COMS W4721 Spring 2020 Homework 4 Programming Assignment: Factorization Machines

Instruction

Please submit your Python notebook on Gradescope for the assignment labeled "Homework 4 Programming Part".

This write-up contains all of your supporting materials which include plots, source code, and proofs for each of the parts in the assignment. Submit your assignment on Gradescope by clearly marking the pages for each part. On the first page of your write-up, please typeset: (1) your name and your UNI; (2) all of your collaborators whom you discussed the assignment with; (3) the parts of the assignment you had collaborated on. The solutions to the problems need to start from the second page. Please write up the solutions by yourself. The academic rules of conduct is found in the course syllabus.

Suggestions

If necessary, please define notations and explain reasoning behind the solutions as concisely as possible. Solutions without explanations when needed may receive no credit. Points can be deducted for solutions with unnecessarily long explanations for lack of clarity. Source code comment can be useful for explaining the logic behind your solutions. Please start early!

Problem 1 (30 points)

This assignment will be graded out of 20 points but you will have a chance to earn 10 extra homework points if you complete everything in entirety.

In this assignment, you will work on implementing the factorization machine for recommendation problems. You will implement factorization machine for both classification and regression.

S. Rendle. "Factorization machines". In: *2010 IEEE International Conference on Data Mining*. IEEE. 2010, pp. 995–1000.

We will be re-using the dataset from the previous homework: the Criteo data (https://www.kaggle.com/c/avazu-ctr-prediction) and using another dataset: the MovieLens 100K Dataset (https://grouplens.org/datasets/movielens/). Both of these datasets are included in your homework package.

Throughout the notebook you'll see TODO tags in the comments. This is where you should insert your own code to make the functions work! If you get stuck, we encourage you to come to office hours. You can also try to look at APIs and documentation online to try to get a sense how certain methods work. If you take inspiration from any source online other than official documentation, please be sure to cite the resource! Good luck!

Small warning: This homework may be time-consuming for those who are not strong in Python. If you feel you may spend too much time on this homework, complete as much as you can, as there are parts that are not intensive. Please budget your time carefully since we are near the end of the semester.

You will also see a new decorator for enabling Numba which is a just-in-time compiler for Python which can optimize numpy-based code. Unlike the last homework, we will not be using dense **numpy.array** to store the feature data X. Instead, it will be stored using **scipy.sparse.csr_matrix**. It is advised that the prototypes for each of the functions you implement are kept intact.

Background: Factorization Machine (FM) type algorithms combines the best of both worlds in linear regression and matrix factorization. It can model interactions between features using factorized parameters. FM models work considerably better on extremely sparse data compared to kernel methods and neural network.

Recall that in linear regression, we have the following model for input data in $\mathbb{R}^d$:

$$\hat{f}(\mathbf{x}) = w_0 + \mathbf{w}^T\mathbf{x}$$

where w_0 is the bias term and $\mathbf{w} \in \mathbb{R}^d$ is the weight vector. This can be computed in $O(d)$ time but it is unable to handle features interactions.

One way to cpature feature interaction is to utilize basis expansion. The resulting model is shown below:

$$\hat{f}(\mathbf{x}) = w_0 + \mathbf{w}^T\mathbf{x} + \sum_{j=1}^{d}\sum_{j'=j+1}^{d} w_j w_{j'} x_j x_{j'}$$

This model now has $O(d^2)$ complexity for prediction. In addition, after one-hot-encoding of sparse data, we have many columns with zero and have the same problem of not having enough data.

FM uses a latent factor $\mathbf{v}_j \in \mathbb{R}^k$ (where k is hyperparameter we can tune) for each feature for $1 \leq j \leq d$ and the interaction between the feature j and the feature j' is modeled between the dot product $\mathbf{v}_j^T \mathbf{v}_{j'}$. The resulting model is the following:

$$\hat{f}(\mathbf{x}) = w_0 + \mathbf{w}^T \mathbf{x} + \sum_{j=1}^{d} \sum_{j'=j+1}^{d} \mathbf{v}_j^T \mathbf{v}_{j'} x_j x_{j'}$$

Naively evaluating the expression above results in $O(kd^2)$. However, we can rewrite the expression above into a form that is more efficient to evaluate:

$$\hat{f}(\mathbf{x}) = w_0 + \mathbf{w}^T \mathbf{x} + \frac{1}{2} \sum_{f=1}^{k} \left(\left(v_f^T \mathbf{x} \right)^2 - \sum_{j=1}^{d} v_{f,j}^2 x_j^2 \right)$$

This new expression can be evaluated in $O(kd)$ time (more efficient!) Collectively note that we have the following parameters in FM models: the bias term $w_0 \in \mathbb{R}$, the weight term $\mathbf{w} \in \mathbb{R}^d$, and the factor matrix $\mathbf{V} = \begin{bmatrix} \leftarrow & \mathbf{v}_1^T & \rightarrow \\ & \vdots & \\ \leftarrow & \mathbf{v}_k^T & \rightarrow \end{bmatrix} \in \mathbb{R}^{k \times d}$

Now we need to discuss the loss functions which are used to optimize the parameters $\mathbf{\Theta} = \{w_0, \mathbf{w}, \mathbf{V}\}$. We can discuss the two loss functions, each for regression and classification. First, the optimization problem:

$$\underset{w_0, \mathbf{w} \in \mathbb{R}^d, \mathbf{V} \in \mathbb{R}^{k \times d}}{\operatorname{argmin}} \underbrace{\frac{1}{N} \sum_{i=1}^{N} \ell(y^{(i)}, \hat{f}(\mathbf{x}^{(i)}))}_{training\ error} + \underbrace{\frac{\lambda}{2} \left(\mathbf{w}^T \mathbf{w} + ||\mathbf{V}||_F^2 \right)}_{regularization}$$

and then the loss functions:

$$\ell_{sq}(y, \hat{f}(\mathbf{x})) = (y - \hat{f}(\mathbf{x}))^2$$
$$\ell_{logistic}(y, \hat{f}(\mathbf{x})) = \ln(1 + \exp(-y \cdot \hat{f}(\mathbf{x})))$$

We will learn the parameters by stochastic gradient descent. This requires us to derive for each parameter $\theta \in \Theta$, the expression for $\frac{\partial \ell(y^{(i)}, \hat{f}(\mathbf{x}^{(i)}))}{\partial \theta}$. Note that by the application of chain rule, for each $\theta \in \mathbf{\Theta}$:

$$\frac{\partial \ell(y^{(i)}, \hat{f}(\mathbf{x}))}{\partial \theta} = \frac{\partial \ell(y^{(i)}, \hat{f}(\mathbf{x}))}{\partial \hat{f}(\mathbf{x})} \frac{\partial \hat{f}(\mathbf{x})}{\partial \theta}$$

(a) (12 points) **Implementing the Loss Functions and Their Partial Derivatives**
First, we need to derive the partial derivatives of the loss function with respect to $\hat{f}(\mathbf{x})$. What is:

$$\frac{\partial \ell_{sq}(y^{(i)}, \hat{f}(\mathbf{x}))}{\partial \hat{f}(\mathbf{x})} = ?$$

What is:

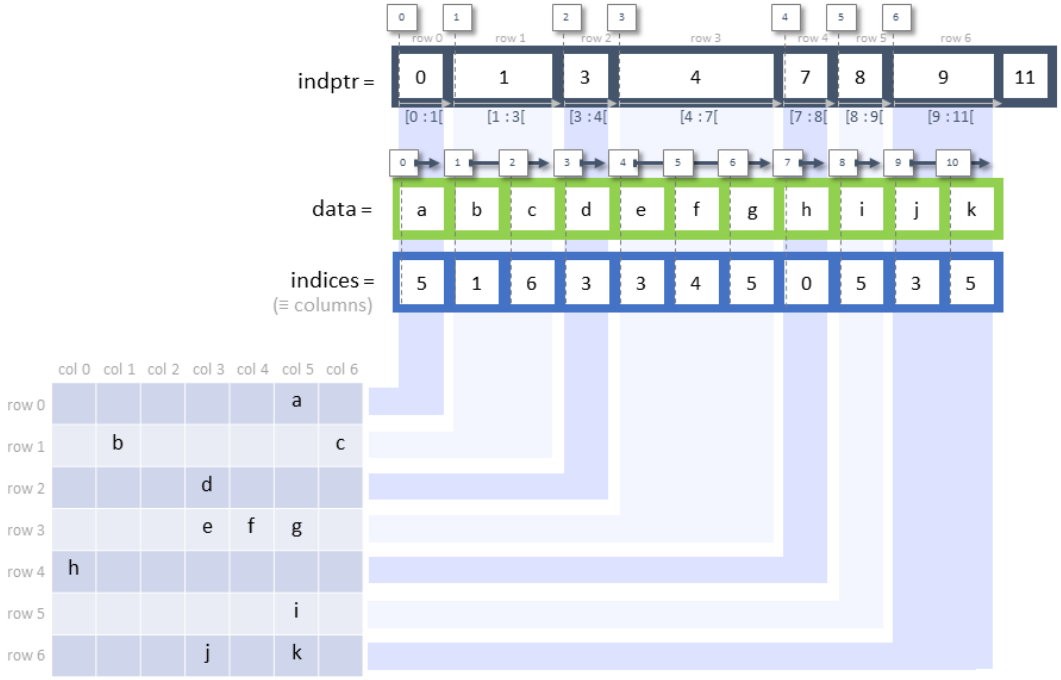
$$\frac{\partial \ell_{logistic}(y^{(i)}, \hat{f}(\mathbf{x}))}{\partial \hat{f}(\mathbf{x})} = ?$$

Given your derivations, now we are ready to implement the classes corresponding to the regression classification. Implement the following functions below (see TODO tags):

1. **squared_loss** which takes the y and the $\hat{f}(\mathbf{x})$ and outputs the squared loss.
2. **partial_derivative_squared_loss** which takes the y and the $\hat{f}(\mathbf{x})$ and outputs the partial derivative of the squared loss with respect to the model output.
3. **log_loss** which takes the y and the $\hat{f}(\mathbf{x})$ and outputs the log loss.
4. **partial_derivative_log_loss** which takes the y and the $\hat{f}(\mathbf{x})$ and outputs the partial derivative of the log loss with respect to the model output.

(b) **Learning the Sparse Data Manipulation**

In this assignment, our data matrix X is extremely sparse (contains many zeros). In order to write efficient code using sparse data, we need to take advantage of how the data is stored under **scipy.sparse.csr_matrix** structure.



i. (6 points) Answer the complexity question on the notebook.

ii. (6 points) Implement the two functions: **compute_dot_between_dense_and_sparse** and **compute_dot_products** functions.

(c) (2 points) **Implementing the Prediction Function**

If you are able to pass the previous checkpoint, then you are ready to implement the following function which outputs

$$\hat{f}(\mathbf{x}) = w_0 + \mathbf{w}^T\mathbf{x} + \frac{1}{2}\sum_{f=1}^{k}\left(\left(v_f^T\mathbf{x}\right)^2 - \sum_{j=1}^{d} v_{f,j}^2 x_j^2\right)$$

You may find the functions implemented in the previous part useful.

(d) (2 points) **Implementing the SGD**

Implement the SGD pass over the data using the algorithm below:

Input: $\{(\mathbf{x}^{(i)}, y^{(i)})\}_{i=1}^N$
$S \leftarrow$ shuffled indices of $[1, \cdots, N]$
$w_0, \mathbf{w}, \mathbf{V}$: the current bias, the weight vector, the factor matrix
η: learning rate, λ: regularization parameter

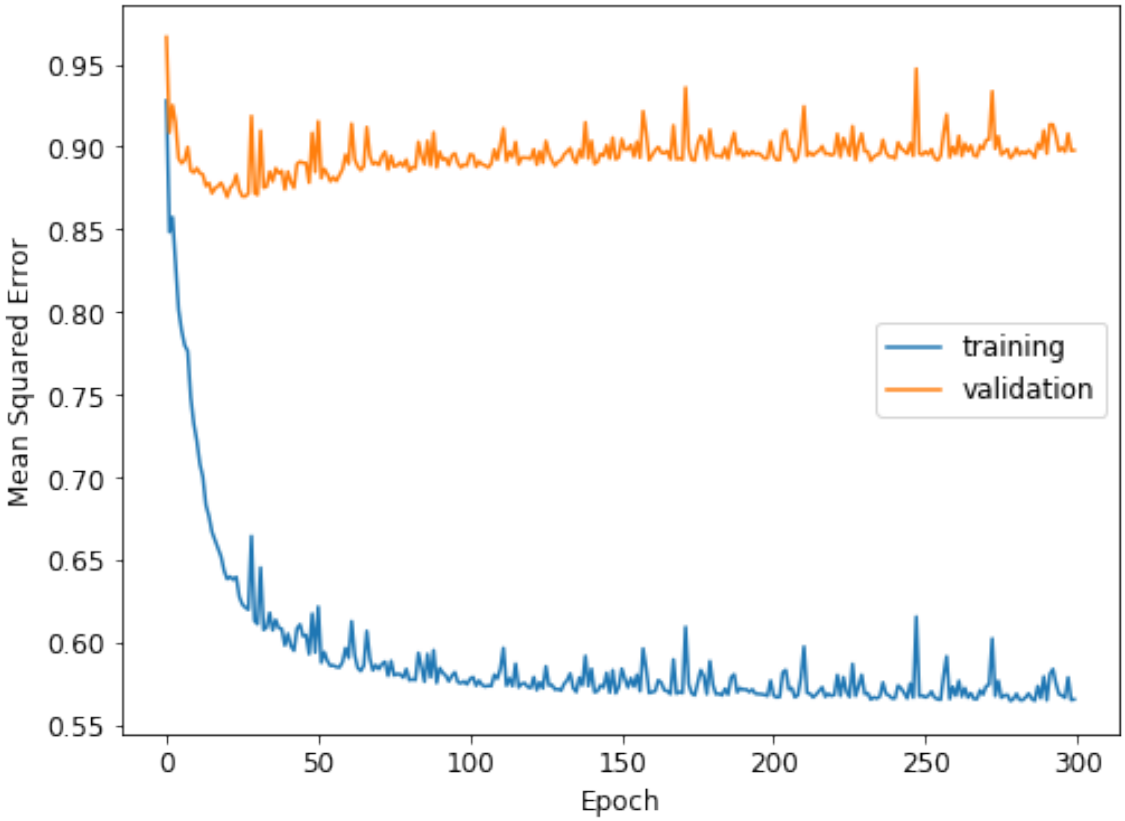
for i_j in S **do**
  $\mathbf{x}^{(i_j), y^{(i_j)}} \leftarrow i_j$-th data in $\{(\mathbf{x}^{(i)}, y^{(i)})\}_{i=1}^N$
  // The bias term is not regularized.
  $w_0 \leftarrow w_0 - \eta \cdot \frac{\partial \ell(y^{(i_j)}, \hat{f}(\mathbf{x}^{(i_j)}))}{\partial w_0}$
  // Update the $\mathbf{w}$ vector only corresponding to the nonzero component of $\mathbf{x}^{(i_j)}$
  for each feature p for which $x_p^{(i_j)} \neq 0$ $(1 \leq p \leq d)$ **do**
    $w_p \leftarrow w_p - \eta \cdot \left(\frac{\partial \ell(y^{(i_j)}, \hat{f}(\mathbf{x}^{(i_j)}))}{\partial w_p} + \lambda \cdot w_p\right)$
    for each $1 \leq f \leq k$ **do**
      $v_{f,p} \leftarrow v_{f,p} - \eta \cdot \left(\frac{\partial \ell(y^{(i_j)}, \hat{f}(\mathbf{x}^{(i_j)}))}{\partial v_{f,p}} + \lambda \cdot v_{f,p}\right)$
    end for
  end for
end for

(e) (2 points) **Finishing Touches**

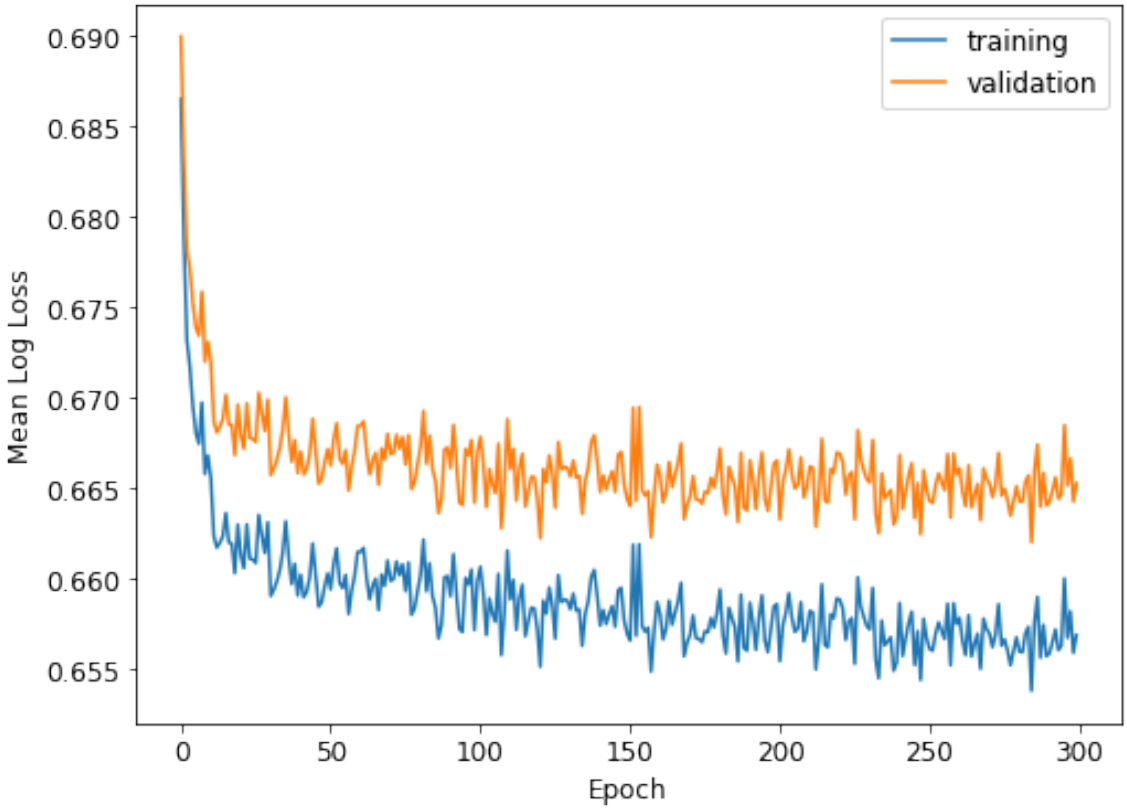
Finish the implementation of the class **fm_model**.

You are now ready to run your code on the Criteo dataset and the MovieLens dataset. For each of these test cases, you may wish to tune the number of factors k and find out what works the best. For classification test cases, see how this compares to the results in the previous assignment using logistic regression. For this part, no writeup is required and doing additional analysis is up to you.

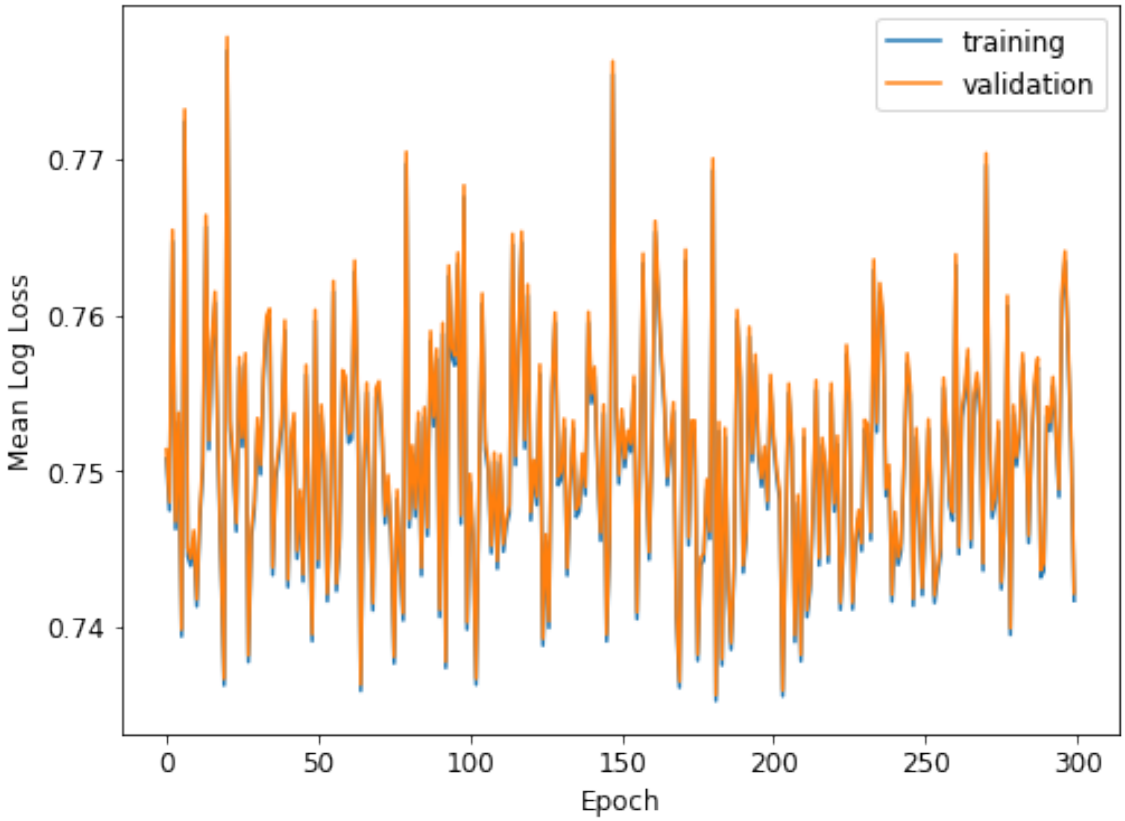
Here are some example plots - again, we are not looking for exact plots. Your results may vary. For the MovieLens data using $k = 5$, $\eta = 0.01$, $\lambda = 0.1$, the FM regression model gives:



For the MovieLens data using $k = 5$, $\eta = 0.01$, $\lambda = 0.1$, the FM classification model gives:



For the Criteo data using $k = 6$, $\eta = 0.01$, $\lambda = 0.1$, the FM classification model gives:



All of the results above are run with only one invocation of SGD method with no warm starts. Keep in mind that SGD methods converge quickly in the few first ierations. Usually, these methods are stopped early and the ending iterates are used as a warm start for another optimization. You may want to try this technique to improve your model.