Machine Learning 2013: Project 1 - Regression Report

rabeehk@student.ethz.ch
robing@student.ethz.ch
aseifodd@student.ethz.ch

November 1, 2013

Experimental Protocol

For reproducing our results, one should use the code.m file beside this report and put the *training.csv* and *testing.csv* file next to it. By running the code.m file in MATLAB, if you have all the default MATLAB features, the program will learn a linear regression based on data provided in *training.csv* file and produce a **ypred.txt** file next to itself which contains all the predications for *testing.csv* input. You can submit this file to the judge system and see the the score.

1 Tools

We only used MATLAB for this project. Besides general features and functions of MATLAB we utilized *ridge* function from Curve-Fitting toolbox. The source code is uploaded in the submission system.

2 Algorithm

We used the ridge regression algorithm for regressing the data. Whilst we started out with a completely self-implemented version of ridge regression, for simplicity and efficiency reasons we moved to the built-in **ridge** MATLAB function. Then, we estimated a range for hyperparameter(λ) parameter of ridge regression and exploited cross validation on each possible parameter value. At the end, we use the parameter with least average error rate for determining weights of predicts.

3 Features

Most of our effort in the project was focused in this part. We realized that relationship of some features is completely non-linear with respect to y. So, we tried some transformations including $sqrt, log2, log, exp, 1/x$ on those features and also on y. It turned out that $sqrt(y)$ is a good transformation for the output. So, we use *square root* for the training and *power* for transforming the y predications to real data. Besides transforming y, we added the $log2(data)$ and $sqrt(data)$ of all feature to our feature set, mainly because most of the features are powers of 2 or step-wise increments. So, we

hope that getting $log2$ or $sqrt$ straighten their curvature. For determining which transform is better for a specific original feature or y, we used *corr* function in MATLAB to get an rough idea on how linearly correlated the new feature is with respect to y. Then we choose the transformations with high correlation. In our case, they were log and $sqrt$. We also tried to consider interaction between features, so we decided to used *x2fx* function of MATLAB to produce every possible multiplication of original features and also their power 2 and add them to our feature set. Of course we eliminated the constant vector added to our features by this *x2fx* function. After integrating all these feature, we will have a very large feature set which we should select best features out of it. At this stage, we calculate correlation between y and each feature. This is done by *corr* function. Then, we drop features that their p-values are higher than 0.05. This threshold was mentioned in the MATLAB documentation and also seem to work well on our data. After basic filtering of features, we use a MATLAB function named *stepwisefit* to drop out every feature that does not contribute enough to a linear regression towards y. We provide this function with our feature set, the y values and a *inmodel* parameter which indicates which feature is in the initial feature set. We initialize this function with all features and the algorithm will drop out non-correlated features in each step until converge to a final set. There are two parameters called *penter* and *premve* which control how features are dropped and therefore, affect the size of result features set. We used default values for these parameters, as they produce the best result. This algorithm interestingly might add features that were removed in some previous steps and this feature is useful in our case, because decrease the chance of trapping in local optimum. At the end, our feature space is reduced drastically by *stepwisefit* function and the most relevant features for linear regression are selected. But we observed that 13th column of original data has a strong correlation to y. So, we decided to add this feature and it's *log* and *sqrt* to the final set to improve it's weight on regression. Then, we use this feature set for learning the ridge regression parameters.

4 Parameters

We have one parameter in our learning algorithm and it is the hyperparameter(λ) of ridge regression. We used cross validation over the range of $[0, 20e-3]$ with step sizes as $1e-5$ on the training set to find the best value. Of course, this hyperparameter range is chosen empirically. Our cross validation is 10 fold, but the results with other reasonable fold numbers like 5-12 were not much different.

5 Lessons Learned

We tried several transformation on the data as well as y, but most of them didn't improve the result. For example, some transformations like $ex(data), 1/data, exp(y), 1/y, y.^2, data.^-2$ had bad effects on regression result. We interpret this as the curvature between that specific data and y was on some shape that this transformation just improved it's curvature instead of straightening non-linearity. We learned that **sequentialfs** which is general purpose feature selection algorithm, does not work well in our case. We guess this is because of the specific dropping criteria which it use for eliminating features. Furthermore, it cannot add features which were removed earlier. Therefore, it is more prone to trapping in local optimum and losing some precious features. Furthermore, it is really slow at runtime. So, we can safely say that for linear regression feature selection if our criteria is least square error, the *stepwisefit* function is a better choice than *sequentialfs*. Finally we learned that systematic approach for creating new features is more successful than just randomly adding new features from different transformations.