IITM-CS5691 : Pattern Recognition and Machine Learning | Release Date: 29th Apr, 2021
New Method Writeup | Due Date: 28th May, 2021

Roll No: EE19B132 | Name: Sagnik Ghosh

- Dear Student, You may have tried or thought of trying different methods for your data contest. Choose one of the methods that was not taught in class, and submit a writeup of this "new method" in the template provided below. This is an individual submission - i.e., while you would've done your kaggle submission as a team of two members or while you may've discussed this method with your teammate, **you will have to write about the new method in your own words independently and submit it individually**.
- **Template:** Fill in whatever fields are applicable for your algorithm (overall 1-2 page writeup; since some fields may not be applicable for certain methods, we haven't shown points below).

1. (points) [Name of Method, and its ML Problem and Paradigm: problem could be regression/classification/clustering/etc., and paradigm could be supervised/unsupervised/..., generative/discriminative/direct, linear/non-linear models, etc.)]:

   Solution: Name of Method: Matrix Factorization

   ML Problem: Dimensionality Reduction (Matrix Decomposition)

   ML Paradigm: Unsupervised

2. (points) [Brief introduction/motivation: One paragraph to describe briefly the new method (its name, what it does, its main application, etc.)]

   Solution: The name of the new method used is Matrix Factorization or Latent Matrix Factorization. It would be easier to understand the method in the context of the Data Contest at hand and it can be easily generalised for other applications. We are given a **train.csv** file with each row corresponding to a particular customer rating a particular song with some score.

   Motivation: Our initial approach was to feed both this data naively into a CatBoost model by treating the 'customer_id' column as a categorical feature. (CatBoost handles categorical features as is.) This worked as well as you would expect as there are around 14000+ customers in all and it gave us a mediocre score on the Public Test Set. This led us to search for methods which would deal with the problem related to the 'customer_id' column more efficiently. More specifically, we wanted quantify the traits of each customer. We finally found a method which exactly suited our needs and it is a very popular method used in many recommender systems.

What It Does: We construct a matrix with each row corresponding to a particular customer and each column corresponding a particular song, and each element in the matrix is how much a particular customer rated a particular song. Naturally, this matrix will be very sparse because each customer has rated only a small subset of all the songs and hence most values in this matrix would be missing values. What Matrix Factorization method does is to generate feature vectors for each customer and for each song such that we need to take the dot product of the feature vector of the customer and the feature vector of the song to obtain the rating that that customer gave that particular song. (This dot product is also the element in the matrix corresponding to that particular customer's rating for that song.)

The **underlying assumption** that we are making is that **if** the matrix were to be completely filled (all the missing values filled), it would inherently be a low rank matrix and hence we can decompose it into lower dimensional matrices.

3. (points) [Closely related method seen in class, and relation of your selected new method to method you eventually used for the data contest]:

Solution: The method seen in class closely resembled by Matrix Factorisation is Principal Component Analysis in terms of the final output. Given a matrix of N D-Dimensional feature vectors, PCA returns N dimensionally reduced feature vectors. Whereas, in the case of Matrix Factorisation, if we have a N*D dimensional matrix, we get two matrices of shapes N*K and K*D, where K can be chosen. It can be related to PCA as follows. N*K dimensional matrix can be interpreted as the **dimensionality reduced** representation if we take the rows of the original N*D matrix as data points and the K*D matrix as the dimensionality reduced representation if we take the columns of the original N*D as the data points.

Note that Matrix Factorisation and Principal Component Analysis are only related because of the similarity in outputs but there are very apparent distinctions in the process of training as we will see in later sections.

Matrix Factorization is also similar to Linear Least Squares regression in the sense that the objective function to be optimized is the very similar. In Linear Regression the objective is to find the optimal w to minimize $\sum_{i=1}^{N}(y_i - w^T x_i)^2$ whereas in Matrix Factorization the objective is to find the optimal p_j and q_k to minimize $\sum_{i=1}^{N}(y_i - p_j^T q_k)^2$ where p_j and q_k are feature vectors corresponding to particular customers and particular songs respectively.

In the context of the Data Contest, we implemented Matrix Factorization as is without any many modifications and we used the generated features and the predicted scores as input features to the CatBoost model along with other input features provided in songs.csv and other files.

4. (points) [Training Input and Output: (e.g., $\{x_i, y_i\}_{\{i=1\ldots N\}}, x_i \in R^d, y_i \in \{-1, +1\}$, etc.]:

Solution: Training Input: $\{x_{1i}, x_{2i}, y_i\}_{\{i=1...N\}}$, $x_{1i} \in \{C_1, C_2, ..., C_J\}$, $x_{2i} \in \{C_1, C_2, ..., C_K\}$, $y_i \in R$

Note that this is exactly the format of 'train.csv' file provided to us where x_{1i} corresponds to the customers and x_{2i} corresponds to the songs.

Training Output: $\{p_i\}_{\{i=1...J\}}$ and $\{q_j\}_{\{j=1...K\}}$, $p_i \in R^M$, $q_j \in R^M$.

p_i's are the M-Dimensional feature vectors corresponding to each customer and q_j's are the M-Dimensional feature vectors corresponding to each song as described in previous sections.

5. (points) [Training Objective function (e.g., loss function that is to be optimized) or probabilistic model (over which MLE or Bayesian inference done):]

Solution: The loss function that is to be optimized is as follows:

$$L = \sum_{i=1}^{N} (y_i - p_j^T q_k)^2 + \lambda(\|p_j\|^2 + \|q_k\|^2) \text{ such that } j = x_{1i} \text{ and } k = x_{2i} \ \forall i$$

In simpler words, p_j and q_k are the customer and song feature vectors corresponding to the i^{th} training sample.

Here, λ is the regularisation parameter.

Please refer to the previous sections for the notation used in the above expression.

6. (points) [Training Algorithm: Brief description of key aspects of the algorithm]

Solution:

- We are given a J*K matrix in which each row corresponds to a particular customer and each column corresponds to a particular song. The desired outputs are p_j's corresponding to each customer and q_k's corresponding to each song.
- As mentioned before, the loss function to be optimized is:

  $$L = \sum_{i=1}^{N} (y_i - p_j^T q_k)^2 \text{ such that } j = x_{1i} \text{ and } k = x_{2i} \ \forall i$$

  We now need to optimize this with respect to both p_j's and q_k's. This can be done using various optimization algorithms but we will choose **Stochastic Gradient Descent** because it gives us an implementation advantage.

- The gradient of the loss function with respect to some p_j is $-2\sum_{i=1}^{N} q_k(y_i - p_j^T q_k) + 2p_j$. Similarly, the gradient of the loss function with respect to some q_k is $-2\sum_{i=1}^{N} p_j(y_i - p_j^T q_k) + 2q_k$. The above expressions can be used to make the optimization updates.
- **Stochastic Gradient Descent:** We have the 'train.csv' file in which each row $\{x_{1i}, x_{2i}\}$ corresponds to a particular customer and a particular song and that in turn corresponds to a particular p_j and a particular q_k as defined before.

  Instead of updating each p_j and each q_k using all the data points at once, we can iterate over all N datapoints and update only the p_j and q_k corresponding that datapoint alone. This means that we would make N updates in all for after traversing the whole dataset once.

  This gives us an **implementation advantage** because we do not have to explicitly construct the J*K matrix which has all the ratings.
- **Update Step:** Let us consider the updates for a single datapoint in 'train.csv'.

$$p_j \rightarrow p_j + \eta[q_k(y_i - p_j^T q_k) - \lambda p_j]$$
$$q_k \rightarrow q_k + \eta[p_j(y_i - p_j^T q_k) - \lambda q_k]$$

  where p_j and q_k are the customer and song feature vectors corresponding to the i^{th} datapoint and λ is a hyperparameter which controls regularisation.

  The above two updates are performed for every datapoint and this algorithm can be run on the whole dataset for as many iterations as necessary till convergence.

  After a fair bit of experimentation, we used the following parameters for the final training: $\eta = 0.01, \lambda = 0.05, M = 100$ where M is the number of dimensions in the generated feature vectors.

7. (points) [Testing Input and Output: (e.g., $x \in R^d$, $y \in \{-1, +1\}$)]

Solution: Testing Input: $\{x_{1i}, x_{2i}\}_{\{i=1...N\}}$ where $x_{1i} \in \{C_1, C_2, ..., C_J\}$, $x_{2i} \in \{C_1, C_2, ..., C_K\}$.
Testing Output: $\{y_i\}$ where $y_i \in R$

8. (points) [Testing Algorithm: Brief description of key aspects of the algorithm]

Solution: We already saw that we got feature vectors p_j's and q_k's corresponding to each customer and each song respectively. Given a test input $\{x_{1i}, x_{2i}\}$ we have the corresponding p_j and q_k. Taking their dot product will give us the predicted rating $y_i \in R$.

Note that these predicted ratings are then fed into the final CatBoost model as input features.

9. (points) [Critique of the method: (1-2 paragraphs discussing its strengths and weaknesses in your own words)]

Solution: Strengths:

- Matrix Factorization generates feature vectors characterizing each customer and each song, not only can they be used as features in the final model, they can be used to make predictions directly by just taking the dot product which in turn can be used as a feature in the final model.
- Another strength of Matrix Factorisation is that it is fairly straight-forward to implement and when implemented with Stochastic Gradient Descent, there is no need to even construct the rating matrix explicitly.
- Matrix Factorization model complexity can also be tuned using regularisation which can control overfitting.

Weaknesses:

- Matrix Factorization is a linear model and hence it cannot capture non-linear dependencies of the rating on customer-preferences and song features.
- Let us explain the first weakness in the context of the Data Contest. When we first tried implementing it, we trained the Matrix Factorization model on 'train.csv' and then we used the same 'train.csv' to train the final CatBoost model. We noticed that the final Boosting model overfit a lot. We reasoned that this was because the final scores to be predicted are already embedded in the train set because of Matrix Factorization. We rectified this by splitting the original 'train.csv' into train-1 and train-2. In this case, we used train-1 to generate customer and song feature vectors using Matrix Factorisation and train-2 for the CatBoost model. But this came at a cost, we could not use all the training data on the CatBoost model and we had to use a significant portion of it on Matrix Factorization alone.
- Although the algorithm generates both customer and song feature vectors, there is no natural way to incorporate the already given song features in the training loop. This meant that we had to further use a CatBoost model to judiciously use the information given to us in form of song features in 'songs.csv' file.

- There are not many ready-to-use implementations of Matrix Factorization in any of the popular Python Libraries (among the ones we were allowed to use in the Data Contest). While Sklearn NMF is a ready to use implentation of Matrix Factorization, it is not suitable for the case of a Recommender System because it cannot deal with missing values. Hence, we had to implement it from scratch. While this was challenging and interesting, it was hard to optimize to the standards of a popular Python package. This is not a weakness of the Matrix Factorization method per se but I think it is noteworthy nonetheless.