Networked Life: Project

25/02/2020

Project Goal

- Introduce new algorithms and techniques to provide custom recommendations to users based on what other users liked.

- Learn some basic skills and gain practice experience in machine learning, and enrich your resume.

Main tasks

- Expand on the homework Q4 (i.e., linear regression) by reusing your written code previously, applying it to the Netflix training dataset.
- Develop a classic neural network model to predict a user's rating: restricted Boltzmann machines (RBM).
- Build extensions to the RBM model using some hints we give you and any information you can find online

Submission and Grading

- Project Group: Same with homework group.
- Submission:
  - Code+Report.
  - Deadline: Week 13
- Grading:
  - (25%) Part 1: Linear regression (linearRegression.py, projectLib.py)
  - (25%) Part 2: Basic RBM (rbm.py, mainRBM.py)
  - (25%) Part 3: Extensions of the RBM
  - (25%) Part 4: Report
  - 5 bonus points for the group submitting the best results each week
  - 10 bonus points for the best results overall

Datasets

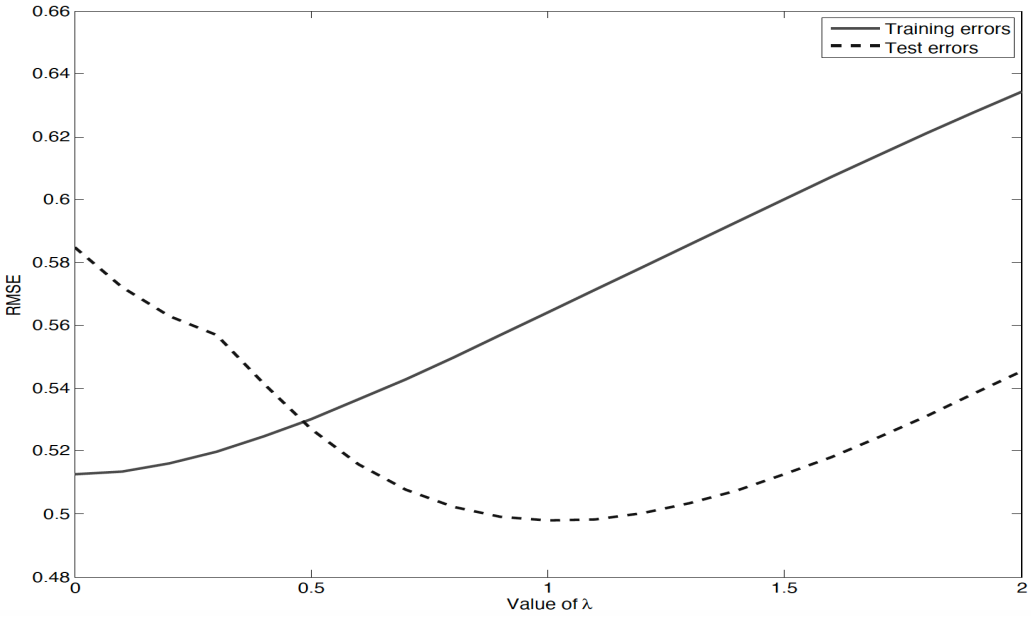
- Two datasets in the folder (studentcode):
  - training.csv: imported to a matrix by getTrainingData function in projectLib file.
  - Validation.csv: imported to a matrix by getValidationData function in projectLib file.
- Both files have three columns: one for the movie ID, one for the user ID and one for the rating ID, in that order.
  - A row of 5,3,4 means that user 3 has given to movie 5 a rating of 4.

Part 1:Linear regression on Netflix dataset

- Question 1.1.
  - Finish the parameter estimation ***param*** function to compute the estimator for the training set, without regularization.
  - Finish the ***predict*** function to compute the predicted rating of every (movie, user) pair of the training set. Then you can compute the RMSE to compare the two.

Part 1: Linear regression on Netflix dataset

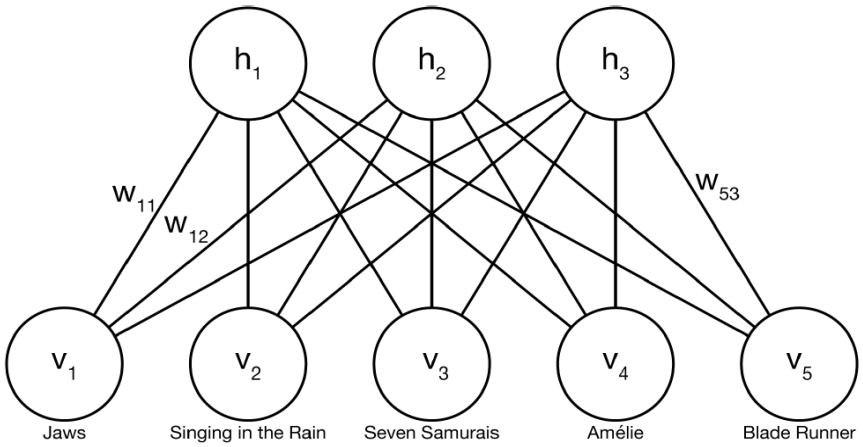
- Question 1.2.
  - Regularize this model so that it does not overfit, by adding a penalty to your biases. You should now complete the ***param_reg*** function.



Part 2: Predicting rating with Restricted Boltzmann Machines (RMB)

- Visible layer V
  - Each node is corresponding to a movie.
  - Receives input (e.g., user's ratings).

- Hidden layer H
  - Each node represents a feature (e.g., whether user likes sci-fi movie or not).



How does the predictor make prediction

- When visible layer receives input, the input is transformed to the hidden layer.
- **Question 2.1:** In the rbm.py file, complete the function sig.

$$\mathbb{P}(h_j = 1|\mathbf{v}) = \sigma\Big(\sum_{i \in V} v_i W_{ij}\Big), \text{ where } \sigma(x) = \frac{1}{1+e^{-x}} \ .$$

❖ Output is the probability of feature j to be active.

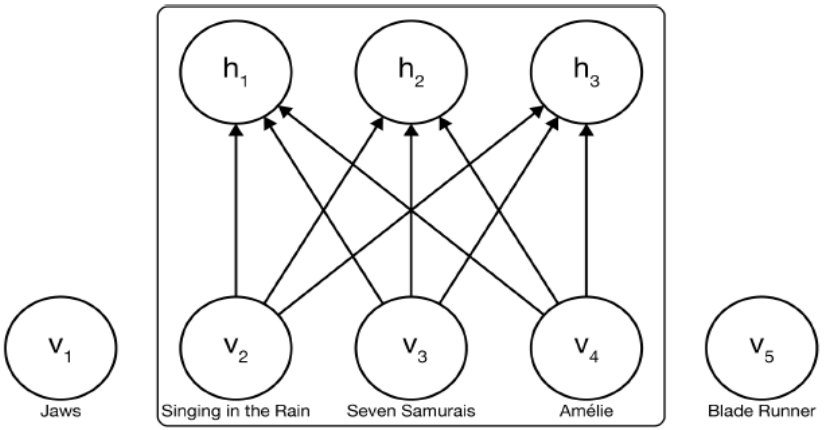


Figure 3: Visible input flows up in the network.

How does the predictor make prediction

- RBM requires the input data to be binary form. To adapt to the requirement, we encode each rating as a binary vector.
  - The rating from a user can be one of the five options. K = {1, 2, 3, 4, 5}.
  - If a user gives a rating 3, that piece of data will be encoded by the vector (0, 0, **1**, 0, 0).

$$\mathbb{P}(h_j = 1|\mathbf{v}) = \sigma\Big(\sum_{k\in K}\sum_{i\in V} v_i^k W_{ij}^k\Big)$$

- Question 2.2.
  - Write a function to propagate the visible input (binary vectors of ratings) to the hidden layer, in ***visibleToHiddenVec***.

How does the predictor make prediction

- After we compute the probability of features (hidden nodes), we then compute the predicted results of the user from hidden nodes to visible nodes.

$$\mathbb{P}(v_i^k = 1|\mathbf{h}) = \sigma\Big(\sum_{j\in J} h_j W_{ij}^k\Big)$$

- Output is not a probability.
- We need to transform it to a distribution.

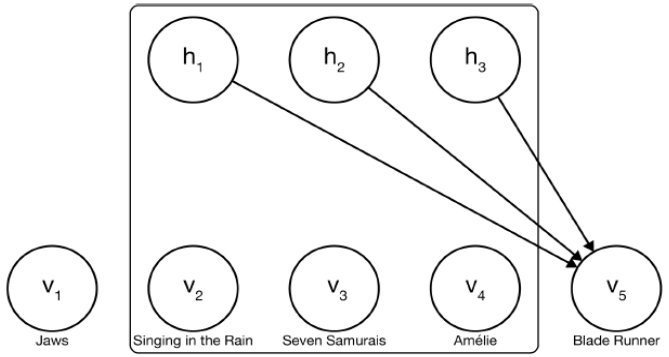


Figure 4: Hidden activations flow down to give a distribution over the ratings.

- **Question 2.3.**
  - Implement the ***hiddenToVisible*** function, taking as input a binary vector of hidden units and the edge weights.

How does the predictor make prediction

- **Question 2.4:** Implement the function **getPredictedDistribution** to get the predicted distribution over the ratings for a movie

How does the predictor make prediction

- Transform the probability distribution of user's rating at different scores to standard rating.
- A user's probabilities for a movie i at 5 different score options (i.e. , 1,2,3,4,5) are (0.05, 0.1, 0.2, 0.15, 0).
  - Predict the rating with the highest score.
    - In the above example, the user's rating to the movie i is 3.
  - Predict the rating with normalized probabilities.

$$p_i^k = \frac{\overline{v}_i^k}{\sum_{l=1}^{5} \overline{v}_i^l} .$$

In the above example, $p_i^1 = \frac{0.05}{0.05 + 0.1 + 0.2 + 0.15 + 0} = 0.1$

Once we have computed the p_i^k, we can simply do $\sum_{k \in K} p_i^k \cdot k$ to get an average rating.

How does the predictor make prediction

- Predict the rating with SoftMax probabilities.

$$p_i^k = \frac{\exp(\overline{v}_i^k)}{\sum_{l=1}^{5} \exp(\overline{v}_i^l)} \ .$$

Once we have computed the p_i^k, we can simply do $\sum_{k \in K} p_i^k \cdot k$ to get an average rating.

- Question 2.5.
  - Implement the three functions ***predictRatingMax,*** ***predictRatingMean*** *and* ***predictRatingExp***, described above.

Find the Weights

- Find the weights that maximize the probability that our model would generate the data it has been trained on, which is called log-likelihood.
- Use a gradient ascent method.
  - Start from a random weight.
  - Update weights by adding the gradient $\nabla W_{i,j}^k$

- Learning:

  – We start with the data $\mathbf{v} = v_i^k$ for a particular user (or a batch of users, see the mini-batch extension).

  – Compute $\mathbb{P}(h_j = 1|\mathbf{v})$ with `visibleToHiddenVec`.

  – Call positive gradient $(PG)_{i,j,k} = \mathbb{P}(h_j = 1|\mathbf{v}) \cdot v_i^k$ (computed by `probProduct`).

Find the Weights

- Unlearning:
  - Sample the states of the hidden units according to $\mathbb{P}(h_j = 1|\mathbf{v})$.
  - Use `hiddenToVisible` to compute *"negative" data* $\overline{\mathbf{v}}$.
  - Essentially repeat the steps of the Learning part, this time with "negative" data, i.e compute $\mathbb{P}(h_j = 1|\overline{\mathbf{v}})$ and call "negative" gradient $(NG)_{i,j,k} = \mathbb{P}(h_j = 1|\overline{\mathbf{v}}) \cdot \overline{v}_i^k$.

- Synthesis: Update only the weights for the movies that the user has rated.

$$W \overset{\sim}{\leftarrow} W + \epsilon \nabla W_{i,j}^k$$

Where:

$$\nabla W_{i,j}^k = (PG)_{i,j,k} - (NG)_{i,j,k}$$

Part 3: Some extensions by online references

- Adaptive learning rates.
- Early stopping.
- Regularization.

Gradient oscillates

Test algorithms

- Apply your model on the test data.
- Test your program on test data (test.csv).
  - When you produce the text file, name it in the following format: <name of the team>+v<version number>.txt.
  - After receiving your file, we will compute the RMSE obtained on the test set and give you its value.

Competition for bonus score

- Submit result once every week from week 9 to week 12.
  - Based on RBM, not Regression.
- At the end of each week, the current best RMSE (over all previous weeks) will be broadcasted to the class, for you to know which is the current goal to beat. Also, each week, the team with the best outstanding RMSE over the test set will get a bonus of 5 points.
- At most 10-bonus score for each group.

Week	Dates
9	Monday, March 23-Sunday, March 29 (11:59 PM)
10	Monday, March 30-Sunday, April 5 (11:59 PM)
11	Monday, March 6-Sunday, April 12 (11:59 PM)
12	Monday, March 13-Sunday, April 19 (11:59 PM)

Submission requirement of final project documents

- The source code of your implementation. You are able to modify the signature of the provided functions as long as you properly document why.
- A project report figuring some details of your implementations, e.g., plots showing the evolution of your RMSE as the training occurs or how the extensions have improved your results.

References:

- [1] G. Louppe. Collaborative filtering: Scalable approaches using restricted Boltzmann machines. PhD thesis, Université de Liège, Belgique, 2010.
- [2] R. Salakhutdinov, A. Mnih, and G. Hinton. Restricted Boltzmann Machines for collaborative filtering. In Proceedings of the 24th international conference on Machine learning, pages 791–798. ACM, 2007.
- [3] Asja Fischer and Christian Igel. Training Restricted Boltzmann Machines: An Introduction. Pattern Recognition 47:25-39, 2014.
- [4] Geoffrey Hinton. A Practical Guide to Training Restricted Boltzmann Machines. Technical Report. https://www.cs.toronto.edu/~hinton/absps/guideTR.pdf.
- [5] Neural networks [5.1] : Restricted Boltzmann machine. https://www.youtube.com/watch?v=p4Vh_zMw-HQ.

Thank you!

Questions?