Introduction

In this exercise you will be implementing a “state of the art” recommender system. You will start by building an implicit matrix factorization algorithm using the Alternating Least Squares (ALS) algorithm and finish training an efficient Gradient Boosting Decision Tree ensemble using XGBoost. All of this will be done in Python using a Jupyter notebook. You don’t need to be an expert in Python to follow the tutorial though.

But, first things first, we need to make sure your system is up to date with the necessary requirements.

Requirements and installation instructions

As part of our class on Industrial Recsys, we will be doing a short Python exercise. In order to follow and do it on your own, you will need to have Python (you can probably work with anything > 2.7, but I’d recommend going to Python 3 and a version => 3.5) and a few packages installed:

- Numpy and Scipy
- Jupyter
- Pandas
- Sklearn

All but the last one are available as part of the Anaconda distribution. This will be the easiest way for you to get ready so I’d recommend going that route. Otherwise, follow the individual links above.

The only packages we will be using and are not available in Anaconda are **implicit** and **xgboost**. You will need to download and install directly following the instructions on the webpage:

- Implicit
- Xgboost

Note 1: You need to have a reasonably up-to-date installation of the above packages for things to work out-of-the-box. If at any point in time you get stuck with some Python code that should work, but doesn’t, ask yourself whether you have updated your dependencies.

Exercise

Note 2: At any point during the exercise, ask Xavier for help. You can even get a working iPython notebook with all the different steps in it if you need it or can't get things to work by the end of the session.

1. Create a blank Jupyter notebook
2. Execute the tutorial by Jesse Steinweg-Woods step by step in your notebook
    a. **Note 3**: For the sake of time, please skip section on "Implementing ALS for Implicit Feedback" and go straight into "Speeding up ALS", you can always get back to this interesting section at the end if you have time
    b. Note 4: The tutorial by Jesse is a bit outdated and uses a deprecated interface for accessing *implicit*. Read the current instructions/examples here. Ask for help or the required lines of Python code from the instructor.
3. Using original dataset, combine ALS prediction with country and unit price to create a new dataset with those features encoded as numbers
4. Sample random negatives and add to the dataset
5. Train an xgboost ensemble adding the dataset that includes ALS plus other features. Optimize for ROC AUC metric
6. Using GridSearchCV from sklearn perform a grid search to optimize the following hyper-parameters of the tree ensemble: max-depth, min_child_weight. You should limit your search to 3 different values for each and 5 fold cross-validation.
    a. Note 5: Depending on your laptop the process of training xgboost might still take a few minutes. You might want to limit the number of hyper-parameters to explore and even reduce the folds to use in cross-validation. Also, you should try to set the number of jobs ('n_jobs') in the cross validation to at least the number of cores in your machine and set verbose=2 in GridSearchCV to monitor progress. While parallelizing jobs should in theory speed up things, many people, including myself, have found that their machine freezes with n_jobs>1.
7. What is the "test AUC" for the optimal model?
8. Bonus
    a. Think of other features that you could add to the ensemble
    b. Optimize some of the other hyper-parameter from XGBoost
    c. See how xgboost and some of the hyperparameters behave to other metrics besides AUC
    d. Build a Python restful service that uses the trained model to serve recommendations for a user

Final working Jupyter notebook

You can download from here.