 = error1
                finaly_test = copy.deepcopy(y_test)
                finalX_test = copy.deepcopy(X_test)
        if er2 is None:
            et2 = copy.deepcopy(selector)
            er2 = error2
        else:
            if error2 < er2:
                et2 = copy.deepcopy(selector)
                er2 = error2
    print(name, ':\n', 'MAD (out of sample):', '%.4f' % er1, '; MSE (out of sample):',
'%.4f' % er2)
    # use the best estimator(respectively based on MAD and MSE) to predict all
    y_preds1 = et1.predict(X)
    y_preds2 = et2.predict(X)
    # calculate E_out
    finaly_preds = et1.predict(finalX_test)
    count = 0
    tol = 0.8
    for i in range(len(finaly_test)):
        if abs(finaly_preds[i]-finaly_test[i]) <= tol:
                count += 1

        print('With tolerance ', '%.4f' % tol, ', E_out is ', '%.4f' %(1 -
count/len(finaly_test)))

        # put corresponding vectors of prediction of all samples and MAD, MSE into a
dictionary with keys as names of the estimators

        # put corresponding vectors of prediction of all samples and MAD, MSE into a
dictionary with keys as names of the estimators

        ets[name] = (y_preds1, mean_absolute_dev(y_preds1, y), y_preds2,
mean_squared_err(y_preds2, y))

    return ets



# Nonlinear regression with Logistic L1, L2 regulatizer and BayesianRidge
regularizer

# Parameters: X: feature matrix, y: quatity vector

# Returns: a dictionary created by make_KfoldCV_regression() method

def regularizer_nonlinear_regression(X, y):

    # define estimators

    estimator1 = LogisticRegression(penalty='l1', C = 0.8)

    estimator2 = LogisticRegression(penalty='l2', C = 0.8)

    estimator3 = BayesianRidge(n_iter=300)

    # create the vector of tuples required as parameter of make_KfoldCV_regression(),
here we have 3 estimators

    estimators = [('Logistic L1 Regularizer', estimator1), ('Logistic L2 Regularizer',
estimator2), ('Bayesian Ridge Regularizer', estimator3)]

    return make_KfoldCV_regression(estimators, X, y)



# SVM regression

# Parameters: X: feature matrix, y: quatity vector
# Returns: a dictionary created by make_KfoldCV_regression() method

def SVM_regression(X, y):

    # define estimators

    estimator1 = SVR(kernel='linear')

    estimator2 = SVR(kernel='rbf')

    # create the vector of tuples required as parameter of make_KfoldCV_regression(),
here we have 2 estimators

    estimators = [('SVM with linear kernel', estimator1), ('SVM with RBF kernel',
estimator2)]

    return make_KfoldCV_regression(estimators, X, y)



# launch a selected regression and show MAD, confusion matrix and plot

def lauch_regression(regression_handler, X, y, n_features = None):

    # call regression

    if n_features is None:

        dic = regression_handler(X, y)

    else:

        dic = regression_handler(X, y, n_features)

    # handle results

    actual_list = []

    preds_list = []

    name_list = []

    for key in dic:

        # print confusion matrix

        print('confusion matrix for %s:' % key)

        print_confusion_matrix(y, dic[key][0])
        # create parameters for plot_result()

        actual_list.append(y)

        preds_list.append(dic[key][0])

        name_list.append(key + ', MAD (in sample and out of sample error): ' + 
str(numpy.round(dic[key][1],4)) + '\n MSE (in sample and out of sample error): ' + 
str(numpy.round(dic[key][3],4)))

    # plot result

    plot_result(actual_list, preds_list, name_list)



# run the program

# red wine

input('Press any key to load red wine data...')

X, y = read_data('winequality-red.csv')

print('%d rows, %d features loaded!' % (len(X), len(X[0])) )

print('================================================')

input('Press any key to start linear regression...')

lauch_regression(basic_linear_regression, X, y)

print('================================================')

input('Press any key to start regularizer linear regression...')

lauch_regression(regularizer_linear_regression, X, y)

print('================================================')



input('Press any key to select best K features by feature selection...')

X_new, indices = select_KBest(X, y)

print(X_new.shape, indices)

print('================================================')
```

```
input('Press any key to start linear regression with above K features...')

lauch_regression(basic_linear_regression, X_new, y)

print('================================================')

input('Press any key to start regularizer linear regression with above K features...')

lauch_regression(regularizer_linear_regression, X_new, y)


print('================================================')

input('Press any key to start feature selection by RFE...')

lauch_regression(RFE_linear_regression, X, y, n_features=6)

print('================================================')


input('Press any key to start regularizer nonlinear regression...')

lauch_regression(regularizer_nonlinear_regression, X, y)

print('================================================')

input('Press any key to start SVM regression...')

lauch_regression(SVM_regression, X, y)


# white wine

input('Red winecx Finished!\nPress any key to load white wine data...')

X, y = read_data('winequality-white.csv')

print('%d rows, %d features loaded!' % (len(X), len(X[0])) )

print('================================================')

input('Press any key to start linear regression...')

lauch_regression(basic_linear_regression, X, y)

print('================================================')
```

```
input('Press any key to start regularizer linear regression...')
lauch_regression(regularizer_linear_regression, X, y)
print('==================================================')


input('Press any key to select best K features by feature selection...')
X_new, indices = select_KBest(X, y)
print(X_new.shape, indices)
print('==================================================')
input('Press any key to start linear regression with above K features...')
lauch_regression(basic_linear_regression, X_new, y)
print('==================================================')
input('Press any key to start regularizer linear regression with above K features...')
lauch_regression(regularizer_linear_regression, X_new, y)


print('==================================================')
input('Press any key to start feature selection by RFE...')
lauch_regression(RFE_linear_regression, X, y, n_features=6)
print('==================================================')


input('Press any key to start regularizer nonlinear regression...')
lauch_regression(regularizer_nonlinear_regression, X, y)
print('==================================================')
input('Press any key to start SVM regression...')
lauch_regression(SVM_regression, X, y)
```