Machine Learning CW3

Aditya Rajagopal (ar4414)

March 10, 2017

Pledge

I, Aditya Rajagopal, 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.

1 Problem 1

For ease of notation, let $\min\limits_{h\in\mathcal{H}} \hat{R_n}(h) = \hat{R_n}(h^*)$ and $\min\limits_{h\in\mathcal{H}} R_n(h) = R_n(h^*)$ where $h^* \in \mathcal{H}$ is the optimal hypothesis on the training set $\mathcal{D}$. It is given that $h_\epsilon \in \mathcal{H}$ is the ϵ-optimal hypothesis on the training set $\mathcal{D}$, i.e $\hat{R_n}(h_\epsilon) \leq \epsilon + \hat{R_n}(h^*)$.

$$\begin{aligned} R_n(h_\epsilon) - R_n(h^*) &= R_n(h_\epsilon) + [-\hat{R_n}(h_\epsilon) + \hat{R_n}(h_\epsilon)] + [-\hat{R_n}(h^*) + \hat{R_n}(h^*)] - R_n(h^*) \\ &= [R_n(h_\epsilon) \underset{(1)}{-} \hat{R_n}(h_\epsilon)] + [\hat{R_n}(h_\epsilon) \underset{(2)}{-} \hat{R_n}(h^*)] + [\hat{R_n}(h^*) \underset{(3)}{-} R_n(h^*)] \end{aligned}$$

As both h_ϵ and h^* are single hypotheses from the same hypothesis class $\mathcal{H}$, the terms (1) and (3) in the equation above can be bounded using the Hoeffding inequality. From the proof given in Lecture 1, slide 31 and the bound given in Lecture 1 slide 32, it is shown that terms (1) and (3) can each be upper bounded with probability of at least $1-\delta$ by the term $\sqrt{\frac{log\left(\frac{2|\mathcal{H}|}{\delta}\right)}{2n}}$. Term (2) is bounded by ϵ as defined in the problem.
Therefore, with probability at least $1-\delta$,

$$R_n(h_\epsilon) - R_n(h^*) \leq 2\sqrt{\frac{log\left(\frac{2|\mathcal{H}|}{\delta}\right)}{2n}} + \epsilon$$

2 Problem 2

Consider the set $\mathcal{Y}$ of n data points such that, $y_1 < y_2 < \ldots < y_n$. The error that is to be minimised is $f(y) = \frac{1}{n}\sum_{i=1}^{i=n} |y - y_i|$.

Consider the case when $y < y_1$,

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

As y increases, each term $y_i - y$ in Eq.1 decreases until the case when $y = y_1$. Therefore, $f(y_1) < f(y)$ for $\forall y < y_1$. Therefore, the minimum value of $f(y)$ cannot be for the case when $y < y_1$.

For any y, such that $y_k \leq y \leq y + d \leq y_{k+1}$

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

Therefore, $f(y+d) - f(y) = (2k-n)d$.

As $d \geq 0$, on the interval $[y_k, y_{k+1}]$

$$f(y)\ is \begin{cases} decreasing, & 2k < n \\ constant, & 2k = n \\ increasing, & 2k > n \end{cases}$$

Therefore, it is seen that $f(y)$ is minimised in the interval $[y_{\frac{n}{2}}, y_{\frac{n}{2}+1}]$ which for the set $\mathcal{Y}$ is the middle element of an ordered set, or in other words the median of the set. Therefore, the value m that minimises the error $f(y) = \frac{1}{n}\sum_{i=1}^{i=n}|y - y_i|$ is the median.

3 Problem 3

3.1 Constant Base Predictors

To generate base predictors, the mean rating of each user (independent of movie) and each movie (independent of user) was used. These values were calculated by sorting the training set first by users and calculating the mean rating each user gave across across all the movies they have rated. Then the mean rating for each movie was calculated as the mean of the ratings all the users who watched that movie gave it. These mean values were used as predictors for each movie and each user. The test error was calculated as the mean squared error, i.e. for user i and movie j in the test set of size n, the average user-based and movie-based test errors were calculated as

$$e_{user} = \frac{1}{n}\sum_{i=1}^{n}(y_{i,j} - \bar{y_i})^2 \qquad e_{movie} = \frac{1}{n}\sum_{i=1}^{n}(y_{i,j} - \bar{y_j})^2$$

where $\bar{y_i}$ is the average rating given by that user and $\bar{y_j}$ is the average rating for that movie as calculated from the test set. It is important to note that not all movies and users present in the test set were present in the training set, hence when calculating this average, the total number of points was $< n$ as users/movies not found in the training set were not considered. Using these predictors gave the following errors $\mathbf{e_{user} = 0.928571869917}$ and $\mathbf{e_{movie} = 0.999343927213}$. Therefore, e_{user} is a better constant base predictor that e_{movie}. A possible reason for this is that there are significantly more movies than users, therefore taking an average across every movie watched by a particular user would give a better estimate than calculating the average the other way around. Also, it is plausible that ratings given by users would be more consistent as the range of movies watched by users would potentially be restricted to movies the user likes or thinks they might like. Hence ratings for a particular user would be less varied, making it easier to predict.

3.2 Linear Regression Baseline

For this section, the algorithm that was used was ridge regression. Ridge regression performs linear regression on a dataset with a constraint on the weight vector known as a regulariser. As the error function being minimised is the squared error, the training error is defined as $\hat{R_n}(\mathbf{w}) = \frac{1}{n}(Z\mathbf{w} - \mathbf{y})^T(Z\mathbf{w} - \mathbf{y})$ where Z is the feature matrix, $\mathbf{w}$ is the weight vector and $\mathbf{y}$ is the label vector. There are multiple choices of regulariser that can be used such as the ℓ_2 regulariser which bounds the weights vector as $||\mathbf{w}||^2 \le C$ for some constant C and the ℓ_1 (Lasso) regulariser that bounds the weight vector as $\sum_{i=1}^{K}|\mathbf{w}_i| \le C$ when learning on K features. To save computation time, an analytical approach to minimsation was taken in this question as opposed to a numerical approach such as gradient descent. As shown in lecture 7 slide 23, an analytical approach to minimising using Lasso regularisation requires the assumption that $Z^TZ = I$, i.e. the movie feature matrix has orthonormal columns. This assumption is not true for the dataset provided and hence using a Lasso regulariser would mean using gradient descent. Therefore the ℓ_2 regulariser was chosen for this problem.

This meant minimising the function $\mathcal{L}(\mathbf{w}) = \hat{R_n}(\mathbf{w}) + \frac{\lambda}{n}||\mathbf{w}||^2$. Differentiating this term and setting it to 0 gives $Z^T(Z\mathbf{w} - \mathbf{y}) + \lambda\mathbf{w} = 0$. Therefore the weight vector that gives the minimum error is $\mathbf{w}_{reg} = (Z^TZ + \lambda I)^{-1}Z^T\mathbf{y}$, where λ is the regularisation parameter.

The algorithm followed for this question is as follows:

- **Cross validate across** λ to choose the best regularisation model for the situation. As the aim is to learn weights for each user, a matrix of λ values is created, where each λ corresponds to a user in the training matrix. The cross validation is done in the function *cross_val_lamda* in the script *learning.py*. For each user in the training matrix, a feature matrix and the corresponding y (rating) vector is created based on the movies that user has watched. This feature matrix is then normalised (in order to amplify the effect of λ as shown in lecture 7 slide 11) and the $\mathbf{y}$ vector centred to remove any bias before performing cross validation. In order to save computation time, the analytical formula found in [Abu-Mostafa, 2012], page 150, was used for cross validation. The formula is

$$E_{cv} = \frac{1}{N}\sum_{n=1}^{N}\left(\frac{\hat{y_n} - y_n}{1 - H_{nn}(\lambda)}\right)^2 \tag{2}$$

  where $H(\lambda) = Z(Z^TZ + \lambda I)^{-1}Z^T$ and $H_{nn}(\lambda)$ corresponds to the diagonal terms of this matrix. $\hat{\mathbf{y}} = H(\lambda)\mathbf{y}$ where $\mathbf{y}$ is the ratings vector for that user of dimension (Nx1). Using this, the λ that gave the lowest E_{cv} for a particular user was selected as the λ for that user. λ values in the interval (0, 10000] in steps of 10 were used when testing for a λ for each user. The range of λ values obtained ranged from a minimum of a 1000 to some users capping the range at 10000.

- **Weights for each user** were calculated using the λ matrix obtained from cross validation and the formula for $\mathbf{w_{reg}}$ given above. Again the feature matrix was normalised and the $\mathbf{y}$ vector was centred. When calculating test and training errors, it was ensured that the predictions obtained from the learned weights were shifted back.

The training and test errors obtained for this algorithm were $\hat{R_n}(\mathbf{w}) = 0.860467612583$ and test error of $\mathbf{R_n(w) = 0.896250407519}$.

This is a 3.48% improvement in the test error from the value of e_{user} obtained using constant predictors. Furthermore, it is useful to note that the training and test errors are close to each other hence showing that the predictor isn't overfitting the dataset. A final point to consider for this section is that linear regressions without regularisation was not possible to begin with. This is because for some users, the matrix $Z^T Z$ becomes singular, hence $\mathbf{w}_{reg}$ cannot be calculated for a $\lambda = 0$. This is why the range of λ tested on did not include $\lambda = 0$.

3.3 Linear Regression with Transformed Features

For this section, 2 transformations were tried and compared.

3.3.1 Legendre

All 18 features of the dataset were transformed into Legendre polynomials up to a certain degree and then training was performed on this augmented feature set. So for instance if the transformation was up to a legendre polynomial of the 4th degree, then the feature set would include $[x, \frac{1}{2}(3x^2 - 1), \frac{1}{2}(5x^3 - 3x), \frac{1}{8}(35x^4 - 30x^2 + 3)]$, hence resulting in 72 features. The property of the Legendre polynomials that makes it a viable transformation is that the polynomials are orthogonal to each other over the interval [-1,1]. As all the data points in the given feature matrix are either 0 or 1, this transformation results in a set of orthogonal features. Cross validation was performed across 3 different maximum degrees of the Legendre polynomials, i.e. up to degree 2,3,4, resulting in 16,54, and 72 orthogonal features respectively. For each of these polynomial degrees, the user λ's were again found using cross validation as discussed in the previous section. The results of this test were that the **optimal maximum degree** was **4** (72 features) and the training and test errors for this were $\hat{R_n}(\mathbf{w}) = 0.819647894663$ and $\mathbf{R_n}(\mathbf{w}) = \mathbf{0.883404162604}$.

This is a 1.4% improvement from the ridge regression method implemented before. A possible explanation for this improvement, could be due to the fact that there were a larger number of features to learn from. Furthermore, it may have been the case that the quality of the hypothesis was not dependent on the individual features only, but rather the combination of various features which is a consequence of using a polynomial. Another point to note is that due to the orthogonality of the features, a potential improvement to this test would be to use an ℓ_1 (Lasso) regualariser, as the Lasso regulariser regularises better with an orthogonal feature set.

As the original feature set is binary, the transformed set is again made up of only 2 different real values per Legendre polynomial. Hence the transformation, though polynomial, doesn't really spread the feature values. Hence this improvement was a little unexpected.

3.4 Collaborative Filtering

The idea behind collaborative filtering, is that the features chosen to learn on for each movie are not pre decided, but rather learnt in tandem with the weight vector for each user that corresponds to these learnt features. Rather, the only thing decided before hand is K, the number of features per movie that is to be learnt. The algorithm starts with 2 matrices, initialised with small random values. One matrix Θ is the user weights matrix of dimensions (K x NumUsers) made up of (K x 1) weights vectors for each user. The other matrix X is the movie feature matrix of dimensions (K x NumMovies) which is made of (K x 1) dimensional feature vectors.

The aim of collaborative filtering is to simultaneously learn the values of these 2 matrices so that the matrix $\Theta^T X$, consisting of each user's rating of every movie can be used to make predictions. The previous algorithm of ridge regression only trained each user's weight vector based on movies that user has watched. Hence, the users prediction on a movie that the user has never watched before, might not very accurate. Although the aim of the ridge regression algorithm is to learn the features so that the weights can be generalised to any movie's weights, on a small dataset such as this where a particular user may not have watched many movies, this generalisation can be poor.

Collaborative filtering on the other hand improves on this as even if a user i_1 has not watched a movie j, another user i_2's impression of movie j would train the feature vector of movie j (x_j). Now assuming that users are similar to a certain degree in the way they respond to movies, it is possible to see that the affect of user i_2's impression on x_j, would transform the vector x_j in such a way that knowing user i_1's weights for each feature, the prediction generated by $\theta_1^T x_j$ for user i_1 would be quite accurate. Thus the idea is that users collaboratively assist each other in rating movies that those users haven't seen.

The algorithm for collaborative filtering, for a training set of n users and m movies aims to minimise the loss function,

$$J(x_1, \ldots, x_m, \theta_1, \ldots, \theta_n) = \frac{1}{2} \sum_{(i,j):r(i,j)=1} (\theta_j^T x_i - y_{(i,j)})^2 + \frac{\lambda}{2} \sum_{i=1}^{m} ||x_i||^2 + \frac{\lambda}{2} \sum_{j=1}^{n} ||\theta_j||^2 \tag{3}$$

This loss function is a combination of the least squared error and ℓ_2 regularisation terms for both X and Θ. The factors of $\frac{1}{2}$ are included to make the derivative term nicer in the following steps. The term $(i,j) : r(i,j) = 1$ refers to every movie(i), user(j) pair for which user j has watched movie i. $y_{(i,j)}$ is the prediction user j gave movie i in the test set. Minimising this term with respect to $x_1, \ldots, x_m$ and $\theta_1, \ldots, \theta_n$ using gradient descent requires to take the partial derivative of Eq. 3 w.r.t movies and users. These partial derivatives, give the gradients used in gradient descent and the update step for the batch gradient descent algorithm is shown below.

$$x_i = x_i - \alpha \left(\sum_{j:r(i,j)=1} [(\theta_j^T x_i - y_{(i,j)})^2 \theta_j] + \lambda x_i \right)$$

$$\theta_j = \theta_j - \alpha \left(\sum_{i:r(i,j)=1} [(\theta_j^T x_i - y_{(i,j)})^2 x_i] + \lambda \theta_j \right)$$

where α is a parameter defined as the step size of gradient descent. It is important to note that both these steps need to happen in parallel, i.e. after updating movie vector x_i, the user vector θ_j needs to updated before the previous update of x_i is written to the matrix X. In the above equations, the terms multiplied by α, are the partial derivatives of the loss function 3. In batch gradient descent, for a fixed movie i, all users j that have watched this movie are used to calculate the partial derivative, and for a fixed user j, all movies i watched by this user are used to calculate the partial derivative. For a large dataset, it is evident that this is extremely computationally expensive. Hence the algorithm used is stochastic gradient descent, where rather than using all movies watched by a user or vice versa, a single movie is selected at random is selected from the set of movies watched by that user when calculating the partial derivative. Hence the above equations become,

$$x_i = x_i - \alpha \left((\theta_j^T x_i - y_{(i,j)})^2 \theta_j + \lambda x_i \right) \qquad \text{[for any j:r(i,j)=1]}$$

$$\theta_j = \theta_j - \alpha \left((\theta_j^T x_i - y_{(i,j)})^2 x_i + \lambda \theta_j \right) \qquad \text{[for any i:r(i,j)=1]}$$

The justification behind this algorithm is that the expected value of the gradient calculated using each partial derivative, is the same as the gradient calculated by the batch algorithm. Hence for a large enough number of samples, these two algorithms give a similar result.

There are a few parameters that were needed to be chosen carefully in order to achieve convergence to the minimum value. The first of these is K (the number of features) and the second being λ (regularisation parameter). The step size α and the number of iterations to run gradient descent for were chosen by trial and error to ensure that the algorithm didn't blow up or take too long to converge. The optimal values for these were $\alpha = \mathbf{0.001}$ for iterations upto 10^6 iterations, and $\alpha = \mathbf{0.0001}$ for iterations greater than this upto 4x10^7 (maximum used in the code). The other two parameters were chosen using cross-validation.

For cross validation, the first step is to choose the appropriate fold, i.e. how many data points do we remove for testing each time. Initially, $\frac{1}{6}$th of the training dataset was chosen, i.e. 11,667 points were taken out each time for testing. Having a larger cross-validating test set means that the cross validation (CV) error is very close to the test error, but because the data is trained on fewer points, this test error will be very large. On the other hand, a smaller validation set, means that the test error will be small, but the cv-error, may not be a very accurate representation of this. Hence there is a trade-off to be made. By trial and error, the initial choice of $\frac{1}{6}$th proved to be selecting non-optimal values of K and λ, hence the validation set size was decreased to $\frac{1}{18}$th, i.e. 3889. This was the final value for size of the validation set while cross validating for K and λ.

The implementation of the stochastic gradient descent algorithm itself is as follows.

- The first movie in the sorted training set was chosen, and a random user was picked that had watched the movie. This movie user pair was used to update the corresponding movie's feature vector.

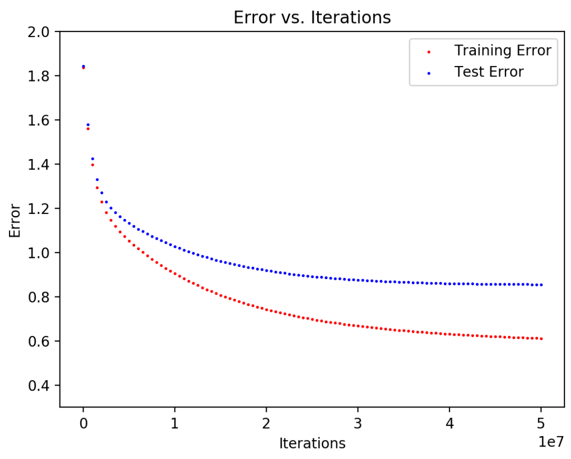
- The first user in the sorted training set was chosen, and a random movie was picked that the user had watched. This movie user pair was used to update the corresponding user's weight vector.
- The above 2 steps were performed for each iteration, and every iteration, the movie and user selected to update was sequentially incremented through the sorted training set

Multiple cross validation tests were carried out to carefully select ranges for the values of K and λ, however the final cross validation setup that resulted in choosing the best values of K and λ was as follows.

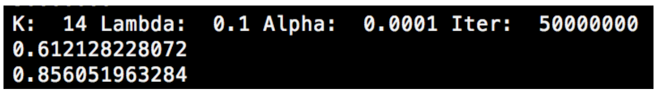
Sequential cross validation was performed where first for a random value of K the based value of λ from a range was selected using cross validation. Then using this selected value of λ, a value of K was selected using cross-validation between a range of K values. The setup was

- 10^6 iterations per cross validation training, 5x10^7 iterations when training on the entire set
- $\alpha = 0.001$ while cross validating, $\alpha = 0.0001$ when training on the entire data set
- List of λ's to cross validate over - [0.001, 0.0001, 0.005, 0.1, 1]
- List of K's to cross validate over - [10, 12, 14, 16]
- Initial random value of K chosen when cross validating on λ - $K = 15$

The results of this test are shown below:



(a)



```
K:  14 Lambda:  0.1 Alpha:  0.0001 Iter:  50000000
0.612128228072
0.856051963284
```

(b)

Figure 1

The value of **K** selected by cross validation was **14** (i.e. 14 features per movie) and the value of λ was **0.1**. The graph on the left shows the training and test error plotted against the number of iterations. As can be seen, the data is not being overfit as the test error plateaus, rather than deviating from the training error and increasing again. This setup results in a training error of $\hat{R}_n(\mathbf{w}) = 0.612128228072$ and $\mathbf{R_n}(\mathbf{w}) = \mathbf{0.856051963284}$. This is a 3.04% improvement from the error obtained using the Legendre non-linear transformation.

As expected, the collaborative filtering approach to solving this learning problem proved to be the best solution amongst all the methods discussed in this report. One point to note is that this combination of K and λ is not the most optimal combination of the 2 even though it produces the most optimal training error yet as they were obtained using sequential cross validation. The optimal combination would be better obtained using nested cross validation where for each K, the best λ is obtained. Doing this was however computationally expensive and limited the number of K's and λ's that could be validate over, hence as a compromise, sequential cross validation was used. Another potential improvement to the above method would be to implement batch gradient descent rather than stochastic gradient descent as this considers all points on every iteration and would result in better test error values.

References

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