SalesIntelligence°
DIGITREE GROUP

The Session-Based kNN implementation in TensorFlow

Szymon Moliński
Data Scientist
szymon.molinski@salesintelligence.pl
Digitree Group S.A

Research project **E-commerce Shopping Patterns Prediction System** was funded under Priority Axis 1.1 of Smart Growth Operational Programme 2014-2020 co-funded by European Regional Development Fund. Project number: **POIR.01.01.01-00-0632/18**

European Funds
Smart Growth

Republic of Poland

European Union
European Regional Development Fund

Session-based k-NN

Why do we use it?

1. **Simple**: complexity of k-NN algorithm is very low.
2. **Reliable**: performance of Weighted Session-based k-NN is comparable with the state-of-the-art and extensively tuned deep learning models[1,2].
3. **Fast**: with mapped sessions and items we get fast results of prediction.
4. **Benchmarking**: the complex neural networks models are compared to simple k-NN.
5. **Overfitting Protection**: neural networks quickly overfit with small datasets. For some vertical categories it is better to use more generalizing model.
6. **Explainability**: The output may be easily explained.

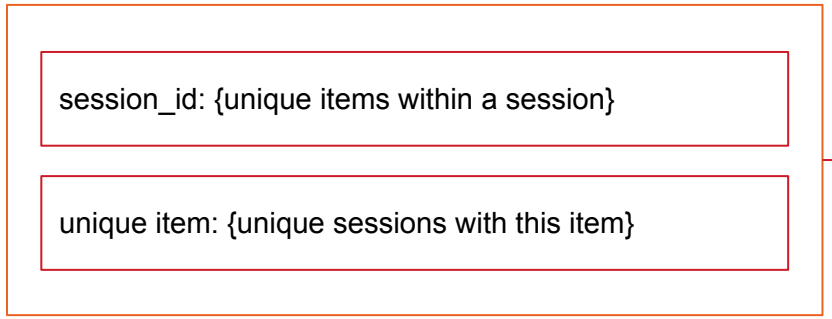
[1] Guo H., Tang R., Ye Y., Liu F., Zhang Y. (2019) A Novel KNN Approach for Session-Based Recommendation. In: Yang Q., Zhou ZH., Gong Z., Zhang ML., Huang SJ. (eds) Advances in Knowledge Discovery and Data Mining. PAKDD 2019. Lecture Notes in Computer Science, vol 11440. Springer, Cham. https://doi.org/10.1007/978-3-030-16145-3_30
[2] Own research

Session-based k-NN

Algorithm Structure: process could be done with Python dict, set and list

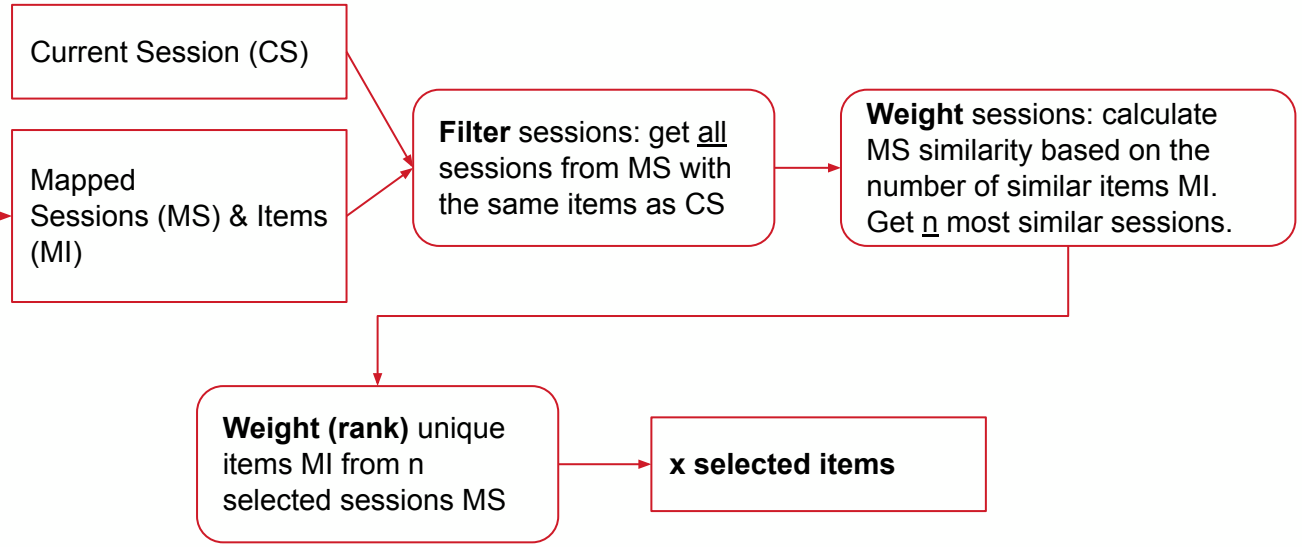
Fit

Build maps of sessions and items. Map structure is (dict: set)



Predict

Calculates distance between a given session items and memorized sessions to return the recommendations (next items)



Session-based k-NN

TensorFlow and TensorFlow Extended implementation: Why do we do it?

1. **Mutualisation**: the core model is served as neural network written in TensorFlow. All benchmarking models should be placed in the same environment.
2. **TFX == TF**: **TensorFlow Extended pipelines work best with models written in TensorFlow and TensorFlow graphs**. However, it is not easy task to present memory-based model with sessions- and items-maps as a TensorFlow structure.
3. **Monitoring**: Model output may be compared with other architectures within the pipeline.

Transform and Trainer blocks from TFX pipeline are the most significant for implementation: they differ from model to model but other blocks are the same; e.g.: data input and model validation is always the same

Session-based k-NN

TensorFlow and TensorFlow Extended implementation

We cannot use simple Python structures within TFX pipeline if we want to run algorithm as a **TensorFlow function**. Other problem is that the input data sources are **TF.records** data type and must be converted into Tensors (not dicts, not lists, not arrays).

Fit

Build Four output Tensors

Transform module returns sessions as a single records with session id and item is

Tensor ST with x encrypted sessions (as **string**)

Ragged Tensor RST with x rows with encrypted items (as **string**) : session ST[i] -> items RST[i]

Tensor IT with x encrypted items (as **string**)

Ragged Tensor RIT with x rows with encrypted sessions (as **string**) : item IT[j] -> sessions RIT[j]

Predict

Trainer module with TFX specific **run_fn** function. To save trained model it is important to set **signature_dict**.

The general logic of prediction is the same as with the Python core structures. Differences are within an implementation in TensorFlow. It requires use of Tensors instead of simple containers.

Implementation Aspects

1) Index matters! Index of (session / item) in (ST / IT) marks their (items / sessions) in Ragged Tensors (RST / RIT).
2) We don't know maximum number of items per session and sessions per item. That's why Ragged Tensors are used. It complicates prediction because TensorFlow is very strict on Tensor's shape tracking. It is hard to speed-up algorithm without any length-related constraints.
3) Even constant values (zero, one) must be implemented as Tensors.
4) We can create TensorArrays with dynamic size to create lists of elements which are expanding through iterations: e.g.: list of closest items or sessions.
5) Due to the fact that k-NN is a memory-based model there is upper limit of the input data size. It is not possible to use this algorithm on very big databases with multiple sessions and large items base.