Sample Project: Movie Recommendation
Solution, Spring, 2018

Consider the movie recommendation problem studied in class: in this problem the goal is to predict the ratings users assign to movies in a minimum mean-squared error sense. In the provided dataset (`movie-data.zip`), viewer ratings corresponding to 9066 movies and 671 users are provided together with a feature vector for each movie (whose components are indicators of different genres).[1] The data is given in `csv` format with a header row describing the meaning of each column:[2]

- `movie-titles.csv`: title and id of the movies in the dataset (only for information purposes);
- `movie-features.csv`: binary (0-1-valued) genre features for the movies, each row contains variables `movieId,feature1,...,feature18`;
- `ratings-train.csv`: training set $\mathcal{D}$, containing `(userId,movieId,rating)` triplets in each row;
- `ratings-test.csv`: test set $\mathcal{T}$, containing `(userId,movieId,rating)` triplets in each row.

Here `userId` $\in \{1, 2, \ldots, 671\}$, `movieId` $\in \{1, 2, \ldots, 9066\}$, and `rating` $\in \{0.5, 1, 1.5, \ldots, 5\}$.

The task in this problem is to develop and test different predictors for the movie ratings whose performance is measured by the squared error. Training should only be performed on the training set, final test results should be measured on the test set, and no parameter of the training method (including which algorithm to use) should be selected based on the test set. For each subquestion, explain clearly and concisely what you do and why (including formulas with derivations if necessary), and present the answers in the most meaningful way.

(a) Constant base predictors: The simplest predictor for this problem is to provide a rating prediction for any movie (independent of the user) or any user (independent of the movie). Find such predictors and report their performance. Which one would you prefer, constant ratings based on the movies or the users?

Solution: This problem was designed to be similar to a real machine learning problem. In this task, we have to examine one of the simplest predictors that may come to one's mind. Those quick solutions usually serve as baseline, benchmarking the performance of later, more complex approaches. In the current problem, such a predictor can be defined in two ways: (i) For each movie, take the average of the ratings in the training set and use it as the prediction for the ratings of the movie (this is, e.g., what happens if you look at the average rating of movies on the IMDB website). (ii) For each user, average the ratings given by the user to the movies in the training set, and use this to predict ratings for each new movie and the user considered.

To find the better predictor from these two options, we can use cross-validation and choose the one with the lower cross-validation error (you must not do this based on the test error), which is averaging by users.

[1] The data is extracted from a dataset provided by GroupLens about ratings collected by MovieLens `https://grouplens.org/datasets/movielens/`. For licensing and other information, see `http://files.grouplens.org/datasets/movielens/ml-latest-small-README.html`.

[2] Recommendation: use built-in csv-reader functions to read the files.

An implementation remark: for those movies that are not in the training set, you can use, e.g., the average rating given by all the other users. Table 1 shows the 10-fold cross-validation error and the test error for both predictors. Notice that the cross-validation error is quite a good estimate of the test error.

Table 1: Cross-validation error of the constant predictor.

Method	Based on the movies	Based on the users
Cross-validation error	1.029	0.9295
Test error	1.0170	0.9269

(b) Linear regression baseline: Based on the features provided for the movies, estimate the rating given by each user using linear regression, possibly with some added regularization. That is, if $\hat{v}_j \in \mathbb{R}^d$ is the (possibly normalized) feature vector for movie j, find weights $u_i \in \mathbb{R}^d$ for user i with the aim of minimizing the error $(r_{ij} - u_i^\top \hat{v}_j)^2$ over the test set $\mathcal{T}$.

Solution: Among the possible regularized regression models covered in the class, we choose to adopt ridge regression which is defined by the minimization objective $\sum_{j=1}^{n_i}(r_{ij}-u_i^\top \hat{v}_j)^2+\frac{\lambda}{d}\|u_i\|_2^2$ over the training set $\mathcal{D}$ (n_i is the number of movies the ith user rated in the training set and in this summation the movies are renumbered between 1 and n_i; throughout we normalize the value of the the coefficient of the penalty term by d, the number of terms computed in the penalty, to get a more unified range of regularization parameters for different values of d). This choice is motivated by the fact that with only 18 features, we do not expect the feature selection property of lasso and elastic net would improve the performance. Also, ridge regression enjoys the advantages of a closed form solution, which, among others, significantly speeds up experimentation.

As a good practice, we normalize the data by centering and scaling each feature in the training data. This is done by removing the sample mean and diving by the standard deviation. The ratings are all centered as well. Note that this normalization procedure is carried out separately for each user (since we treat each user as a completely separate task). Also note that the test data must be normalized with the same values (obtained from the training set), and you must not recompute the normalization function using the test data. Finally, in order to choose a proper value of λ, we performed 10-fold cross-validation over the set of potential values $\{0.1, 0.5, 1, 5, 10, 50, 100, 500, 1000, 5000, 10^4, 5\times 10^4, 10^5, 5\times 10^5, 10^6, 5\times 10^6\}$. After cross-validation, the model with the optimal λ parameter is retrained on the whole training data.

Figure 1 shows the cross-validation error for different values of λ; notice that the error first decreases then increases, which suggests that we managed to cover the interesting range of λ (here we use the same penalty coefficient λ for all users–one could also use a different λ per user). The large values of λ are due to the fact that in our formulation they balance the cumulative error; the usual small values of λ should be considered if we used the average error in the optimization.

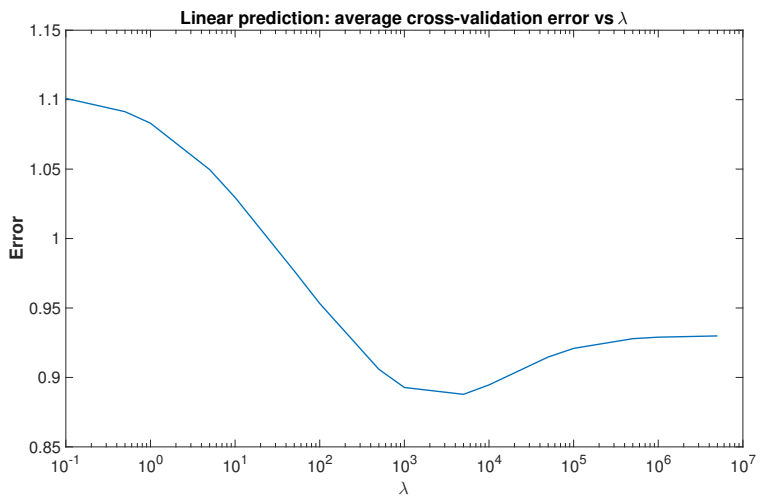

Figure 1: Cross-validation error for ridge regression as a function of λ.

Method	Ridge	Lasso	Elastic net
Cross-validation error	0.8872	0.9201	0.8951
Test error	0.8812	0.9153	0.8902
Best parameter(s):	$\lambda = 5000$	$\lambda = 5$	$\lambda = 10$, $\alpha = 0.1$

Table 2: Performance and optimal parameters of different regularized regression models.

For comparison, we repeat the experiment also with lasso and elastic net (to demonstrate that our original argument was reasonable). The objective functions for lasso and elastic net are

$$\sum_{j=1}^{n_i}(r_{ij} - u_i^\top \hat{v}_j)^2 + \frac{\lambda}{d}\|u_i\|_1$$

$$\sum_{j=1}^{n_i}(r_{ij} - u_i^\top \hat{v}_j)^2 + \frac{\lambda}{d}\left(\frac{1-\alpha}{2}\right)\|u_i\|_2^2 + \frac{\lambda\alpha}{d}\|u_i\|_1$$

respectively, and $\alpha \in (0, 1)$ is a parameter controlling the relative strength of the two regularization terms. Note that for elastic net, $\alpha = 1$ recovers lasso and $\alpha = 0$ recovers ridge regression. Due to this extra parameter, elastic net needs a cross-validation procedure over the (λ, α) pairs. For the purpose of the experiment, α has been cross-validated over the set $\alpha = \{0.1, 0.3, 0.5, 0.7, 0.9\}$. Table 2 shows the results of linear regression with different regularizations.[3]

For the implementation, we relied on the built-in MATLAB functions `ridge()`, `crossvalind()` and `lasso()`. These functions, respectively, implement ridge regression, a data splitting procedure to perform k-fold cross-validation, and a procedure to train lasso/elastic net.

[3]You may notice that the running time of lasso is much longer than the running time of ridge regression; this happens since lasso has no closed form solution and iterative algorithms are used to find a minimizer.

(c) Linear regression with transformed features: Suggest some way of transforming the features in a non-linear manner and repeat the above with the transformed features.

Solution: One option for generating new features is to consider pairwise products of features. Note that due to the binary nature of the original features, *polynomial* transformations involving only powers of a single feature do not provide additional information.[4] The rationale for using product features (i.e., products of features) is related to the intuition that a given user might not be interested in a purely horror or a purely action movie, but s/he might like an action-horror movie. In other words, products create new features representing mixed genres by means of a logical AND, which cannot be expressed by linear combinations. Computing the products can easily be done using the MATLAB function `x2fx(·,'interaction')`. Since the learning methods we use center the data (both the features and the output), the constant features are not needed, so we can remove them from the data. This transformation leads to a set of 171 features (including the original features and products of two features, but no squares), which we are going to use. As a result, it might be useful to consider again lasso and elastic net, since we have now so many features. The rest of the experimental setup is the same as in the previous section.

The results are reported in Figure 2 and Table 3. The former shows the cross validation error for different values of λ in the case of the best predictor, while the latter reports the minimum cross-validation error and the related test error for each of the considered regularized predictors. Ridge regression still seems the best, as the transformation does not improve the performance as one might have expected.

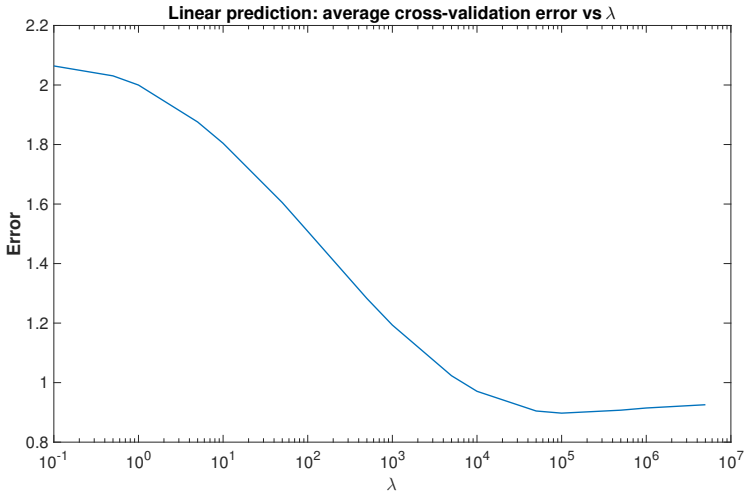

Figure 2: Cross-validation error for ridge regression on the transformed features as a function of λ.

[4]This is true not because the features are 0-1 valued–which changes if you normalize them–, but because they can take only two values; in this case any higher order polynomial can be expressed by a first order polynomial: if $p(x)$ is the polynomial and $b_1 \neq b_2$ are the two possible values of the feature x, then there exist coefficients a_0 and a_1 such that $p(b_i) = a_0 + a_1 b_i, i = 1, 2$ (just solve the equations to get a_0 and a_1).

Method	Ridge	Lasso	Elastic net
Cross-validation error	0.8970	0.9230	0.9087
Test error	0.8908	0.9188	0.9013
Optimal parameter(s):	$\lambda = 100000$	$\lambda = 50$	$\lambda = 100$, $\alpha = 0.1$

Table 3: Performance and optimal parameters of different regularized regression models on the transformed features.

(d) Collaborative filtering: Learn simultaneously the weights for the users and the features for the movies by devising predictions of the form $u_i^\top v_j$ where $u_i, v_j \in \mathbb{R}^K$ for some $K \geq 1$ and v_j is the K-dimensional (learned) feature vector for movie j.

In all the above questions, select the parameters (such as K, coefficients for penalties, step size for gradient descent, etc.) in a disciplined way (e.g., using cross-validation) and do not forget to report how these are done. If you are using gradient descent, do not forget to derive the update rule and give the exact algorithm you are using. Analyze the results and report your findings (including a comparison of the solutions you get at different stages of this problem). You can use any available standard library functions in your code (e.g., for ridge regression).

Solution: The first training objective we consider is the penalized empirical risk

$$R = \frac{1}{2|\mathcal{I}(\mathcal{D})|} \sum_{(i,j)\in\mathcal{I}(\mathcal{D})} (r_{ij} - u_i^\top v_j)^2 + \frac{\lambda}{2K} \left(\sum_{i=1}^{n_u} \|u_i\|_2^2 + \sum_{j=1}^{n_m} \|v_j\|_2^2 \right),$$

where $\mathcal{I}(\mathcal{D})$ is the set of all $(user, movie)$ pairs in the training data, and n_u and n_m are the number of users and movies in the data, respectively.

We optimize R using stochastic gradient descent. Letting

$$\ell_{i,j} = \frac{1}{2}(r_{ij} - u_i^\top v_j)^2 + \frac{\lambda}{2K} \left(\sum_{i=1}^{n_u} \|u_i\|_2^2 + \sum_{j=1}^{n_m} \|v_j\|_2^2 \right),$$

if (I, J) is chosen uniformly at random from $\mathcal{I}(\mathcal{D})$, then $\mathbb{E}[\ell_{I,J}] = R$, and so we can update all parameters using the gradients of $\ell_{I,J}$: here

$$\frac{\partial \ell_{i,j}}{\partial u_k} = \begin{cases} -(r_{ij} - u_i^\top v_j)v_j + \frac{\lambda}{K} u_i & \text{if } i = k; \\ \frac{\lambda}{K} u_k & \text{if } i \neq k. \end{cases}$$

$$\frac{\partial \ell_{i,j}}{\partial v_l} = \begin{cases} -(r_{ij} - u_i^\top v_j)u_i + \frac{\lambda}{K} v_j & \text{if } j = l; \\ \frac{\lambda}{K} v_l & \text{if } j \neq l. \end{cases}$$

Then, in every iteration we can update $u_k \leftarrow u_k - \eta \frac{\partial \ell_{I,J}}{\partial u_k}$ for all $1 \leq k \leq n_u$ and $v_l \leftarrow v_l - \eta \frac{\partial \ell_{I,J}}{\partial v_l}$ for $1 \leq l \leq n_m$. Note that because of the penalty, now we have non-zero gradients for u_k and v_l even when $k \neq i$ and $l \neq j$, whose effect is to reduce the coefficients that do not affect the selected rating directly. The results for 5-fold cross-validation are given in Table 4[5] using a constant step

[5]Training was performed with normalized rating values, but normalizing the variance is not important.

λ	$K=2$	$K=4$	$K=5$	$K=10$	$K=20$	$K=30$	$K=40$
0.0001	0.9061	1.0706	1.1329	1.4442	2.2718	3.5052	5.6263
0.0002	**0.8853**	1.0002	1.0488	1.3401	1.9717	2.8961	4.9890
0.0005	0.8922	0.9435	0.9708	1.1635	1.5695	2.4563	3.6421
0.0008	0.9163	0.9234	0.9396	1.0898	1.3704	2.0029	2.8435
0.001	1.0125	0.9133	0.9232	1.0530	1.2858	1.8150	2.4929
0.005	1.1204	1.0956	1.0491	0.9243	0.9183	0.9597	1.0106
0.01	1.1204	1.1204	1.1203	1.0621	0.9217	0.8982	0.9069

Table 4: 5-fold cross-validation error for the first method in collaborative filtering.

size $\eta = 0.01$ and stopping if the error in $1,000,000$ steps does not reduce more than 0.0001 (other step sizes, e.g., $\eta = 0.001$ give similar values; 5-fold cross-validation is used to reduce computation). The best result is obtained for $K = 2$ and $\lambda = 0.0002$. Retraining the model with these parameters yields a test error 0.8557, which is better than our previous attempts.

Naturally, one may use different penalties, e.g., penalizing the parameters of users and movies depending on how many times they appear in the data set. Denoting by $\mathcal{D}_i$ and $\mathcal{D}_j$ the subsets of the data containing user i and, respectively, movie j, one can consider the new penalized objective

$$R' = \frac{1}{2|\mathcal{I}(\mathcal{D})|} \sum_{(i,j)\in\mathcal{I}(\mathcal{D})} (r_{ij} - u_i^\top v_j)^2 + \frac{\lambda}{2K} \left(\sum_{i=1}^{n_u} \frac{|\mathcal{D}_i|}{|\mathcal{D}|} \|u_i\|_2^2 + \sum_{j=1}^{n_m} \frac{|\mathcal{D}_j|}{|\mathcal{D}|} \|v_j\|_2^2 \right),$$

which scales the penalty corresponding to the parameters better in the sense that both the empirical risk and the corresponding penalty scale roughly with the number of times a parameter appears. Defining now

$$\ell'_{i,j} = \frac{1}{2}(r_{ij} - u_i^\top v_j)^2 + \frac{\lambda}{2K}(\|u_i\|_2^2 + \|v_j\|_2^2),$$

one can see again that if (I, J) is selected uniformly at random from $\mathcal{D}$, then $\mathbb{E}\left[\ell'_{I,J}\right] = R'$, so we can optimize R' using stochastic gradient descent by making updates using the gradients of $\ell'_{I,J}$ ((I, J) selected uniformly at random):

$$\frac{\partial \ell'_{i,j}}{\partial u_k} = \begin{cases} -(r_{ij} - u_i^\top v_j)v_j + \frac{\lambda}{K} u_i & \text{if } i = k; \\ 0 & \text{if } i \neq k. \end{cases}$$

$$\frac{\partial \ell'_{i,j}}{\partial v_l} = \begin{cases} -(r_{ij} - u_i^\top v_j)u_i + \frac{\lambda}{K} v_j & \text{if } j = l; \\ 0 & \text{if } j \neq l. \end{cases}$$

The added benefit of this formulation is that it is computationally much more efficient: in every step we only need to update u_I and v_J: $u_I \leftarrow u_I - \eta \frac{\partial \ell'_{I,J}}{\partial u_I}$ and $v_l \leftarrow v_J - \eta \frac{\partial \ell'_{I,J}}{\partial v_J}$ (since all the other gradients are zero). The results of 5-fold cross validation with $\eta = 0.01$ are shown in Table 5, with the best result achieved for $K = 4$ and $\lambda = 1$.[6] Retraining on the whole training set using the same

[6]In both approaches, cf. Table 4, one can optimize λ further by adaptively refining the interesting region; the region for K cannot be refined in this case, since all values between 2 and 5 are considered. In our case this has only lead to limited variations in the results and has not changed the best value of K; this could have been different, since the best λ values for $K = 2, 3, 4$ have almost the same CV error in the table.

λ	$K=1$	$K=2$	$K=3$	$K=4$	$K=5$	$K=10$	$K=15$	$K=20$	$K=25$
0.001	1.0211	0.8874	0.9488	0.9938	1.0392	1.1935	1.3212	1.3918	1.4590
0.005	1.0189	0.8908	0.9417	0.9835	1.0312	1.1995	1.3086	1.4035	1.4564
0.1	0.9948	0.8968	0.9040	0.9506	0.9821	1.1397	1.2644	1.3395	1.3915
0.5	0.9947	0.8471	0.8527	0.8766	0.9053	1.0214	1.1089	1.1889	1.2534
1	1.0691	0.8946	0.8495	**0.8459**	0.8535	0.9434	1.0063	1.0737	1.1205
5	1.1202	1.1161	1.1026	1.0744	1.0439	0.8791	0.8480	0.8509	0.8659
10	1.1204	1.1198	1.1177	1.1127	1.1048	1.0348	0.9416	0.8851	0.8619
50	1.1204	1.1204	1.1204	1.1204	1.1203	1.1180	1.1103	1.0981	1.0854

Table 5: 5-fold cross-validation error for the second method in collaborative filtering.

Method	Constant movie	Constant user	Ridge regression	Ridge regression new features	Collaborative filtering 1	Collaborative filtering 2
Error	1.0170	0.9269	0.8812	0.8908	0.8557	0.8262

Table 6: The test error for different prediction methods.

parameters results in a final test error of 0.8262, the best result in the whole problem; a summary of the final test error of the different approaches is shown in Table 6.