Introduction to Machine Learning, Assignment 3

Diederik Vink

July 12, 2017

Pledge

I, Diederik Adriaan Vink, pledge that this assignment is completely my own work, and that I did not take, borrow or steal work from any other person, and that I did not allow any other person to use, have, borrow or steal portions of my work. I understand that if I violate this honesty pledge, I am subject to disciplinary action pursuant to the appropriate sections of Imperial College London.

Contents

1 Problem 1

Problem 1 of Assignment 3 aims to provide a high-probability upper bound on Equation (1).

$$R(h_\epsilon) - \min_{h \in \mathcal{H}} R_n(h) \tag{1}$$

where we let $R_n(h^*) = \min_{h \in \mathcal{H}} R_n(h)$ and let $\hat{R}_n(h^*) = \min_{h \in \mathcal{H}} \hat{R}_n(h)$. This gives

$$R(h_\epsilon) - R_n(h^*) \tag{2}$$

Furthermore, from Problem 1 we are given that:

$$\hat{R}_n(h_\epsilon) - \hat{R}_n(h^*) \leq \epsilon \tag{3}$$

Using the same steps for the proof at the bottom of Lecture 1 Slide 32:

$$\begin{aligned}
R_n(h_\epsilon) - R_n(h^*) &= R_n(h_\epsilon) + (\hat{R}_n(h_\epsilon) - \hat{R}_n(h_\epsilon)) - R_n(h^*) + (\hat{R}_n(h^*) - \hat{R}_n(h^*)) && \text{same procedure as "Proof" from Lecture 1 Slide 32} \\
&= (R_n(h_\epsilon) - \hat{R}_n(h_\epsilon)) + \hat{R}_n(h_\epsilon) + (\hat{R}_n(h^*) - R_n(h^*)) - \hat{R}_n(h^*) \\
&\leq Hoeff + Hoeff + R_n(h_\epsilon) - \hat{R}_n(h_\epsilon) && \text{as } R_n(h^*) - \hat{R}_n(h^*) \leq Hoeff \text{ and } \hat{R}_n(h_\epsilon) - R_n(h_\epsilon) \leq Hoeff \text{ from Lecture 1 Slide 31} \\
&\leq 2Hoeff + \epsilon && \text{using Equation (3)}
\end{aligned}$$

where $Hoeff = \sqrt{\frac{\log \frac{2M}{\delta}}{2n}}$. The reason we can say $\hat{R}_n(h_\epsilon) - R_n(h_\epsilon) \leq Hoeff$ and $R_n(h^*) - \hat{R}_n(h^*) \leq Hoeff$ comes from Lecture 1 Slide 32. Just like on that slide, in this situation we know that h is a hypothesis in the hypothesis class $\mathcal{H}$ so $|\hat{R}(h^*) - R(h^*)| \leq \sqrt{\frac{\log \frac{2M}{\delta}}{2n}}$ is true with the probability of $1 - \delta$.

2 Problem 2

The data points $y_1, y_2, \ldots, y_{n-1}, y_n \in \mathbb{R}$ are part of a set $\mathcal{Y}$ of size n. The aim of the problem is to minimize Equation (4)

$$\frac{1}{n} \sum_{i=1}^{n} |y - y_i| \tag{4}$$

where the loss function is $\ell(y, y_i) = |y - y_i|$.

Consider that the points are ordered so that $y_1 < y_2 < \ldots < y_{n-1} < y_n$ and $y < y_1$. $f(y)$ will be defined as follows:

$$f(y) = \sum_{y \in \mathcal{Y}} |y - y_i| \tag{5}$$

$f(y)$ can be rewritten as follows:

$$\begin{aligned}
f(y) &= \sum_{\forall (i \leq n) y_i \in \mathcal{Y}} |y - y_i| \\
&= \sum_{\forall (i \leq n) y_i \in \mathcal{Y}} (y_i - y) && (\text{as } \forall (i \leq n), y < y_i) \\
&= \sum_{i=1}^{n} (y_i - y)
\end{aligned}$$

From the equations above, it can be deduced that as y increases, each term of the summation will decrease, until $y = y_1$ where the summation above will no longer be true for $f(y)$. It can therefore be said that $f(y_1) < f(y)$ for all $y < y_1$.

Now for any $k < n$ we have

$$y_k \leq y \leq y + d \leq y_{k+1} \tag{6}$$

where $d \geq 0$. We can can see the following:

$$\begin{aligned}
f(y+d) &= \sum_{i=1}^{k}((y+d)-y_i) + \sum_{i=k+1}^{n}(y_i-(y+d)) \\
&= dk + \sum_{i=1}^{k}(y-y_i) - d(n-k) + \sum_{i=k+1}^{n}(y_i-y) \quad \text{as there are } k \text{ terms in } [1,k] \text{ and } n-k \text{ terms in } [k+1,n] \\
&= d(2k-n) + \sum_{i=1}^{k}(y-y_i) + \sum_{i=k+1}^{n}(y_i-y) \\
&= d(2k-n) + \sum_{i=1}^{n}|y-y_i| \\
&= d(2k-n) + f(y)
\end{aligned}$$

From the last line of this working out:

$$\begin{aligned}
f(y+d) &= d(2k-n) + f(y) \\
f(y+d) - f(y) &= d(2k-n)
\end{aligned}$$

Therefore for any y on the interval $[y_k, y_{k+1}]$ we can deduce that:

$$f(y) \text{ is } \begin{cases} \text{if } k < \frac{n}{2} \text{ then } f(y+d) < f(y) \text{ and } f(y) \text{ is decreasing} \\ \text{if } k = \frac{n}{2} \text{ then } f(y+d) = f(y) \text{ and } f(y) \text{ is constant} \\ \text{if } k > \frac{n}{2} \text{ then } f(y+d) > f(y) \text{ and } f(y) \text{ is increasing} \end{cases} \tag{7}$$

For any y on the interval $[y_k, y_{k+1}]$, if $k = \frac{n}{2}$ y will be in the middle of the data set and therefore the median. Using the middle case from (7) it can be seen that at the median, $f(y)$ is constant. Where $k < \frac{n}{2}$, $f(y)$ has a negative gradient and where $k > \frac{n}{2}$, $f(y)$ has a positive gradient. It is therefore clear that if y is on the interval $[y_{\frac{n}{2}}, y_{\frac{n}{2}+1}]$, $f(y)$ is at a minimum, y is the median and $k = \frac{n}{2}$.

3 Problem 3

3.1 Problem 3a

The constant base predictors that were used for this problem were the mean values. For the movie prediction, the mean of all non-zero movie ratings was taken. For the user prediction, the mean of all non-zero user ratings was taken. The resulting test errors for movie and user were least square errors, calculated as follows:

$$e_{user} = \frac{1}{|users|} \sum_{i=0}^{|users|} (\bar{y} - y_i)^2 \qquad e_{movie} = \frac{1}{|movies|} \sum_{i=0}^{|movies|} (\bar{y} - y_i)^2 \tag{8}$$

where $\bar{y}$ is the average result of all users or movies respectively and y_i is the rating for that user or movie respectively. After testing $e_{user} = 0.9286$ and $e_{movie} = 0.9993$. From these results it is clear that the user base predictor performs better and is the preferable choice. A possible explanation for this behavior is due to the training set provided. The training set contains 671 users and 9066 movies. As the number of movies is far greater than the number of users, on average each user has more ratings than a movie would. This means that a more accurate prediction can be with regards to their rating procedure. Furthermore, as these ratings are based on behavior, some users will on average provide higher ratings than other users. By basing the prediction on the user rather than the movie this bias is incorporated.

3.2 Problem 3b

General calculation for the linear regression baseline is done using the linear regression as stated in Lecture 7 Slide 5 as $w_{lin} = (Z^T Z)^{-1} Z^T \mathbf{y}$ with Z as the feature matrix (movie-features.csv) and $\mathbf{y}$ as the vector of ratings. For the w_{lin} calculation to hold it is mandatory that Z is orthogonal so that $Z^T Z$ is not singular and can be inverted. As this could not be guaranteed for the dataset provided, ridge-regression allowed for a guarantee of non-singularity and regularization. Ridge-regression was implemented as $w_{reg} = (Z^T Z + \lambda I)^{-1} Z^T \mathbf{y}$ (sourced from Lecture 7 Slide 5). Both w_{reg} and w_{lin} are weight vectors. The Z matrices used in the ridge-regression are specific to each movie. They are extracted from the overall dataset and ensure a unique w_{reg} is calculated for each movie.

The value of lambda is obtained using the validation technique of cross validation. The generic method of cross validation as described on page 149 of "Learning from data" [Abu-Mostafa, 2012] can be used. This is computationally quite expensive and as a result the analytic cross validation (E_{cv}) equation is used where $E_{cv} = \frac{1}{N} \sum_{n=1}^{N} \left(\frac{\hat{y}_n - y_n}{1 - H_{nn}(\lambda)} \right)^2$ where $H(\lambda) = Z(Z^T Z + \lambda I)^{-1} Z^T$ and $\hat{\mathbf{y}} = H(\lambda)\mathbf{y}$ (sourced from page 150, [Abu-Mostafa, 2012]). Z is always standardized according to the specifications in Lecture 7 Slide 11 to accentuate the effect of λ as is demonstrated in Lecture 7 Slide 11. Furthermore $\mathbf{y}$ is centered to remove bias present in $\mathbf{y}$. After the optimal λ for each movie has been found, ridge-regression is preformed for each movie finding $w_{reg} = (Z^T Z + \lambda I)^{-1} Z^T \mathbf{y}$ for each individual movie where Z is still the feature matrix and $\mathbf{y}$ is the vector of ratings for that movie. The training error was still calculated using the a least squared error. As users and movies are not longer independent the error, the same formula for e_{movie} can be used except now $\bar{y}$ is replaced with $\hat{y}$, where $\hat{y} = w_{reg}^T + Z_{standardized} + y_{average}$. See Lecture 7 Slide 11 for to see source of $\hat{y}$. Here again a standardized feature matrix was utilized to ensure consistency as the cross validation and ridge-regression were performed upon standardized data sets as well.

Running the tests as described above resulted in a test error $e = 0.8963$ and a training error $e = 0.8605$. As can be seen, this method clearly provides a better error than in Problem 3a as is expected. Furthermore, as can be seen, the test error is only circa 4% off of the training error indicating that there is very little overfitting taking place.

3.3 Problem 3c

Two main methods of non-linear transformations were used for the data in this report; Legendre polynomials and the combination of features with the number of features reduced using PCA to ensure that the feature set does not grow out of proportion.

Legendre

For the Legendre polynomial non-linear transformations the scipy library function “scipy.speical.legendre()” was used. The non-linear transformation consisted of calculating increasing degrees of Legendre transformations,

having each degree of Legendre polynomial form a feature for the new feature matrix.

$$Z = \begin{bmatrix} | & | & | & | \\ L_1 & L_2 & L_3 & L_4 \\ | & | & | & | \end{bmatrix}$$

where $L_0(x) = 1, L_1(x) = x, L_2(x) = \frac{1}{2}(3x^2 - 1), L_3(x) = \frac{1}{2}(5x^3 - 3x), L_4(x) = \frac{1}{2}(35x^4 - 30x^2 + 3), \ldots, L_q(x) = \frac{1}{2^q q!}\frac{d^q}{dx^q}(x^2 - 1)^q$. L_0 is not included because it has no effect upon the overall result. Because each Legendre takes the entire matrix as an input, each L_q is a transformation of the original matrix, of 18 columns. So if the highest degree is 4 for a particular transformation would result in a matrix of size 72. For each Legendre degree, a λ cross validation is performed as in Problem 3b. This nested cross validation allows for the best possible combination of Legendre polynomial matrix transformation and λ value. Legendre polynomials are designed so that they will always be orthogonal for piecewise smooth functions on [-1,1]. Due to this orthogonality, it is possible to be able to replace the least squares error calculation with Lasso for improved error analysis.

Polynomialization

The other form of transformation that was used was reducing the number number of features in the feature matrix using PCA and then creating a non-linear transformation by adding multiplying together the various columns through the "sklearn.preprocessing.PolynomialFeatures().fit_transform" function, with **interaction_only** set to true. For example, with the feature matrix reduced to 4 features through PCA creating

$$Z = \begin{bmatrix} | & | & | & | \\ a & b & c & d \\ | & | & | & | \end{bmatrix} \tag{9}$$

, where a, b, c and d are all vectors, Z can be expanded to 11 features with columns of

$$Z' = \begin{bmatrix} | & | & | & | & | & | & | & | & | & | \\ 1 & a & b & c & d & (a*b) & (a*c) & (a*d) & (b*c) & (b*d) & (c*d) \\ | & | & | & | & | & | & | & | & | & | \end{bmatrix} \tag{10}$$

The first column of Z' is a column of all ones. From now on in this report, this process will be referred to as polynomialization and the the degree of polynomialization indicates how many variables are multiplied together. The example is a polynomialization of degree 2. Just like with Legendre, for each degree on polynomialization, the analytical cross validation of λ is performed to acquire the optimal combination of polynomialization degree and λ value.

Just doing a PCA transformation (always reducing down to 4 features (degree = 4)) is not considered a non-linear transformation as it merely maps one column onto the other. Only once the columns are combined, it becomes a non-linear transformation. There is also the option of having columns multiply by themselves, but this produces massive matrices and was deemed less useful than crossing columns.

Results

After either of the non-linear transformations is completed, the transformed matrix is tested in exactly the same way as in Problem 3b (including the standardization of Z). The result of these tests are represented in the table below:

Non-linear Transform	**Polynomialization**	**PCA**	**Legendre**
Degree	3	4	10
Training Error	0.8734	0.8926	0.7974
Test Error	0.9074	0.9123	0.8832

Table 1: Performance of various non-linear transformations and PCA (for reference

As can be seen, the Legendre non-linear transformation is outperforming the polynomialization non-linear transformation. Compared to the linear regression, Legendre caused circa a 1.2% improvement, whereas Polynomialization was 1.5% worse. The reason that Polynomialization preforms so badly is that due to the PCA as this causes the test error to be worse that for normal linear regression. There is a decrease in error from PCA only to Polynomialization, but only of 0.5%. This is still less than the improvement Legendre has to offer, showing that Legendre truly is the better option.

3.4 Problem 3d

Collaborative filtering took a drastically different approach to learning. So far the initial feature matrix that was provided was accepted as the feature matrix Z, which provided a ratings matrix after $w_{reg}^T Z$ for each movie. Instead for collaborative filtering, a new feature matrix with K features is created for the movies (x) while simultaneously developing a weight matrix (θ) that contains the weight vectors of every user. The predicted ratings matrix is calculated through $\theta^T x$.

The collaborative filtering that was implemented for this report utilizes stochastic gradient descent as its learning algorithm. Stochastic Gradient Descent takes the gradient of your error function and updates both x and θ accordingly. The formula for the combined error function is as follows:

$$e = \frac{1}{2} \sum_{(i,j):r(i,j)=1} ((\theta_j)x_i - y_{(i,j)})^2 + \frac{\lambda}{2} \sum_{i=1}^{|movies|} \|x_i\|^2 + \frac{\lambda}{2} \sum_{j=1}^{|users|} \|\theta_i\|^2 \tag{11}$$

In Equation (11), $\frac{\lambda}{2} \sum_{i=1}^{|movies|} \|x_i\|^2$ and $\frac{\lambda}{2} \sum_{j=1}^{|users|} \|\theta_i\|^2$ are both regularizer terms for x and θ respectively. Furthermore, $(i,j) : r(i,j) = 1$ stands for selecting every (i,j) pair where the there is a rating for that pair. $(r(i,j) = 1)$. $y(i,j)$ is the rating for that specific user, movie pair. In this situation, i represents a specific movie and j a specific user.

The aim of any type of gradient descent is to minimize x and θ for movies and users. This is done via gradient descent, decreasing x and θ by subtracting scaled version of the gradient. A formulaic interpretations can be seen below:

$$x_i = x_i - \alpha \left(\sum_{j:r(i,j)=1} (\theta_j)x_i - y_{i,j)})\theta_j + \lambda(x_i) \right) \tag{12}$$

$$\theta_j = \theta_j - \alpha \left(\sum_{j:r(i,j)=1} (\theta_j)x_i - y_{i,j)})x_i + \lambda(\theta_j) \right) \tag{13}$$

where $\frac{d}{dx_i} = \sum_{j:r(i,j)=1} (\theta_j)x_i - y_{i,j)})\theta_j + \lambda(x_i)$ and $\frac{d}{d\theta_j} = \sum_{j:r(i,j)=1} (\theta_j)x_i - y_{i,j)})x_i + \lambda(\theta_j)$. What is shown in the equations above shows batch gradient descent. If the summations are removed and a random value of i for Equation (12) and a random value of j for Equation (13) are selected, the batch gradient descent becomes stochastic gradient descent. α is the scaling factor that is present to ensure that the value subtracted from x_i and θ_j is a reasonable amount, controlling the rate of decrease of x and θ.

The stochastic gradient runs for a specified number of iterations, updating both x_i and y_j for a random i and random j on each iteration. As a result on each iterations, x_i and θ_j are progressively minimized. Because on each iteration a new θ and a new x are used, this results in the two factors decreasing in collaboration with each other. Because a combined error function is used, any change in x affects θ and vice versa. This intertwined improvement is where the strength of collaborative filtering lies. As the two values change in correspondence with each other, when x_i is updated for a specific i, based on a random movie j, this will improve the rating for all movies, not just movie j. This is because user i's weight is more accurate for a feature all movies have access to rather than before when some movies had no connection to certain features. This means even though a user has only watched specific movies, the ratings the user has provided for some movies improves the the over quality of rating because the features for the movies are being defined alongside. In the old method, is a user had not watched any movies with a specific feature, the predictions for this feature were not justified. Furthermore, if a random user has watched a movie and provided a rating, this will affect the feature values for that movie. As a result, every rating to a movie, regardless of which user has rated it, will benefit all users. This is because as the feature values are updated for a specific movie, the accuracy of these feature values increases. And as the each user has its own set of weights for each feature, they will be get a more accurate movie rating prediction. This shows that simultaneously updating this information provides a far stronger setup of feature matrices and allows users to benefit off of the ratings provided by other users. There is the consideration to make that users will bias results as some users tend to generally rate movies higher than other users. Such biases can be dealt with separately.

Because the features matrix is now no longer the same as it has been up until this point, it is not possible to use the analytic cross validation formula. This has lead using normal cross validation negatively impacted the computation quite drastically. This is why for cross validation, the parameters are selected sequentially and not in a nested fashion. Even though this does not provide the optimal combination of parameters, it was the selected method as it provided a realistic runtime. In an ideal situation, this would have been avoided.

The cross validation used here takes out a set size of of 3,889 samples. This creates an 18 fold cross validation. This has possibly lead to inaccuracy in the results, as taking such a large chunk of data will result in possibly completely removing the ratings for some users from that specific test set missing the 3889 chunk on that iteration of the cross validation. Even for collaborative filtering, if no rating for a user is provided, the validation error will be quite misrepresentative, as the weight matrix for that user will not have been updated at all.

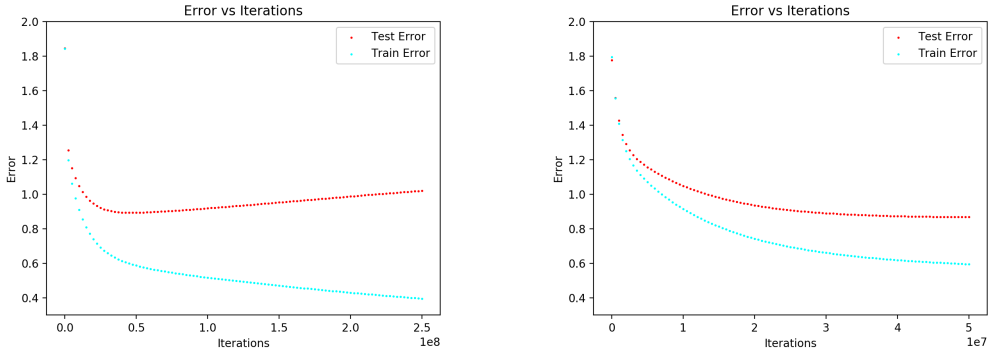
```
        lamda: 0.1     k: 8    lasso:  2       error: 3.07483789974    time:  340.157763004
        lamda: 0.05    k: 8    lasso:  2       error: 2.98128924678    time:  248.171981812
        lamda: 0.01    k: 8    lasso:  2       error: 3.21760173867    time:  246.994387865
        lamda: 0.005   k: 8    lasso:  2       error: 3.0508690178     time:  248.488093853
        lamda: 0.001   k: 8    lasso:  2       error: 3.12016788097    time:  222.228751898
        lamda: 0.0005  k: 8    lasso:  2       error: 3.10754978937    time:  217.263694048
        lamda: 0.05    k: 4    lasso:  2       error: 6.16894478976    time:  208.499955893
        lamda: 0.05    k: 7    lasso:  2       error: 3.72008512309    time:  217.551417112
        lamda: 0.05    k: 10   lasso:  2       error: 2.09351093379    time:  219.831302166
        lamda: 0.05    k: 14   lasso:  2       error: 1.50494288293    time:  225.842535019
k:  14  lamda:  0.05 lasso:  2  trainError:  0.595019504053  testError:  0.86837967716  iterations:  50000000  alphaScale:  0.0001
```

Figure 1: Results for collaborative filtering

The results shown above demonstrate the progressive cross validation used to select the optimal parameters and the final result that is produced, when running for 50,000,000 iterations with a scaling factor (α) of 0.0001. The value of α is always relative to the number of iterations to ensure that the values of x and θ do not decrease at too high of a rate. The lasso: 2, indicate that least squared errors was used as already mentioned. The progressive testing of each set of values shows how first the best λ is chosen and then the best value of K for that λ.

The final setup shows the test error using the best setup result from the cross validation test. The test error of 0.86837967716 is the lowest so far, improving on non-linear regression by 1.6% and a 3.1% improvement on normal linear regression. These are very good improvements showing the efficacy of redefining the feature matrix and simultaneously creating the features and weights matrices.

There were various parameter combinations that provided more favorable results, like K = 14, $\lambda = 0.1$ which resulted in a test error of 0.854598964465. This is a better result, but due to the fact that due to resource and time limitations the cross validation was done sequentially, and that training data will not always give perfect predictions explains why not ncessarily the best combination is produced by cross validation.



(a) 250 million iterations **(b)** 50 million iterations

Figure 2: graphs showing change in error per iteration.

Figure 2a shows that if too many iterations of stochastic gradient descent occur, the training error decreases drastically, but the test error increases. This is because the algorithm is overfitting to the data to an increasingly large degree. Figure 2b demonstrates that around 50 million iterations the change in test error stays quite stable and plateaus. This means that doing any more iterations will not improve the result, and potentially make it worse, as shown in Figure 2a.

Using batch descent could have provided better results, as this would have evenly updated every user and movie, unlike stochastic where this is done randomly. This guarantees using all the data provided. Stochastic gradient descent with enough iterations should take care of this, but it is not a guarantee.

References

Malik Hsuan-Tien Lin Abu-Mostafa, Yaser S Magdon-Ismail. *Learning From Data*. AMLBook.com, 2012.