Machine Learning Coursework 3

Jesus E. Garcia Condado

March 10, 2017

I, Jesus E. Garcia, 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

A ϵ-optimal hypothesis h_ϵ satisfies $\hat{R}_n(h_\epsilon) < \epsilon + \min_{h\in\mathcal{H}} \hat{R}_n(h)$ for any $\epsilon > 0$. We shall define the optimal hypothesis as $h^* = \arg\min_{h\in\mathcal{H}} R(h)$ and therefore $\min_{h\in\mathcal{H}} R(h) = R(h^*)$. Similarly define we will define the ERM hypothesis $g = \arg\min_{h\in\mathcal{H}} \hat{R}(h)$ and therefore $\min_{h\in\mathcal{H}} \hat{R}(h) = \hat{R}(g)$. Then the difference $R(h_\epsilon) - \min_{h\in\mathcal{H}} R(h)$ can be bounded by adding and subtracting equal terms, using the fact that by definition of our ideal hypothesis $R(g) \geq R(h^*)$ and therefore $0 \leq R(g) - R(h^*)$ and using the given bound for $\hat{R}_n(h_\epsilon)$:

$$\begin{aligned} R(h_\epsilon) - \min_{h\in\mathcal{H}} R(h) &= R(h_\epsilon) - \hat{R}_n(h_\epsilon) + \hat{R}_n(h_\epsilon) - R(h^*) \\ &\leq R(h_\epsilon) - \hat{R}(h_\epsilon) + \epsilon + \min_{h\in\mathcal{H}} \hat{R}_n(h) - R(h^*) \\ &= R(h_\epsilon) - \hat{R}(h_\epsilon) + \epsilon + \hat{R}(g) - R(h^*) \\ &= R(h_\epsilon) - \hat{R}(h_\epsilon) + \epsilon + \hat{R}(g) - R(g) + R(g) - R(h^*) \\ &\leq R(h_\epsilon) - \hat{R}(h_\epsilon) + \epsilon + \hat{R}(g) - R(g) \end{aligned} \tag{1}$$

Then we can use the Vapnik-Chervonenkis Inequality which states that:

$$\mathbf{P}\left(\left| \sup_{h\in\mathcal{H}} \hat{R}_n(h) - R(h) \right| > \epsilon \right) \leq 4 m_{\mathcal{H}}(2n) e^{\epsilon^2 n/8} \tag{2}$$

Where $m_{\mathcal{H}}$ is the growth function of the hypothesis class $\mathcal{H}$ which under certain conditions can be bounded as seen in the previous assignment.Then with probability $1-\delta$:

$$R(h) \leq \hat{R}_n(h) + \sqrt{\frac{8}{n} \log \frac{4 m_{\mathcal{H}}(2n)}{\delta}} \tag{3}$$

Since this bound applies to any $h \in \mathcal{H}$ including h_ϵ and g we can derive from equation 1 that with probability $1-\delta$:

$$R(h_\epsilon) - \min_{h \in \mathcal{H}} R(h) \leq 2\sqrt{\frac{8}{n} \log \frac{4m_{\mathcal{H}}(2n)}{\delta}} + \epsilon \tag{4}$$

2 Problem 2

Given data points $y_1, ..., y_n \in \mathbb{R}$ we can reorder their labeling such that $y_1 \leq y_2 \leq ... \leq y_n$. We want to minimize $f(y) = \sum_{i=1}^{n} |y - y_i|$. Lets consider a value m, which can be $m > y_n$, $m < y_1$ or $y_i \leq m \leq y_n$. For the first case we have that the sum

$$\forall m > y_n . f(m) = \sum_{i=1}^{n} m - y_i = nm - \sum_{i=1}^{n} y_i > f(y_n)$$

Similarly for $m \leq y_1$ we have

$$\forall m < y_1 . f(m) = \sum_{i=1}^{n} y_i - m = -nm + \sum_{i=1}^{n} y_i < f(y_1)$$

Therefore the value m that minimizes $f(m)$ lies in the final case $y_i \leq m \leq y_n$. Let y_k be the biggest y_i smaller than m so that $y_k < m < y_{k+1}$ and define d such that $y_k \leq m \leq m + d \leq y_{k+1}$. Then we have that:

$$\begin{aligned} f(m+d) &= \sum_{i=1}^{y_k} (m + d - y_i) + \sum_{i=y_{k+1}}^{n} (y_i - m - d) \\ &= dk + \sum_{i=1}^{y_k} (m - y_i) - d(n-k) + \sum_{i=y_{k+1}}^{n} (y_i - m) \\ &= d(2k - n) + f(m) \end{aligned}$$

If $2k > n$ then we have that $f(m + d) > f(m)$ so $m + d$ is worse than m and if $2k > n$ we have that $f(m + d) < f(m)$ and $m + d$ improves over $f(m)$. Hence we can pick an m such that $2k = n$ to minimie $f(m)$. This means that the value m that minimizes $f(m)$ is greater than or equal to $k = n/2$ numbers and smaller than or equal to $n - k = n - n/2$, that is it is the middle point or median if there is an odd number of points, and it is any point between the middle points if there is an even number of points.

Table 1: Results for constant predictors

Predictor	Train Error	Test Error	Avg n per predictor
movie	0.785	1.002	8.79
user	0.911	0.926	104.3

3 Problem 3

3.1 Part a

For a constant predictor that minimizes the mean square error, we want to find an w that minimizes $\frac{1}{n}\sum_{i=0}^{n}(w-y_i)^2$. Taking the derivative and setting it to zero we have that $0 = \frac{1}{n}\sum_{i=0}^{n} 2(w-y_i)$. Since w is independent of the summation variable we can rearrange as $nw = \sum_{i=0}^{n}(y_i)$ and therefore $w = \frac{1}{n}\sum_{i=0}^{n}(y_i)$. These means that the average of the training ratings minimizes the mean square error for a constant w predictor.

This was implemented with the given training set, for both movie and user constant predictors. A movie predictor for a given movie m therefore consists in the average of all the ratings received by the users who have rated that movie. Similarly for users each user predictor is the average of all the movies such user has watched. The results can be observed in table 1. The table also shows the average number of training samples for each predictor. Hoeffdings inequality gives a probability bound on the difference between train and test error which is proportional to $\frac{1}{\sqrt{n}}$ hence for a test error closer to the training error it is desirable to have a greater number of samples, denoted as n. Hence although the movie constant predictors obtain on average a better training error, the user predictor is still preferable since for every predictor it has 10 times more samples in average. This should led us to choose the user predictor without seeing the test results, since doing so based on the test results could lead to data snooping. The test results confirm that choosing the user predictor results in less test error.

3.2 Part b

We aim to find a linear regression predictor $u_i \in \mathbb{R}^d$ based on user i that minimizes the mean square error of $(r_{ij} - u_i^T v_j)^2$ where r_{ij} is the rating by user i to movie j and v_j is the feature vector of movie j. The solution that minimizes the training error of such prediction is the linear regression solution given by $u_i = V^+ y$ where V^+ is the pseudo inverse, $V^+ = (V^T V)^{-1} V^T$ of the matrix V composed of the feature vectures of all movies user i has watched. Hence V has dimensions $(movies\ watched) \times (movie\ features + 1)$. The plus one comes from the necessary 1 entries in the 0^{th} dimension.

An even better prediction can be obtain by avoiding the deterministic noise produces by over-fitting. An l2 norm was used as regularization, hence the new problem to minimize is $(r_{ij} - u_i^T v_j)^2 + \lambda |u_i|$. The solution can be found analytically as $u_i = (V^T V + \lambda I)^{-1} V^T y$. This limits the solution by imposing the constraint $u_i^T u_i \leq C$ where C is inversely proportional to λ. Hence for $\lambda = 0$ it is unconstrained and equivalent to the original problem.

The value of lambda is a parameter that must be determined carefully. More-

over, since we have a predictor for each user, each trainer might use a different lambda. Changing the lambdas to minimize the test error would not be appropriate, since it would be using the test set for training and the test error would not be representative of the real test error. A better alternative is to use cross validation. Through this method we can estimate the test error by splitting the training set in K and training our predictor on subsets of the split training set which we will call $\mathcal{D}_{train}$, leaving out a different part in ever training which we will call $\mathcal{D}_{val}$ to use as a test error. A hypothesis trained on $\mathcal{D}_{train}$ is then g^-. We can then use $\hat{R}_{val}(g^-)$ to estimate the test error since:

$$\mathbb{E}_{\mathcal{D}_{val}}\left[\hat{R}_{val}(g^-)\right] = R_{val}(g^-) \tag{5}$$

Then for different values of lambda this validation test error can be minimized to minimize the expected test error. We also know that given that m is the training sample size $m = |\mathcal{D}_{train}|$ then with probability at least $1 - \delta$:

$$R_{val}(g^-) \leq \hat{R}_{val}(g^-) + \Omega(m, \mathcal{H}, \delta) \tag{6}$$

Since $\Omega(m, \mathcal{H}, \delta) \propto \sqrt{\log(1/\delta)/m}$, a bigger m results in a validation training error closser to the real test error. Hence we want to maximize m by splitting the data in many smaller sets, so that taking a single set does not greatly reduce m. For example for a training set n with a split in K groups, the training set is $m = n - n/k$, hence we want to maximize K. If it is computationally possible, we can make $K = n$ which makes the validation set be of size one and is referred to as leave one out cross validation. This was the method used to choose the different lambdas for each user, and the final test error was calculated by training the hypothesis with the chosen λ over the whose training set. This yield an error of 1.053 which is not better than the constant predictor. It was also noted that the *lambdas* chosen tended towards being the smallest number in the range.

This is so because of the uneven effect of λ in the features of different energy, since we can derive the relation between the regularize and not regularized predictors as $w_{reg} = Vdiag\big(\frac{d_i^2}{d_i^2+\lambda}\big)V^T w_{lin}$, where V is obtained from the singular value decomposition of the training matrix and d_i^2 is the energy of the features. This can be solved by regularizing the training matrix to have unit variance and zero mean. The **scikit-learn** tools, specifically the class *preprocessing.standardScaler* can be used to create a scaler given the users training features, which results in a normalized distribution and this can be reused for any new input feature in the predictions. Moreover the ratings where centered by subtracting the mean. Applying this method resulted in a new best test error of 0.898.

3.3 Part c

Performing a nonlinear transform on the feature vectors by considering powers of the features was tried out to find no improvement for two reasons. On one hand since the features are restricted to the set $0, 1$ the powers of such variables remain in the set and are the same value for powers without interaction (ie v_3^2).

Moreover the given 18 features can already shatter the 600 movies since there are 2^{18} possible combinations. Hence the regularization in part a was necessary to reduce the over fitting. Trying out different powers with and without interactions gave no improvement and actually worsened the error to values over 1.1. It also considerably increases the execution time due to the additional computations necessary. Using cross validation to determine the hypothesis class (which powers to choose or choosing the untransformed features) as well as on the λ values confirmed that the transforms do not benefit the test error. This allowed choosing the best hypothesis class (the untransformed regularized class from part a) without using the test set to determine so.

A different approach is to first reduce the number of features first and then perform non linear transforms in this reduced set. A first idea was to represent each vector by an integer represented by the binary interpretation of the vector. Since the 18 values are either 1 or 0 each movie vector can be identified by a single integer. This however was unsuccessful for different orders of nonlinear transforms obtaining errors over 2.5. Again cross validation chose the non-transformed vector hypothesis class. This transform is not a good idea since the features used as most significant bits of the integer modify the integer more than those with lower powers of 2. Hence another approach was used to reduce the features based on summing groups of feature vectors. For example a single feature can be obtained by summing all the features or two features can be obtained by summing the first 9 features and then the other 9. Again the results are influenced by what group of features you choose to add together, and the order of the transforms used. Using a range of different summations and a range of orders of transformation the different hypothesis classes were tried through cross validation. The choice of transform varied according to user and a final test error of 0.912 was obtained. Although this is a worse error than with the initial features it reduced number of features which means it is computationally easier. If the original non-transformed hypothesis class was included in the cross validation it was always chosen, and the error from part a obtained. Hence the conclusion is that such hypothesis class is preferable over the transforms tried. A further approach would be to use Principal Component Analysis to obtain the initial features to perform transformations on.

3.4 Part d

Previously we have used the given features for each movie. A collaborative filtering approach can allow as to simultaneously derive features for each movie v_j and weights for each of such features for each user u_i. The aim is to learn the K dimensional vectors u_i and v_j where K is the number of features we have learned from each movie. We therefore wish to minimize the training error over the ratings in th training set $\mathcal{T}$ with $|\mathcal{T}| = n$, that is for each rating $y_{ij} \in \mathcal{T}$ of user i to movie j. We will also add the regularization term to avoid over-fitting, which constraints the solution:

$$\hat{R}_n(u, v) = \frac{1}{n} \sum_{y_{ij} \in \mathcal{T}} (u_i^T v_j - y_{ij})^2 + \sum_{u_i \in u} \frac{\lambda}{n} |u_i|^2 + \sum_{v_j \in v} \frac{\lambda}{n} |v_j|^2 \tag{7}$$

The minimum can be found by gradient descent to minimize $\hat{R}_n(u, v)$ over all u_i and all v_j. The partial derivatives with respect to each u_i and j_i are:

$$\frac{\partial \hat{R}}{\partial u_i} = \frac{2}{n}\Big(\big(\sum_{y_{ij} \in \mathcal{T}} (u_i^T v_j - y_{ij}) v_j \big) + \lambda |u_i| \Big) \tag{8}$$

$$\frac{\partial \hat{R}}{\partial v_j} = \frac{2}{n}\Big(\big(\sum_{y_{ij} \in \mathcal{T}} (u_i^T v_j - y_{ij}) u_i \big) \big) + \lambda |v_j| \big) \tag{9}$$

Then we can minimize the training set by updating all u_i and all u_v by steps in the direction of descent with a parameter α that determines the step size, and also accounts for the $2/n$ term. For every u_i and every v_j in every step we perform:

$$u_i = u_i + \alpha\Big(\big(\sum_{y_{ij} \in \mathcal{T}} (u_i^T v_j - y_{ij}) v_j \big) + \lambda |u_i| \Big) \tag{10}$$

$$v_j = v_j + \alpha\Big(\big(\sum_{y_{ij} \in \mathcal{T}} (u_i^T v_j - y_{ij}) u_j \big) + \lambda |v_i| \Big) \tag{11}$$

This equations can be interpreted as updating each user weight in the direction that decreases the error of all movies it has seen (the summation term) and further reducing it by a value proportional to its norm to avoid having a high norm, and therefore the ability to overfit. It is worth noting that the first terms of the summation are the same for any i, j pair, and therefore can be computed only once per iteration for a faster implementation that allows running more steps. More over the step size of the gradient descent is variable since we are updating each value proportionally to the derivative times alpha. Normalizing this by dividing it by the normal of the derivative resulted in worst error since the fixed step sizes converge slower to the minimum value.

These update rules result in a new hypothesis class for any K and λ values. The α value only determines the efficiency of the implementation. Hence K and λ can be determined through cross validation as in part b and α can be tuned based on the performance. For an α step of 0.001 and 100 iterations the best error was obtained with $K = 4$ and $\lambda = 0.01$ and the error descended from 8.94 to 1.43. This is way worse than previous attempts but I believe it is because of a bug in my code or an incorrect range of values or set size for the cross validation. Cross validation was performed with $K = 10$ since leave one out was computationally unfeasible given the training set has over 9000 movies and 600 users, and therefore each iteration requires over 1500 updates. Multithreading was also used to speed up the process by using the multiple cores in the computer to run different independent cross validation runs in parallel.