Alpenglow: Open Source Recommender Framework with Time-aware Learning and Evaluation[*]

Erzsébet Frigó Róbert Pálovics Domokos Kelen Levente Kocsis András A. Benczúr
Institute for Computer Science and Control
Hungarian Academy of Sciences (MTA SZTAKI)
{frigo.erzsebet, rpalovics, kdomokos, kocsis, benczur}@sztaki.hu

ABSTRACT

Alpenglow[1] is a free and open source C++ framework with easy-to-use Python API. *Alpenglow* is capable of training and evaluating industry standard recommendation algorithms including variants of popularity, nearest neighbor, and factorization models.

Traditional recommender algorithms may periodically rebuild their models, but they cannot adjust online to quick changes in trends. Besides batch training and evaluation, *Alpenglow* supports online training of recommendation models capable of adapting to concept drift in non-stationary environments.

1 INTRODUCTION

Available free and open source recommender systems[2] mostly follow the needs of static research data such as the Netflix prize competition with a predefined subset of the data for training and another for evaluation. In a real service, users request one or a few recommendations at a time and get exposed to new information that may change their preferences for their next visit. Furthermore, recommendation applications usually require top-k list recommendations and provide implicit feedback for training recommender models [1].

In a real application, *top item recommendation by online learning* is hence more relevant than batch rating prediction. We target top-k recommendation in highly non-stationary environments with implicit feedback [1, 2, 4]. Our goal is to promptly update the recommender models after each user interaction by online learning.

We present *Alpenglow*, a conjoint batch and online learning recommender framework. When Alpenglow reads a stream of user–item interaction events, first it constructs a recommendation top list for the user. Next, the consumed item is revealed, the relevance of the top list is assessed, and the model is immediately updated. *Alpenglow* works in a single server shared memory multithreaded architecture.

2 ALPENGLOW IMPLEMENTATION

Alpenglow is capable of training various factorization, similarity, recency, and popularity based models including

- temporal popularity and item-to-item models;
- time sensitive variants of nearest neighbor, e.g. with the introduction of a time-decay;
- batch and online matrix factorization (MF), including asymmetric MF, SVD++ and other MF variants;
- time-aware online combination [2] of all these models.

[*]Support from the *EU H2020* grant *Streamline No 688191* and the “Big Data—Momentum” grant of the Hungarian Academy of Sciences.
[1]https://github.com/rpalovics/Alpenglow
[2]https://github.com/grahamjenson/list_of_recommender_systems

Algorithm 1: An *Alpenglow* experiment in Python.

```
import alpenglow
from alpenglow.experiments import BatchAndOnlineExperiment
import pandas

data = pandas.read_csv("/path/to/sample_dataset")
factor_model_experiment = BatchAndOnlineExperiment(
  top_k=100,
  dimension=10,
  online_learning_rate=0.2,
  batch_learning_rate=0.07,
  number_of_iterations=9,
  period_length=608400    # 1 week in sec
)
rankings = factor_model_experiment.run(data, verbose=True)
results = alpenglow.DcgScore(rankings)
```

The framework is composed of a large number of components written in C++ and a thin Python API for combining them into reusable experiments. It is compatible with popular packages such as the Jupyter Notebook, and is able to process data from Pandas data frames. Furthermore, the framework provides a scikit-learn style API as well, for traditional batch training and evaluation.

The Python API is illustrated in Algorithm 1. In the code sample, user–item pairs are read into a data frame. Then we set up an experiment, in which 10-dimensional factor models are periodically batch trained and then continuously updated by online learning. The learning rates, the batch iterations, the batch training periods and possibly negative and past event sample counts and other parameters are passed to the object. When running the experiment, we obtain the list of rankings, which is finally evaluated by online DCG (see Section 3). Both rankings and online DCG scores are stored in data frames. Ongoing work includes further modularizing the components of mixed batch and online models.

Another advantage of the Python API is that it gives access to the modular construction of the C++ core implementation. Users may construct recommenders from various *models*, *objectives*, *updaters*, *learners* and run their experiments in different *experimental settings*. After a new record is evaluated, the training process is orchestrated by *learners*, which execute batch, online or sampling learning strategies. *Updaters* train the *models* by altering their states, which in turn use the trained states to provide predictions to be evaluated. The *updaters* are often defined using modular *objectives*. The framework also provides a number of preconfigured experiments ready to be run on a given data set. For evaluation, both rating and ranking based measures are available in an online evaluation framework

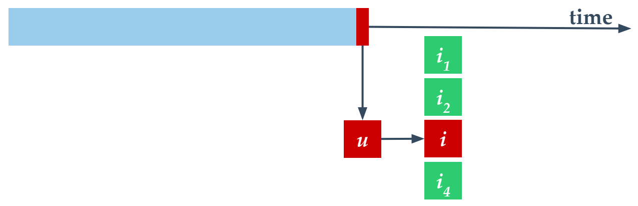

Figure 1: Temporal evaluation and learning in Alpenglow.

including MSE, DCG, recall, or precision, which are all continuously updated.

3 TEMPORAL EVALUATION

In an online setting as in Fig. 1, whenever a new user-item interaction is observed, we assume that the user becomes active and requests a recommendation. Hence for every single unique event, *Alpenglow* executes the following steps:

(1) generates top-k recommendation for the active user,
(2) evaluates the list against the single relevant item that the user interacted with,
(3) updates its model on the revealed user-item interaction.

We use DCG computed individually for each event and averaged in time as an appropriate measure for real-time recommender evaluation [2]. If i is the next consumed item by the user, the *online* DCG@K is defined as the following function of the rank of i returned by the recommender system,

$$\text{DCG@K}(i) = \begin{cases} 0 & \text{if } \operatorname{rank}(i) > K; \\ \dfrac{1}{\log_2(\operatorname{rank}(i)+1)} & \text{otherwise.} \end{cases}$$

4 ONLINE LEARNING

Online algorithms read the data in temporal order and may process each record only once. For example, the online variant of a matrix factorization model with gradient descent updates the corresponding user and item latent vectors after each observed interaction. Compared to batch recommenders, online models may be advantageous, as they

- can adopt to temporal effects, hence may handle concept drift,
- can often be trained significantly faster than their batch variant.

Next we present an experiment of the simplest batch and online trained *Alpenglow* models. We use data carefully distilled from the (*user, artist, timestamp*) tuples crawled from Last.fm [3].

- We deleted artists appearing less than 10 times.
- For each user-artist pair, we kept only the first occurrence and deleted all others so that we only recommend new artists.
- We discarded playlist effects: to avoid learning automatically generated sequences of items, we kept only those user-item interactions that start a user session.

The final data contains 1,500-2,000 events per day for over one year.

Figure 2 shows the best performing 10-dimensional factor models by batch and online training. All of them are trained with stochastic gradient descent (SGD) for mean squared error (MSE) on implicit data. The batch model is only retrained weekly, with several iterations of lower learning rate $lr = 0.07$. In contrast, the online model uses a single iteration and processes each record only once,

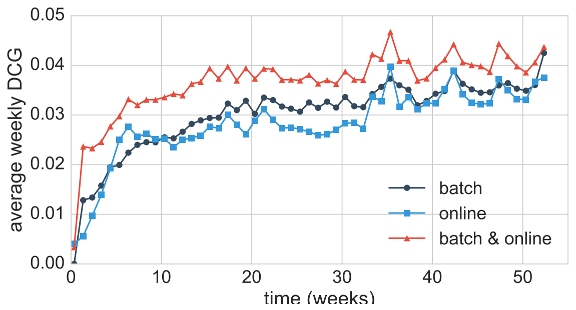

Figure 2: Performance of the batch, online and batch & online methods over the Last.fm data set. DCG scores are computed individually for each unique user-artist interaction and then averaged weekly.

immediately after it is observed, and applies higher learning rate $lr = 0.2$. We evaluated top-k recommendation for each single interaction by using DCG and then computed weekly averages. As seen in Figure 2, the performance of the online model is close to the batch model, despite the fact that it cannot iterate in the data.

Finally, we describe the batch&online model, which is implemented in Algorithm 1. In this model, we periodically re-train the model at the end of each week by batch SGD. Afterwards, we train the model during the next week via lightweight online updates. Batch&online results in significant improvement over both individual models.

5 CONCLUSIONS AND FUTURE WORK

We presented *Alpenglow*, a C++ recommender framework with Python API. The current version of the code is able to produce recommender models that can adapt to non-stationary effects in real recommendation scenarios. It includes batch and online variants of several standard recommendation models.

The goal of *Alpenglow* is twofold. First, it produces temporal recommendation models that can be combined to batch models to achieve significant performance gains. Second, the framework can simulate the streaming recommendation scenario offline. Hence it supports the selection and hyperparameter tuning of models trained on streaming data.

In our future work, we intend to connect Alpenglow with other popular Python packages. Furthermore, our plan is to advance the architecture of the framework towards distributed recommender APIs (Apache Flink, Apache Spark) and data streams, thus making it possible to rapidly prototype and evaluate online learning recommenders.

REFERENCES

[1] X. Amatriain and J. Basilico. Past, present, and future of recommender systems: An industry perspective. In *Proceedings of the 10th ACM RecSys*, 2016.
[2] R. Pálovics, A. A. Benczúr, L. Kocsis, T. Kiss, and E. Frigó. Exploiting temporal influence in online recommendation. In *Proceedings of the 8th ACM RecSys*, 2014.
[3] R. Turrin, M. Quadrana, A. Condorelli, R. Pagano, and P. Cremonesi. 30music listening and playlists dataset. In *RecSys Posters*, 2015.
[4] J. Vinagre, A. M. Jorge, and J. Gama. Evaluation of recommender systems in streaming environments. In *Workshop on Recommender Systems Evaluation, October 10, 2014, Silicon Valley, United States*, 2014.