# Project-I by Group Macau

**Chen Liu**
EPFL
chen.liu@epfl.ch

**Mengjie Zhao**
EPFL
mengjie.zhao@epfl.ch

## Abstract

This report demonstrates the project-I working results of group Macau in the 2015 Pattern Classification and Machine Learning (PCML) course. The report mainly consists of two parts, namely the regression section and the classification section. Within both sections, we examined the data's features and constructed different models. After estimating the testing performance of each model, we employed the best model to generate predictive outputs of testing instance.

## 1 Regression

Regression helps in finding the relationships between input and output of data sets. Hence, it is applied to predict output values of new input values. After studying given data sets of regression, we proposed 10 different learning algorithms with increasing level of complexity to model the input-output relationships. The predictive outputs corresponding to the given testing data set are included in the 'predictions_regression.csv' file.

### 1.1 Regression data description

In the regression section, our training data consists of input variable $\mathbf{X\_train}$ and output variable $\mathbf{y\_train}$. $N = 2800$ data examples are provided, and each input vector $\mathbf{x}_n$ has $D = 70$ dimensions (features). Out of these 70 features, 58 features are real valued and 12 features are categorical, within which 4 features have binary values, 4 features are of 3 categories and the remaining 4 features are of 4 categories.

Our testing data set $\mathbf{X\_test}$ contains $N = 1200$ data examples. However, we only applied it in our final model to generate predictive outputs. In addition, we proposed an approximation of the test error of this testing input. We employed root-mean-square error (RMSE) to evaluate our models.

### 1.2 Regression data visualization and cleaning

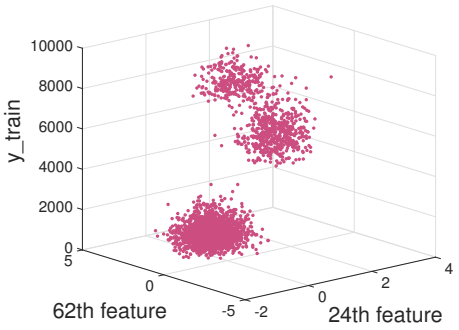
Performing exploratory data analysis is a crucial step before implementing future analysis. We firstly checked the distribution of the training data set $\mathbf{X\_train}$ and found that it is not centered. As a result, data normalization was performed at first. In addition, we identified some outliers and took some actions to reduce the impacts of them, which will be discussed in following contents.

As $\mathbf{X\_train}$ has 70 features, we investigated the correlations between $\mathbf{y\_train}$ and each feature of $\mathbf{X\_train}$ (see Table 1). The results show that not every feature is strongly correlated with $\mathbf{y\_train}$. There are two features that are overwhelmingly correlated to the output compared with others. Hence, we believe that it is feasible to ignore some features that have really small correlations with the expected output. The simplified model will run faster but still preserve good performance.

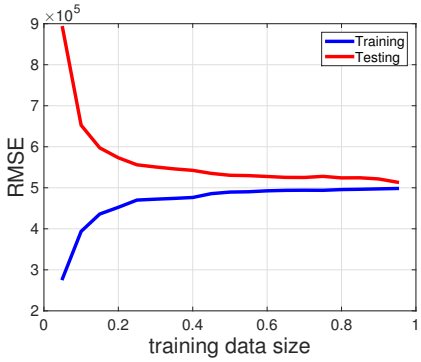
Due to the page limit, Figure.1(a) only illustrates the relationships among $\mathbf{y\_train}$, the $24th$ feature and the $62nd$ feature, the features that most linearly correlated to the outputs, of input $\mathbf{X\_train}$. It can be clearly observed from Figure.1(a) that $\mathbf{y\_train}$ can be approximately separated into 2 or 3

| FeatureIndex | 24 | 62 | 57 | 39 | 42 | 52 | 49 | 35 |
|---|---|---|---|---|---|---|---|---|
| Correlation | 0.8752 | 0.5121 | 0.0763 | 0.0744 | 0.0728 | 0.0622 | 0.0570 | 0.0557 |

Table 1: Eight features of input that are most linearly correlated to output




(a) $\mathbf{y\_train}$ against the $24th$ and $62nd$ feature of $\mathbf{X\_train}$. From the figure, $\mathbf{y\_trains}$ can be approximately separated into 2 or 3 parts when $\mathbf{x_N^{24}}$ and $\mathbf{x_N^{62}}$ having different values.




(b) Learning curves. When training data set takes up more than $80\%$ of total data, training error remains stable and the testing error becomes close to training error

Figure 1: The correlation among outputs, $\mathbf{x_N^{24}}$ and $\mathbf{x_N^{62}}$, as well as the learning curves.

data parts by the $24th$ and $62nd$ feature of $\mathbf{X\_train}$. Under this circumstance, one of our proposed learning algorithms will utilize this trait, firstly dividing the data into 2 or 3 parts then carrying out regression on each part respectively. Detailed illustrations are provided in following sections.

Considering the 12 categorical features within $\mathbf{X\_train}$, it should be indicated that we did not applied advanced methods such as dummy coding to deal with the categorical inputs. The reason is that we did not experience significant performance alterations after we took actions (deletion) on these categorical features. To avoid losing information and to make our model simpler, we decided to treat these categorical features as the same to other features of $\mathbf{X\_train}$.

By applying the rank() command in MATLAB, we found that the rank of matrix $\mathbf{X\_train}$ is 61, which is smaller than 70, implying that $\mathbf{X\_train}$ is rank-deficient. Hence, in some of our learning algorithms (ridge regression etc.), we improved the eigenvalues of $\mathbf{X\_train}$ to try to reduce the impacts of ill-conditioning.

Since we have no access to the expected output $\mathbf{y\_test}$ of $\mathbf{X\_test}$, we need to split the given $\mathbf{X\_train}$ into training and validation data to estimate test errors and prevent overfitting.

To decide how to split $\mathbf{X\_train}$, we obtained one learning curve, as displayed in Figure.1(b), showing how various proportions of training data sets impact on the RMSE performance of the constructed model. From this figure, it can be easily observed that when more than $80\%$ of $\mathbf{X\_train}$ is split as training data, the RMSE performances are stable. According to this result, we decided to use $5-fold$ cross validation in examining different learning models, which means $80\%$ data was used for training and the remaining $20\%$ data for validation. It should be indicated that the stability in Figure.1(b) is obtained by applying linear regression, however, for consistency we applied $5-fold$ cross validation among all of our learning models.

### 1.3 Regression models

Within this section, several learning models developed by us will be demonstrated one by one. These models have different level of complexity, therefore, the RMSE performances vary significantly. Following Figure.2 gives a pictorial performance comparison between different learning models.

The $x$ axis in Figure.2 represents index of different learning models while the $y$ axis stands for the RMSE performance. The first model is naive least square regression, which can capture the linear relationship between the input $\mathbf{X\_train}$ and output $\mathbf{y\_train}$.

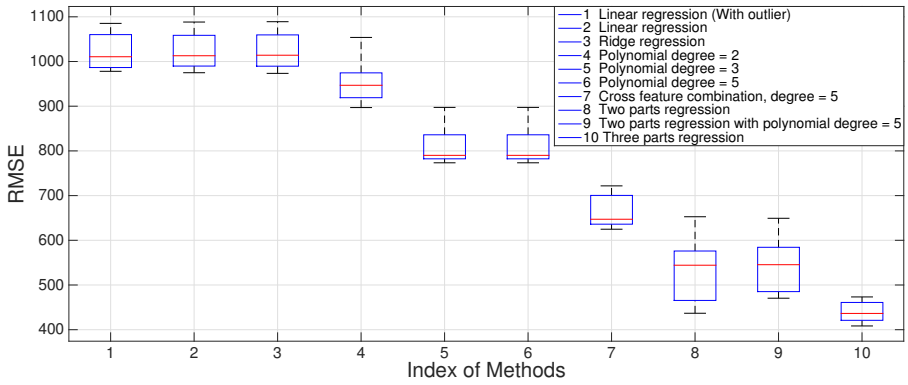
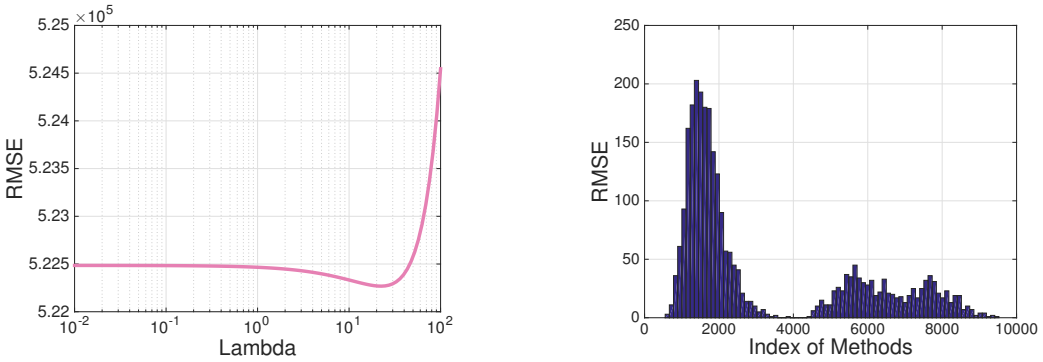



Figure 2: Performance comparison between models with different complexity




(a) Ridge regression learning curve.

(b) Distribution of $\mathbf{y\_train}$

Figure 3: Ridge regression and distribution of $\mathbf{y\_train}$ (utilized in splitting model).

It should be noticed that this model are applied on data with outliers. Starting from model 2, data without outliers will be used. To detect outliers, we took advantage of special properties of outliers to detect them. When conducting regression, outliers are usually far away from our predictive value, contributing a lot to overall loss. Based on this idea, we first conducted linear regression and calculate error-loss of each point. Points whose error-loss is more than 4 times the median value are regarded as outliers. By removing such outliers (usually account for $1\%$ to $2\%$ of all points), we noticed a slight improvement of our model's performance.

As mentioned in above sections, the rank of $\mathbf{X\_train}$ is smaller than the total number of features. We applied ridge regression to increase the eigenvalues of $\mathbf{X\_train}$ to reduce the impacts of ill-conditioning. We saw a slight performance growth when implementing ridge regression. Figure.3(a) displays the testing error performance in ridge regression with different weights $\lambda$ of penalized term.

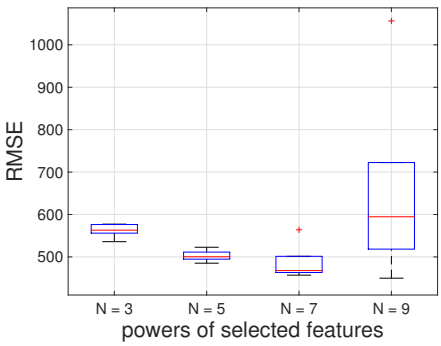
We tried several feature transformations and firstly introduced polynomial features. For power $N$, we augment each feature $\{x_n\}$ to $\{x_n, x_n^2, x_n^3, ..., x_n^N\}$. These models are able to capture polynomial input-output relationship. Model 4, 5 and 6 in Figure.2 implies those models demonstrating better performance firstly with the increase of $N$. However, it is reasonable to infer that when $N$ is assigned with bigger numbers, the models will because ill-conditioning and has a large variance. This is partly because the number of augmented features are too large and the training data is relatively limited, models are underfitting in this scenario.

From model 5 in Figure.2, we used the feature-selection method. During the regression process, instead of applying all features of $\mathbf{X\_train}$, we selected the features that have high correlations with the output, as shown in Table.1. It can be observed that the feature-selection model surpasses the original model, since it lowers the median and variance of RMSE.

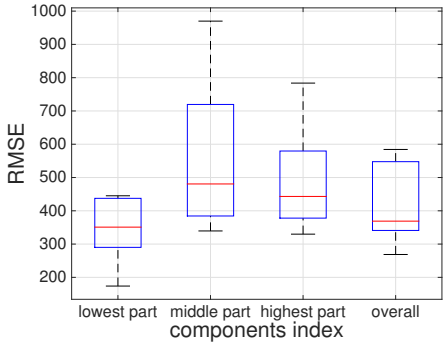
In model 8, 9 and 10, we divided the whole data set into 2 or 3 parts and constructed one learning model for each part receptively. As introduced in section 1.2, we firstly split the data examples according to the value of the $24th$ and $62nd$ feature, then carried out linear regression on each part to calculate the overall error. By applying this method, we received an obvious performance improvement. However, when we try to apply polynomial feature technique to this model, we were encountered the underfitting scenario mentioned above.

Actually, the dividing policy itself introduces much variance, because one point will contribute much to the loss if it is mistakenly classified. This situation will get worse when we divide the whole space into 3 parts, for 2 parts of them are overlapped. This effect can also be observed from the histogram of $\mathbf{y\_train}$, as shown in Figure.3(b). The rightmost parts of data cannot be easily divided. Figure.4(b) shows the partial RMSE loss and variance of each cluster. We can see that in the non-overlapped bottom cluster the RMSE mean and variance is lower, while the upper two overlapped parts' are higher. It is more probably that points are mistakenly classified between these two parts.

In addition, we explored the error performances of feature-selecting models with different power order $N$, as shown in Figure.4(a). Along with the increasing of degrees of selected features, the RMSE errors firstly gradually decreased then increased. On the contrary, the variance keep increasing with the growth of $N$. As a result, we chose $N = 5$ to achieve a relatively good RMSE performance while still remain low variances.




(a) RMSE performance of different values of N in polynomial feature augmentation.




(b) RMSE mean and variance of three parts and the whole data set when we split the data set into three parts and conduct regression separately.

Figure 4: Polynomial model and multi-parts regression.

In addition, we tried build more complex features by introducing not only polynomial features but also 'cross' features. For each $N$-feature point $\{x_1, x_2, ...x_N\}$, we augmented it to $\{\cup_{n=1,1\leq i_1\leq i_2\leq ...\leq i_n\leq N}^{N} \Pi_{j=1}^{n} x_{i_j}\}$. As mentioned above, we only selected features that are most linearly correlated to the output as primitive features. Moreover, we can put more restraint among $i_j$(such as allowing at most one $i_j$ equals to a value other than 1 and $n$) to avoid 'feature explosion'. For fine-tuned features, the model performed better than the previous model of the same power $N$ (Model 7).

According to the performance of different models and their variances, we finally chose the last model (three parts regression) to predict the output of $\mathbf{X\_test}$.

## 2 Classification

Given input data, classification is introduced to categorize outputs into different discrete classes. Within the classification section, several classifiers were constructed based on training data. Different classifiers have various levels of complexity, resulting in different levels of accuracy. We chose the best classifier to predict outputs corresponding to the given testing data set, and the results are included in the 'predictions_classification.csv' file.

### 2.1 Classification data description

In the classification tasks, $N = 1500$ training data examples are given, and each input vector $\mathbf{x}_n$ has $D = 29$ dimensions (features). Among these 29 features, 24 features have continuous values and 5 features are categorical, within which 2 features have binary values, 2 features are of 5 categories and the remaining 1 feature is of 4 categories. Our training output $\mathbf{y\_train}$ has binary values belonging to $\{-1, 1\}$.

The testing data set $\mathbf{X\_test}$ contains $N = 1500$ examples. Similar to regression section, we only apply it in our final model to generate predictive outputs. We will proposed an approximation of the test error corresponding to this testing input.

We employed three methods to evaluate the performance of the models, they are 0-1 loss, negative-loglikelihood and root-mean-square error (RMSE). Negative-loglikelihood is applicable to probability models (logistic regression etc.), while other two methods are also applicable to non-probability models (KNN etc.)

### 2.2 Classification data visualization and cleaning

We firstly looked into the five discrete features. We checked the value of each feature and their corresponding labels. We found that for each value of each label, the point has almost equal probability of belonging to either point. That is to say, these discrete features have little impact on the output labels. In some of the following proposed models, we simply discard these features.

We paid more attention to the distributions of continuous features. We found that in our data set, for same features, the values of some instances is very big while others are very small. This causes a unsmooth and non-Gaussian distribution, which might affect the effectiveness of logistic regression. Instead of polynomial feature, we introduced a $nth - root$ feature to make the input features more smooth and Gaussian, which will be described in detail in the following sections.

### 2.3 Classification Models

In this section, we will introduce our models one by one. For convenience, we employed 0-1 loss as main evaluation. Similar to regression task, We use 5-fold cross-validation to estimate testing loss.

Figure.5 shows the performance of each model. Our baselines are naive logistic regression and k-nearest neighbor (KNN). For naive logistic regression, we obtain a loss of 0.31. For KNN, Figure.6(b) shows its performance with different values of K. Very small and very large K leads the model to be too local and too global respectively. When $K \geq 11$, the performance becomes stable. We took the scenario $K = 12$ as representative of KNN algorithm, whose 0-1 loss is $0.35$ and worse than naive logistic regression.

Penalized logistic regression penalize too complex models to avoid overfitting. However, according to 0-1 loss scale, it performs worse than naive logistic regression. We noticed that if we set the penalized parameter $\lambda$ properly, the penalized model would achieve better negative loglikelihood loss.

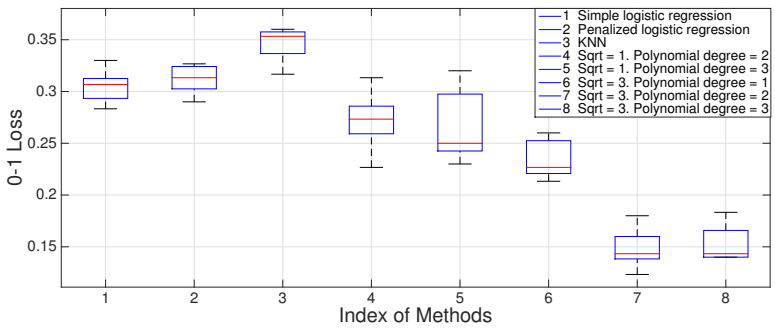
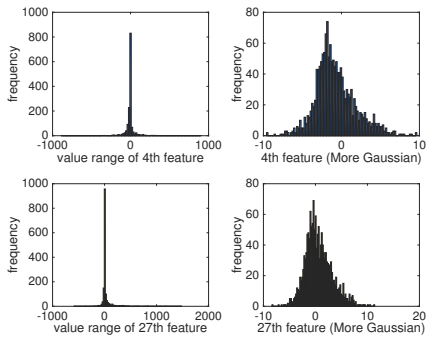
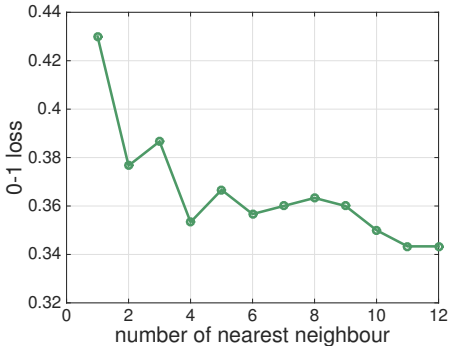



Figure 5: Performance comparison between classifiers with different complexity




(a) Left are value distribution of 4th and 27th feature of the original training data. Right are the corresponding distribution after 2nd-root transformation, they are more Gaussian.




(b) KNN plot, showing the alteration of $0-1$ loss along with the increase of K.

Figure 6: n-th feature transformation and the performance of KNN algorithm.

Similar to regression models, we introduced $N - polynomial$ to augment features. Model $4$ and $5$ shows its performance when $N = 2$ and $N = 3$ respectively. When $N \geq 4$, the model becomes ill-conditioned and has a large variance because of underfitting.

As mentioned in section 2.2, values of continuous features are either too big or too small. To make the input feature more Gaussian, we augmented input features by adding $n-th$ (n = 2,3,...,N) root of each original features, as shown in Figure.6(a). What needs to be pointed out is that all of these are done before data normalization. To avoid illegal operations, when $n$ is even, we defined the $n-th$ root of x as $-\sqrt[n]{-x}$ when x is negative. Figure.5 shows that this model performs well when using different N values. Like polynomial, the model becomes ill-conditioned with large $N$.

When we combined the n-th root trick with polynomial feature augmentation, the model achieved the best performance in cross-validation. Our final model (model 7) used 3-rd root and 2-order polynomial features to generate class labels for $\mathbf{X\_test}$.

## 3 Summary

In this project, we applied regression and classification methods on real-world data. We found it a challenging job especially when the data is noisy and has much redundant information. Under this circumstance, even robust, fine-tuned models only demonstrate average performance.

Data analysis is very important before designing and building up our models. For regression task, we extracted two most important features, obtained three relatively distinct clusters and sharpened the models' performance significantly. For classification task, we noticed the distribution of continuous features and applied suitable feature augmentation methods to boost the effectiveness of logistic regression.

For both tasks, we used cross-validation to avoid over-fitting. By using random variable and running models for several times, we evaluated models' variance and chose the model with better performance and lower variance.

**Acknowledgments**

We would like to thank Emti and all TAs, for offering their help and suggestions during this project. The codes of this project are implemented by group members independently. The report are written by Chen Liu and Mengjie Zhao, who are of the equal contribution. All codes are available on github (link). 