Personalize Expedia Hotel Searches

1. Problem Background

1.1 Task Description

Expedia is the world's largest online travel agency (OTA) . Generally speaking, our task is to learning to rank the hotels according to search query (srch_id) to maximize purchase. In the dataset, an instance should corresponds to an impression of a hotel(prop_id) for a query(srch_id) and possible features in one instance can be summarized into the following categories:

- Hotel characteristics
- Location attractiveness of hotels
- User's aggregate purchase history
- Competitive OTA information

Basically, we are provided two data sets: train.csv, test.csv. Corresponding to the above categories, the following table of features is given. To note that in test.csv, four features are absent: position(ranking position of the hotel for the query), click_bool(whether the user clicked the hotel), gross_bookings_usd(total expense after purchasing), booking_bool(whether the user booked/purchased the hotel).

1.2 Evaluation Metrics.

Since we're more familiar with AUC(Global AUC, not the average per query AUC) and it's much easier to calculate the AUC of our predictive ranking offline by ourselves, **most of our experiments turned to AUC as our evaluation metric.** Hotels for each user query are assigned relevance grades as follows:

5 - The user purchased a room at this hotel

1 - The user clicked through to see more information on this hotel.

0 - The user neither clicked on this hotel nor purchased a room at this hotel

2. Problem Analysis

2.1 Our Approaches

We tried Binary Classification. Essentially, it's not a classification. Before explaining that, since the labels in the data set are Purchase(5), Click(1) and None(0), we normalized them into 1, 0.2, 0 respectively to better fit the need of binary classification, which would not miss any information. To continue the explanation, we actually only cared about the probability of the hotels being purchased by the user, that is, P(y=1) or P(y=0) = 1 – P(y=1), we tried different models to fit this probability. Then as long as we get the probability, we don't have to convert it the class labels at all, since what we need is only the ranking list of hotels for every query in the test dataset. **We tried Regression**, which is intuitive and can be used for ranking. In this approach, we directly consider the labels as numerical output, a signal of users' preference on the hotels.

2.3 Model: SVDFeature

SVDFeature is an extension of Biased Matrix Factorization. Following is the model itself using formulas: $\hat{y} = \sum_i b_i * x_i + \left(\sum_j P_j * x_j\right)^T \left(\sum_k Q_k * x_k\right)$

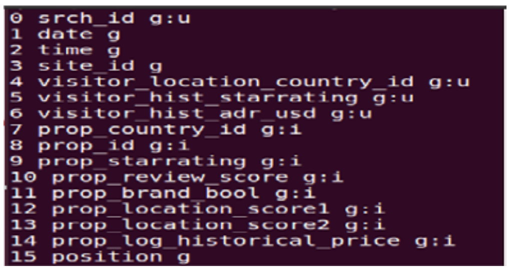
In it, y-hat is the first-phase output. xi are the global features. By global, we mean the feature has no obvious association with the user/query or the item/hotel. bi are the weights we need to learn, corresponding called global biases. Xj are the user/query features, which include features obviously associated with the user/query aspect. Pj are factors (vectors of weights, the length is also called dimension) we need to learn, which means P actually is an m-by-d matrix, where m is the number of user/query features and d is the dimension. The same with xk and Qk, xk are item/hotel relevant features and Qk correspond to the associated factors. Q is also an n-by-d matrix, where n is the number of item/hotel features. Y-hat can be directly used in regression models.

$\hat{y}' = \frac{1}{1 + e^{-\hat{y}}}$ Y-hat' is the second-phase output or squeezed output with sigmoid function.

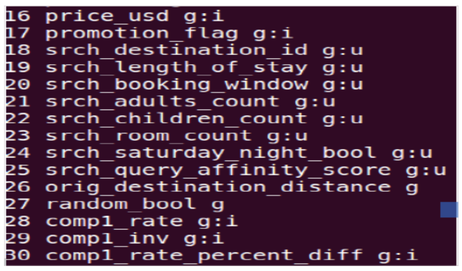
This output can be used in logistic regression, binary classification or fitting the probability of y=1. **Actually, without the factor terms, SVDFeature becomes Logistic Regression.** Another transformation is the inner product in the factor terms, inspired by **Kernel-PCA and SVM**, we found we could change the inner-product(**linear-kernel-function)** to other kernels. We tried **Radius Basis Function kernel and Polynomial kernel**. We do use different loss functions to simulate SVM, like **Hinge Loss.**

2.5 Feature Extraction

As mentioned above in SVDFeature model, **we classify the features into three kinds,** as follows. Global features are tagged with g. User/query features are tagged with u. Item/hotel features are tagged with i.



```
0 srch_id g:u
1 date g
2 time g
3 site_id g
4 visitor_location_country_id g:u
5 visitor_hist_starrating g:u
6 visitor_hist_adr_usd g:u
7 prop_country_id g:i
8 prop_id g:i
9 prop_starrating g:i
10 prop_review_score g:i
11 prop_brand_bool g:i
12 prop_location_score1 g:i
13 prop_location_score2 g:i
14 prop_log_historical_price g:i
15 position g
```



```
16 price_usd g:i
17 promotion_flag g:i
18 srch_destination_id g:u
19 srch_length_of_stay g:u
20 srch_booking_window g:u
21 srch_adults_count g:u
22 srch_children_count g:u
23 srch_room_count g:u
24 srch_saturday_night_bool g:u
25 srch_query_affinity_score g:u
26 orig_destination_distance g
27 random_bool g
28 comp1_rate g:i
29 comp1_inv g:i
30 comp1_rate_percent_diff g:i
```

Besides, we considered the following two significant problems in feature extraction.
First is the removing of position bias, which is a very well-researched problem in ranking and points out the fact that items ranked in the top are more likely to be clicked because of their positions rather than relevance. Generally, there're two ways to handle this.

- **Estimate the position bias and adjust the training labels, train the model without position feature.** This is what we did in Random Forest, the adjustment is as follows:

outputTarget = a * booking_bool + b * click bool + c * position, where a > b > c, in our case, we choose a = 1; b = 0:2; c = 0:05. We did some work using RandomForest based on

Scikit-learn python package, but since we don't have time to give the detailed experimental results, that piece of work is not listed.

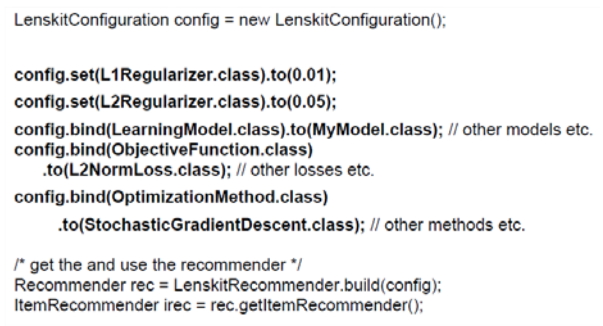
- **Train the model with position feature.** This is what we did in SVDFeature. Since what we want to do is to remove the position bias from the trained model, missing this piece of information in test data doesn't matter. This is the advantage of biased models.

Second is the type of features. There are two types of features involved: numerical features, categorical features. **For a categorical feature, we expand the values of this feature and train the bias or factors for every possible value.** We tried L1 and L2 norm as the **regularizer** with details in the following sections. To note, numerical features are **normalized**.

3. Lenskit and Our extension

3.1 Lenskit Introduction

Lenskit is a recommendation toolkit developed by GroupLens lab at U of M. Basically, our goal is to achieve the following fancy piece of code:



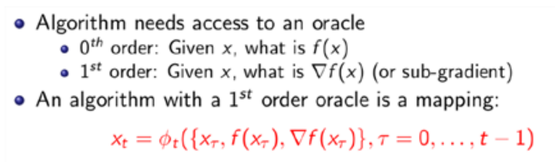
```
LenskitConfiguration config = new LenskitConfiguration();


config.set(L1Regularizer.class).to(0.01);
config.set(L2Regularizer.class).to(0.05);
config.bind(LearningModel.class).to(MyModel.class); // other models etc.
config.bind(ObjectiveFunction.class)
    .to(L2NormLoss.class); // other losses etc.
config.bind(OptimizationMethod.class)
        .to(StochasticGradientDescent.class); // other methods etc.

/* get the and use the recommender */
Recommender rec = LenskitRecommender.build(config);
ItemRecommender irec = rec.getItemRecommender();
```

3.2 Oracle-based training

The question is: "Is it possible to do this? How?". The core here is **oracle-based training.**

Consider f(x) is the model, the problem becomes: $\arg\min_{x} \mathrm{OBJ}(f((x))) + \lambda_1 |x| + \lambda_2 \|x\|^2$

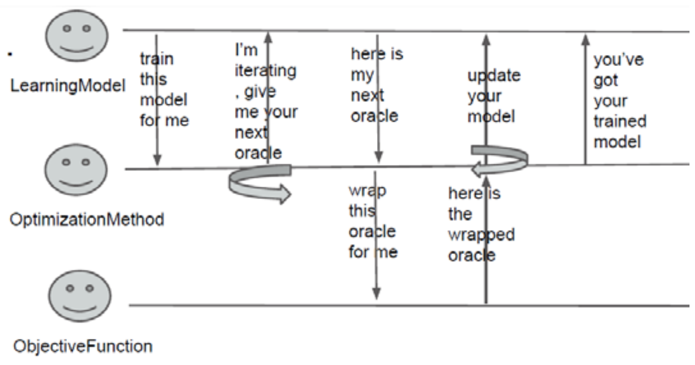
According to oracle-based training and complexity,



- Algorithm needs access to an oracle
  - 0^{th} order: Given x, what is $f(x)$
  - 1^{st} order: Given x, what is $\nabla f(x)$ (or sub-gradient)
- An algorithm with a 1^{st} order oracle is a mapping:

$$x_t = \phi_t(\{x_\tau, f(x_\tau), \nabla f(x_\tau)\}, \tau = 0, \dots, t-1)$$

So, we **designed the following OO framework in Lenskit** to incorporate many kinds of Optimization Methods, Loss Functions and Learning Models, which is **a very fun story**.



3.3 Our finished extension

Finally, we finished the following extension in Lenskit.

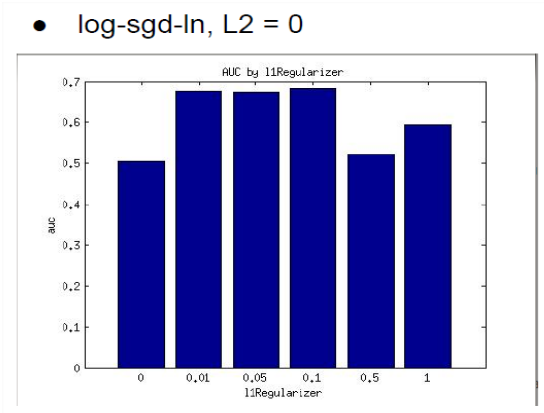
- Optimization Methods (with L1 and L2 reg): StochasticGradientDescent (sgd), AlternatingStochasticGradientDescent (asgd), CompositeObjectiveGradientDescent (cogd), AlternatingDirectionMethodsOfMultipliers (admm)
- Objective Functions: HingeLoss (hinge), L2NormLoss (l2), NegtiveLogLikelihoodLoss (log)
- LearningModels: SVDFeature (including LR, SVM), Kernel Functions: LinearKernel(ln), RBFKernel(rbf), PolynomialKernel(poly)

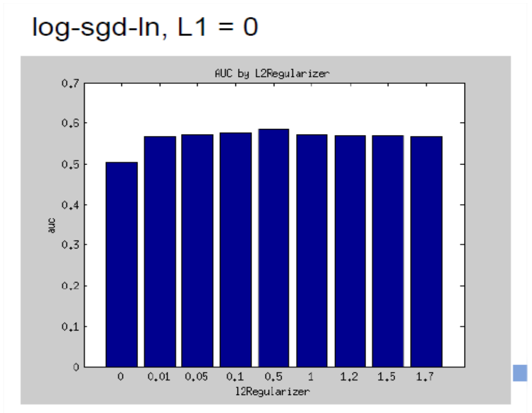
4. Experiments

4.1 Experimental Results

In the following figures given, we use loss-method-kernel to represent different choices regarding Loss Functions, Optimization Methods and Kernels. We only tried SVDFeature model, which essentially includes LR, SVM and Kernel PCA. The dimension is 10.

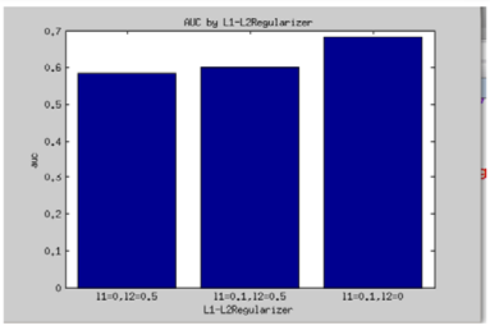
This is the figure showing the different of L1, L2 regularizers. **Apparently, L1 is doing a better job than L2.**

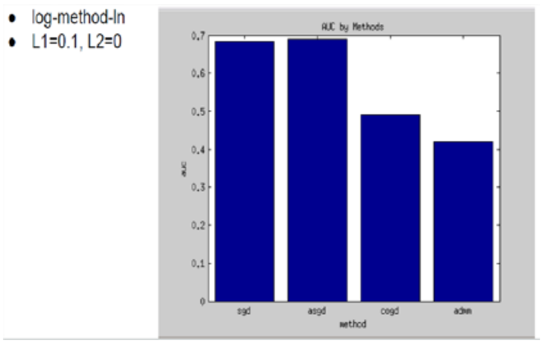




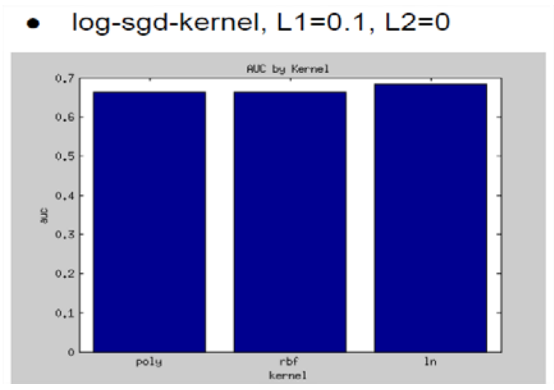
What if we use both L1 and L2 regularizer? **We can see L2 degrades L1, which is not a good thing**.

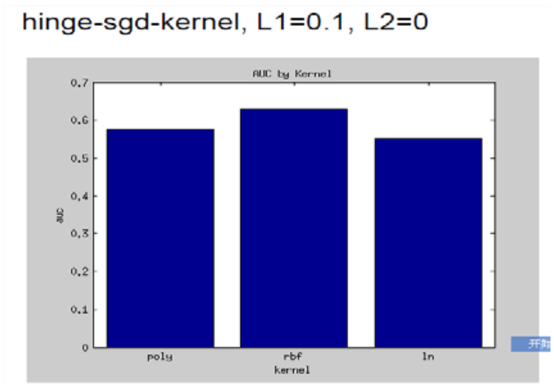
Following figure shows the difference among different optimization methods. The results for admm and cogd were not quite right, maybe because of the codes. We're going to fix it soon.



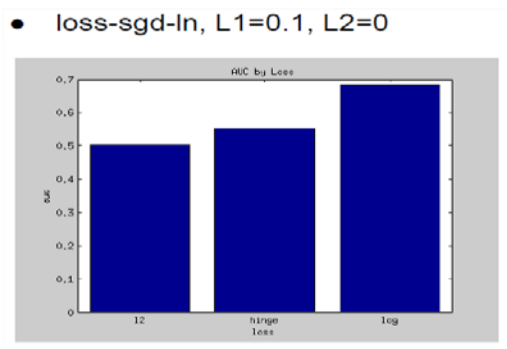


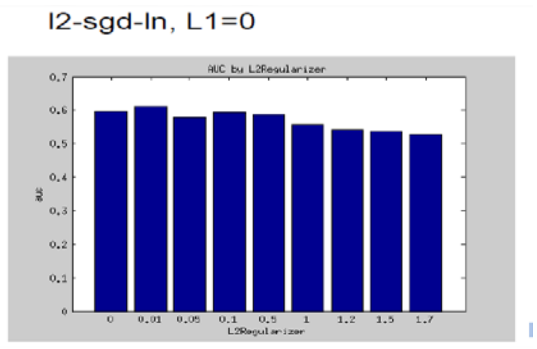
Following is the figure showing differences between kernels functions under log and hinge loss.





Following figure shows the difference among different loss functions.





Since the result for L2-loss is really weird, we tried L2-loss together with L2 regularizer, which improved the result and was very interesting.

4.2 Experimental Conclusions

We get the following experimental conclusions from the above data, which is really helpful to guide future work. To note, the results should be regarded as references, since different data might be different, especially when the models are different.

- Use L1-regularizer with Log-loss, Use L2-regularizer with L2-loss
- Don't use L1 and L2 regularizer together
- Use Hinge-Loss with RBF kernel, Use Log-Loss with Linear Kernel
- Use Alternating SGD

5. References

- Lectures and slides (by Prof. Banerjee and Nicholas) :
  - http://www-users.cselabs.umn.edu/classes/Fall-2013/csci5525/
- Lenskit
  - Main Repository: https://github.com/grouplens/lenskit
  - Our Repository: https://github.com/qiankun925/lenskit
- SVDFeature: http://svdfeature.apexlab.org/wiki/Main_Page