KDD 2020: Debiasing

Team: INNOVA-TSN/UCM

Position: 17

1. Resume

This report summarizes and explain the solution to the 2020 KDD problem found by the INNOVA-TSN/UCM team, made of Christian Bartolomé, Joaquín Berbiela, José Ignacio Bernaldo, Juan Antonio Carrillo, Rodrigo García, Fernando Sebastián, Alejandro Jaraiz, Daniel Vélez y Jorge Verges.

The final model is a blending of 3 boostings (catboost, lightgbm and xgboost) over 7 models in two different language (SAS and Python):

- Model 1: Model Baseline (ItemKNN) (Python)
- Model 2: Model UserCF (Python)
- Model 3: Model VMSKNN (Python)
- Model 4: Model Dani (SAS) (Artesanal ItemKNN Radius items)
- Model 5: Model Dani2 (SAS) (Artesanal ItemKNN Radius items with another parameters)
- Model 6: Model TXT (SAS) (Using last item predict most similar items using PCA from item_feat only TXT features)
- Model 7: Model Dani3 (SAS) (Artesanal ItemKNN Radius items using PCA from TXT + IMG)

The weights have been optimized to get the best NCDG_full and, after that, to improve NCDG_half over that blending. See

\code\model_boosting\4_FINAL_MODEL_XGBOOST_LIGHTGBM_CATBOOST.py.

Run code

Just run the code

Main.sas

2. Model overview

This section deals with the explanation of every model utilised to determine the variables which feed up our Boosting blending:

- Dani Code Explanation (4 models):

  Given a user, this code executes the next process:

1. First, it keeps items from its last few clicks (determined by the "longitudSecuencia" parameter).
2. Then, it searches these items on history and see which are the next items that are usually clicked after them (determined by the parameter "radioPosterior") and which occurred previously (determined by the parameter "radioAnterior".
3. It adds occurrences of items which where clicked before and after and it recommended according to that order (determined by the parameter support). This order responds to the variable number.
4. After that, this code awards items which were clicked in the same session that the item searched on history (this is what activates to specify the "usaSessionId"=1 parameter). This is to say, it searchs the item and, when it makes the radius, it counts how many times both the searched item and the other items are in the same session occurs. That's what determines the variable "numeroConSession".
5. There is also another similar parameter called "Timeref" which forces the radius to be restricted by time. In other words, the code keeps items clicked before and after the searched item but only if those items belong to a close time environment (determined by the "Timeref" parameter. Note that the effect of this parameter is like the previous one.
6. Finally, given that the sorting is done by "descending numero" and "descending numeroSession", there can be ties (many times these variables are worth 1 or 2), and that's something very important to fix. So, the way to solve those ties is calculating the cosine distance between the searched item and the rest of items which are into his environment (into the specified radius balls) and the codes sorts the items according the closer distance (in absolute value, the lowest value of the cosine distance). This distance is calculated with respect to the Principal Components which were obtained from the items' features. In one case, it is only done with TXT variables and the other with all of them.

In the end, the variable "peso" (weight) is determined like this:

PESO = sum(NUMERO, NUMEROCONSESSION, (2-DISTANCIACOSENO), 0)

After getting this variable, the code uses it to sort for each user_id.

It's remarkable to note that these Dani models play with some different configurations of these 7 parameters, which are listed in the next table:

Parameters\Conf.	First configuration	Second configuration	Others
longitudSecuencia	2	3	
radioAnterior	7	2	
radioPosterior	7	2	
soporte	1	1	
timeref	0.001	0.00001	
variables	ALL	TXT	
usaSessionId	1	1	

Thas last two models are based on cosine distance between consecutive items (using PCA 95, from TXT and TXT + IMG)

These Dani Models are the only one artesanal models designed by our team. The Baseline Model was inspired in the one which was in the forum, the Vector-MultiplicationSKNN Model was taken from the GitHub repository and USERCF is a commonly known model.

Anyway, let's write some lines in order to explain the way they work:

- Baseline Code:

This code, as the one which is in the forum, is basically made of 3 functions: "get_sim_item" obtains a huge and asimetric dictionary with the grade of similarity between items which have been clicked by various users, "recommend" focuses on make a 50 items long recommendation for every test user and "get_predict" just make a final prediction by adding the most popular items over items with negative similarity (but looking for more similar items works better than adding popular items).

It's important to note that the improvement of this code is based on the optimization of all the parameters involved in it, such as the one wich decreases the similarity between items in fact of which was clicked first. As this code only looks at the similarity between users history and skips the similarity beetween item features (embedding), it was one of the points to improve on it, specially to undo posible ties. Another improvement we added was to remove, for each user, already clicked items from his final prediction. Lastly, cosine similarity improves a lot the model (using features PCA).

- Vector Multiplication Session-Based Item KNN (VMSKNN):

This model follows the code available at the GitHub repository. In his parameters, we use k=5000 as the number of neighboring session to calculate the item scores from and sample_size=10000 as the length of a subset of all training sessions to calculate the nearest neighbors from. The rest of parameters were similarity = "cosine", weighting = "quadratic", weighting_score="div", idf_weighting=5 and normalize=True.

Like we did in the Baseline code, we also removed already clicked items for each user final recommendation.

- UserCF Model:

User similiraity (like ItemCF but using users)

3. Tricks to get Top20:

1. Using i**tem features** as a weight to similarity improve and to undo possible ties.
2. **Boosting** from all models improve (recall 500 recommendations).

2. Optimization using a vector to **increase probability over half items** (the way we identified **non-usual items** was calculating which ones were clicked less than median **EACH PHASE**).

3. We tried to increase the **NCDG-Half** by multiplying the most uncommon items probability by a factor **1,75**. Previous that, we used SciPy to optimize a vector = [t0, t1] that changed some items probabilities with the intention of improving the NCDG-Full.

4. Last days, we were focused on calculate weights to improve **NCDG-Half** trying to avoid losing a big amount of **NCDG-Full**.

4. Python dependencies

Here we list every Python dependency we used in our report, aligned to its version:

`Bayesian-optimization==1.1.0`	`NumPy==1.18.4`
`catboost==0.23`	`pandas==1.0.3`
`lightgbm==2.3.1`	`pickleshare==0.7.5`
`xgboost==1.1.1`	`scipy==1.4.1`
`tensorflow==1.0.0`	`sklearn==0.0`

5. Language

- SAS
- Python

6. References

- Forum (Baseline Code)

- Session rec Package https://github.com/rn5l/session-rec

-https://towardsdatascience.com/intro-to-recommender-system-collaborative-filtering-64a238194a26

Contact

If you have any questions, please contact fernando.sebastian@innova-tsn.com