Imperial College London

Machine Learning Coursework 3

Youssef Rizk (00940962)

Examined by
Dr. András György

March 10, 2017

Contents

1 Problem 1

In this problem, we seek to determine an upper bound on the test error of an ε-optimal hypothesis h_ε. We are given that an ε-optimal hypothesis obeys the following relationship: $\hat{R_n}(h_\varepsilon) \leq \varepsilon + \min_{h\in\mathcal{H}} \hat{R_n}(h)$. From the Hoeffding inequality, where M represents the size of the hypothesis class (i.e. how many hypothesis are in the hypothesis class), we know that with probability $1-\delta_1$

$$|\hat{R_n}(h) - R(h)| \leq \sqrt{\frac{\log \frac{2M}{\delta_1}}{2n}}$$

and so, we can take

$$R(h) - \hat{R_n}(h) \leq \sqrt{\frac{\log \frac{2M}{\delta_1}}{2n}}$$

If we now substitute h_ε for h and rearrange, we obtain an upper bound on the test error of h_ε.

$$\begin{aligned}
R(h_\varepsilon) - \hat{R_n}(h_\varepsilon) &\leq \sqrt{\frac{\log \frac{2M}{\delta_1}}{2n}} \\
R(h_\varepsilon) &\leq \sqrt{\frac{\log \frac{2M}{\delta_1}}{2n}} + \hat{R_n}(h_\varepsilon) \\
R(h_\varepsilon) &\leq \sqrt{\frac{\log \frac{2M}{\delta_1}}{2n}} + \varepsilon + \min_{h\in\mathcal{H}} \hat{R_n}(h)
\end{aligned}$$

Now, we would also like give an upper bound on $R(h_\varepsilon) - \min_{h\in\mathcal{H}} R(h)$. We can derive this from the previous result as follows,

$$R(h_\varepsilon) - \min_{h\in\mathcal{H}} R(h) \leq \sqrt{\frac{\log \frac{2M}{\delta_1}}{2n}} + \varepsilon + \min_{h\in\mathcal{H}} \hat{R_n}(h) - \min_{h\in\mathcal{H}} R(h)$$

Using $\min_{h\in\mathcal{H}} \hat{R_n}(h) - \min_{h\in\mathcal{H}} R(h) \leq \sqrt{\frac{2\log\frac{2M}{\delta_2}}{n}}$ w.p. $1-\delta_2$, then w.p. $1-\max(\delta_1,\delta_2)$

$$R(h_\varepsilon) - \min_{h\in\mathcal{H}} R(h) \leq \sqrt{\frac{\log \frac{2M}{\delta_1}}{2n}} + \varepsilon + \sqrt{\frac{2\log \frac{2M}{\delta_2}}{n}}$$

2 Problem 2

In this problem, the task is to determine a value y such that the average absolute error, $\frac{1}{n}\sum_{i=1}^{n}|y-y_i|$, is minimized. We take the points arranged such that $y_1 \leq y_2 \leq ... \leq y_n$.

First, consider two points, x_1 & x_2, such that $x_2 \geq x_1$, then for any $\alpha \in [x_1, x_2]$, the sum of the absolute deviations of the two points is $|\alpha - x_1| + |\alpha - x_2| = \alpha - x_1 + x_2 - \alpha = x_2 - x_1$. Consider the case that $\alpha \notin [x_1, x_2]$. If $\alpha < x_1$, then the sum of the absolute deviations is $|\alpha - x_1| + |\alpha - x_2| = x_1 - \alpha + x_2 - \alpha = x_2 - x_1 - 2\alpha > x_2 + x_1 - 2x_1 = x_2 - x_1$. On the other hand, if $\alpha > x_2$, then the sum of the absolute deviations is $|\alpha - x_1| + |\alpha - x_2| = \alpha - x_1 + \alpha - x_2 = 2\alpha - x_2 - x_1 > 2x_2 - x_2 - x_1 = x_2 - x_1$. Thus, we can now observe that the absolute error of the two points from another point α is minimized when $\alpha \in [x_1, x_2]$ and is equal to $x_2 - x_1$.

Now, consider the case that n is odd, the expression for average absolute error can be expanded and rewritten as

$$\frac{1}{n}((|y_1 - y| + |y_n - y|) + (|y_2 - y| + |y_{n-1} - y|) + ... + (|y_{\frac{n-1}{2}} - y| + |y_{\frac{n+3}{2}} - y|) + |y_{\frac{n+1}{2}} - y|)$$

This look similar to the results in the previous paragraph, where x_1 & x_2 take on values of the smallest and largest data points, respectively, and then the second smallest and second largest, and so on. Thus, we essentially get a set of intervals: $[y_1, y_n]$, $[y_2, y_{n-1}]$,...,$[y_{\frac{n+1}{2}}, y_{\frac{n+1}{2}}]$, where the last interval is a single point equal to the median.

Notice that each interval is an element of the preceding interval. Now, we would like to minimize the absolute error between the endpoints of the intervals and some y (α in the previous paragraph) using the same y for all intervals. Since we know that y must be confined between the endpoints, we can define $y \in \bigcap_{i=1}^{\frac{n+1}{2}} [y_i, y_{n+1-i}]$. Since this intersection of intervals is equal to the smallest or "innermost" interval, $[y_{\frac{n+1}{2}}, y_{\frac{n+1}{2}}]$, a single point, y will be chosen equal to $y_{\frac{n+1}{2}}$, which is the median of the data set. Thus, the median here minimizes the average absolute error.

Now, consider the case where n is even. We repeat everything as with the the odd case, however, we note that the "innermost" interval now is $[y_{\frac{n}{2}}, y_{\frac{n}{2}+1}]$. Thus, if we again choose $y \in \bigcap_{i=1}^{\frac{n+1}{2}} [y_i, y_{n+1-i}]$, we end up choosing any $y \in [y_{\frac{n}{2}}, y_{\frac{n}{2}+1}]$. Again, this is a median of the data set.

Thus, it can be seen that the median of the data set (whether unique or not) minimizes the average absolute error.

3 Problem 3

In this problem, we examine the a movie recommendation problem, where we are given a training data set that contains 671 users who have rated some of 9066 movies. The aim is to establish a predictor that minimizes the average mean-squared error. The following sections will explore different ways of achieving this.

3.1 Part A: Constant Base Predictor

The simplest predictor we could employ is a constant predictor for any movie, irrespective of the user, or any user, irrespective of the movie. To provide a constant predictor for movies, we could simply average the ratings that each movie receives. Similarly, we could average the ratings that each user gives in order to make a constant predictor for users. This is done in the MATLAB program. For a constant predictor for users, we obtain a training and test error of 0.9108 & 0.9269, respectively. For a constant predictor for movies, we obtain training and test errors of 0.7851 & 1.4410, respectively. The larger discrepancy between the errors in the latter hints at overfitting. In general, however, because of the lower test error, we would prefer to use a constant predictor for users rather than movies.

3.2 Part B: Linear Regression Baseline

Naturally, we do not expect a constant base predictor to be very successful as a predictor. We can do better using linear regression. However, we employ regularization to improve the performance of the predictor, by penalizing different hypothesis classes differently. This leads to a variation of linear regression known as Ridge Regression, where the optimal weights are changed from the linear regression form to the ridge regression form shown below.

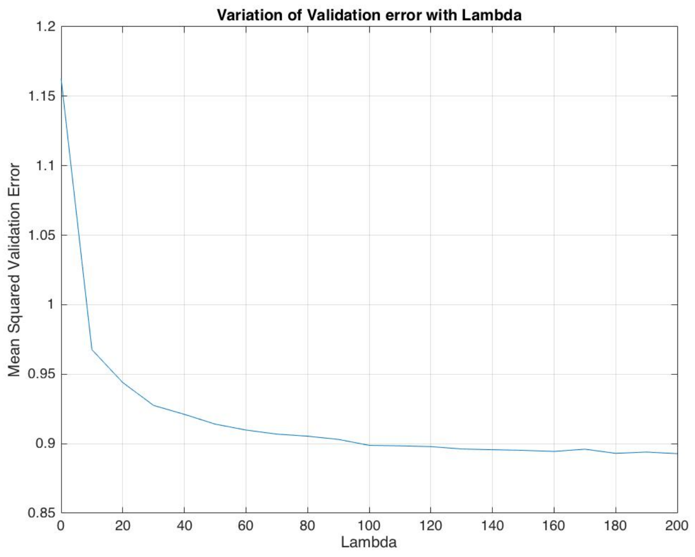
Linear Regression	Ridge Regression
$\mathbf{w}_{\text{lin}} = (\mathbf{Z}^\top \mathbf{Z})^{-1} \mathbf{Z}^\top \mathbf{y}$	$\mathbf{w}_{\text{reg}} = (\mathbf{Z}^\top \mathbf{Z} + \lambda \mathbf{I})^{-1} \mathbf{Z}^\top \mathbf{y}$

where $\mathbf{Z}$ is the training set, where each data point is a row $\in \mathbb{R}^d$ where d is the number of features, and $\mathbf{y}$ contains the ratings of each point.

In order to choose a suitable λ, I begin the program with a cross validation sequence. Here, I partition my training set into 10 disjoint sets. I train my predictor on 9 of the 10 subsets, and I use the remaining set as my validation set, where I test the performance of the generated predictor. I then repeat this, changing the subset that is the validation set, until they have all served as validation sets. I average the 10 generated validation errors to obtain a performance measure of this specific hypothesis class. I repeat this procedure for various λ's, and I choose the one that provides the minimal average validation error.

Interestingly, it seems that the validation performance of the ridge regression predictor improves with increasing λ. I apply cross validation for $0 \leq \lambda \leq 200$, in increments of 1. On average, the best-performing λ is around 200, as plotted below. As expected, the validation error becomes worse as λ increases beyond this order of magnitude.

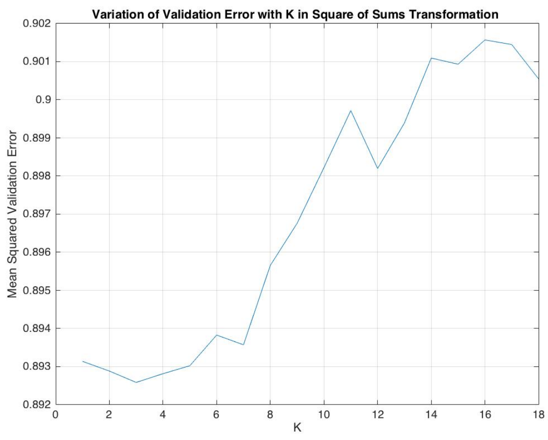
Once the cross validation part is over, I use the generated value of λ and train the predictor on the entire training set and test the performance on the test set. I obtain a test error of 0.8875, a significant improvement over the constant base predictor.

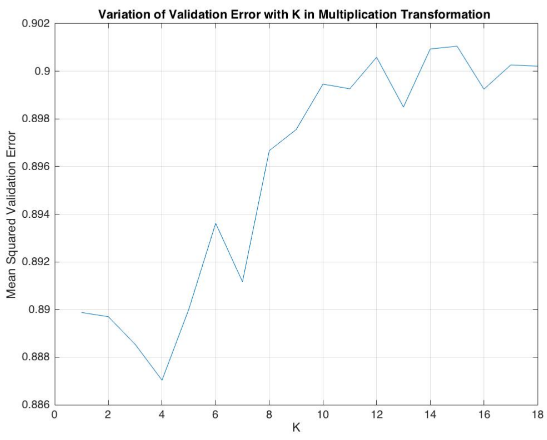


3.3 Part C: Linear Regression with Transformed Features

Building upon the results of the linear regression predictor, it may be possible to improve the results even further by transforming the movie features in a non-linear fashion. The fundamental idea behind my transformations is to group similar features (i.e. genres in this case) and perform a non-linear operation to these and append them to the original feature set. Two transformations that I considered attempted to 1) multiply similar features 2) square the sum of the similar features. In this exercise, I judge similarity by using MATLAB's built-in function `kmeans` which groups the features based on their correlation.

Essentially, I apply the `kmeans` function on the features, which groups similar features into k groups, where k is a parameter of my choosing. In order to choose an optimal k, I cross-validate. Below are two figures showing the variation of validation error with k for each transformation. We see that there is a trend that the validation error increases with increasing k, although there is some randomness. Once a suitable value of k is found, I apply cross validation to the transformed features to find a new optimal λ as was done in the previous section. Importantly, the cross validation for k includes cross validation for λ, i.e. for each value of k, I cross validate to obtain the best λ for that k. The best λ in that case is on average around 160.





Where the non-linear transformation is grouping the features and multiplying the features in the same group, the optimal value k is on average chosen as 4 and we obtain a test error of 0.8826, which is an improvement over the untransformed data set. Where the transformation is squaring the sum of the features in the same group, the optimal value of k chosen with cross validation is on average 4 and we obtain a test error of 0.8865. This is a minor improvement over the untransformed data set. We can, thus, see that the transformed feature set provides an improvement in the test errors, albeit not a significant one.

3.4 Part D: Collaborative Filtering

Attempting to further improve the performance of our predictor, we finally try the collaborative filtering method, whereby we employ stochastic gradient descent (SGD) to learn simultaneously the weights of the users and the movies. In essence, this method does not employ the existing features, but rather establishes new ones.

We try to come up with predictions of ratings by user i for movie j of the form $u_i^\top v_j$ where $u_i, v_j \in \mathbb{R}^K$ for some $K \geq 1$, and where u_i is the user weights for user i and v_j is the movie weights for movie j. Naturally, the uncertainty about the value of K calls for some cross-validation, however, this will be discussed later.

In this exercise, I use SGD, a variant of normal gradient descent. In normal gradient descent, we update the weights according to the formula

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \eta \nabla \hat{R}_n(\mathbf{w}_t)$$

where η is the learning rate. In SGD, we randomly choose a training point, based on which to update the user and movie weights. We must define a loss function which are trying to minimize, and in our case, it can be defined as

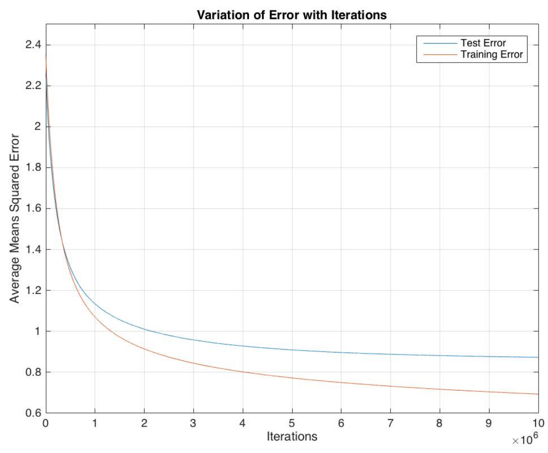
$$\ell_{ij} = \Big(r_{ij} - \sum_{k=1}^{K} u_{i,k} v_{j,k}\Big)^2$$

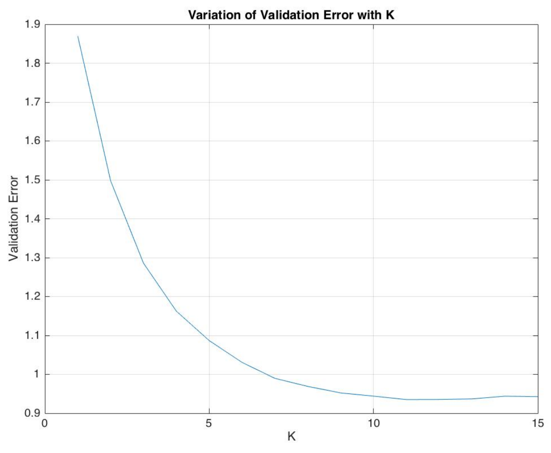
where r_{ij} is the rating user i gives to movie j. Specifically, if ${u_i}^t$ is the user weight of user i at the t^{th} iteration, and r_{ij} is the rating user i gives to movie j, then we update the weights based on

$$\begin{aligned} {u_{i,k}}^{t+1} &= {u_{i,k}}^t - \eta \frac{\partial \ell_{ij}}{\partial {u_{i,k}}^t} && \text{for } 1 \leq k \leq K \\ {v_{j,k}}^{t+1} &= {v_{j,k}}^t - \eta \frac{\partial \ell_{ij}}{\partial {v_{j,k}}^t} && \text{for } 1 \leq k \leq K \end{aligned}$$

Expanding the above expressions yields the update rule, namely

$$\begin{aligned} {u_i}^{t+1} &= {u_i}^t + 2\eta\Big(r_{ij} - \sum_{k=1}^{K} u_{i,k} v_{j,k}\Big){v_j}^t \\ {v_j}^{t+1} &= {v_j}^t + 2\eta\Big(r_{ij} - \sum_{k=1}^{K} u_{i,k} v_{j,k}\Big){u_i}^t \end{aligned}$$





Since this algorithm chooses a random point from the training set to update the weights, we must set a limit to the number of iterations that the algorithm is allowed to run for, i.e. how many times it updates the user and movie weights. Thus, it is necessary to examine the variation of training and test error with the number of iterations. This is shown in the figure above to the left. We can see that the performance of the predictor improves with increasing number of iterations, thus, for practical purposes, I pick 10,000,000 iterations.

Now, importantly, determining the value of the number of features, K, is crucial for improving the performance of the predictor. To do this, I cross validate for different values of K, specifically $1 \leq K \leq 15$. Similar to previously, I partition the training set into 10 disjoint set and train my predictor on 9 of the subsets. I validate on the remaining subset and repeat this such that all subsets have served as validation sets. The results of the cross validation are shown in the figure above to the right, and on average, the best-performing value of K is 11.

To obtain better generalization results, it may be useful to perform some regularization on the predictor. This can be done by modifying the loss function to include a regularization term. Specifically, we attempt to minimize the Lagrangian

$$\mathcal{L}_n(\mathbf{w}) = \hat{R_n}(\mathbf{w}) + \frac{\lambda}{n}||\mathbf{w}||^2$$

which effectively translates to solving the following expression

$$0 = \nabla\hat{R_n}(\mathbf{w}_{\text{reg}}) + 2\lambda'\mathbf{w}_{\text{reg}}$$

where $\lambda' = \frac{\lambda}{n}$. However, I modify this expression formula to fit the SGD context by replacing the empirical training error with the loss function, and instead of taking the gradient, taking the partial derivative with respect to each element of the vector. Thus, the new update rules for the user and movie weights that include regularization are

$$u_i{}^{t+1} = u_i{}^t + 2\eta\Big(r_{ij} - \sum_{k=1}^{K} u_{i,k}v_{j,k}\Big)v_j{}^t + 2\lambda' u_i{}^t$$
$$v_j{}^{t+1} = v_j{}^t + 2\eta\Big(r_{ij} - \sum_{k=1}^{K} u_{i,k}v_{j,k}\Big)u_i{}^t + 2\lambda' v_j{}^t$$

As was the case with the number of features K, cross validation is quite useful to determine an optimal value for λ'. Following exactly the same procedure as was described previously, but this time doing it for a range of different values of λ', ranging from $0 \leq \lambda' \leq 0.01$ in increments of 0.0005, cross validation reports the best-performing λ as around 0.009. Ideally, the increments should be smaller, however, since I am cross validating λ' for each value of K, it is impractical to do any larger increments since the running time increases significantly. With this addition, I obtain a test error of 0.8590, which is an improvement over all the previously employed methods.

3.5 General Summary

The performance results of each of the predictors presented in this problem have been summarized in the below table. It is observed that their performance rank corresponds to the order in which they have been presented, with the constant base predictor achieving the worst results, while the collaborative filtering predictor achieves the best results. Thus, to accurately be able to predict a user's movie preference, the collaborative filtering method should be employed.

Predictor Type	**Test Error**
Constant Base Movie Predictor	1.4410
Constant Base User Predictor	0.9267
Linear Regression Baseline	0.8875
Linear Regression with Transformed Features (Mult.)	0.8826
Linear Regression with Transformed Features (Sum)	0.8865
Collaborative Filtering	0.8590