aLook report

Jan Hynek

February 2018

1 Executive summary - model description

I have used temporal abstraction to create churn variable for each user. For specified time point, I looked at the last transaction before which has become the yellow point in image below. From this point, I looked one year in the future, to see who really churned.

To preserve temporal dimension of the data, I created sets of variables for each user, for given set of periods, along with several time independent ones.

For potential deployment, I created clf.py, managable from terminal, which quickly transforms the matrices, uses model to create prediction.csv, which can be mapped to ID_user afterwards.

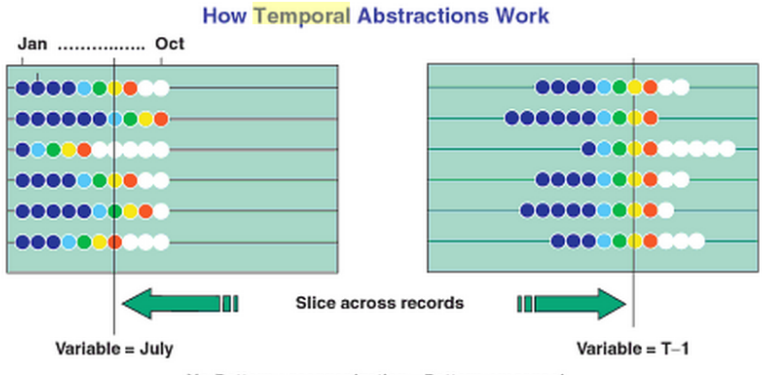

Figure 1: Source: Handbook of Statistical Analysis and Data Mining Applications, p.341.

I have split the datasets for high yielding and low yielding customers. Using grid search I have learned random forest to classify churn of the data for both of these datasets. When model classifies something as churn, it has **75% chance to really be churn**, and **75% of the true churns are identified**, and these numbers are higher in the case of high-yielding customers. When implemented for deployment, split is 30/70, total 8 models were learned to allow free choice of datasets, and obvious churns (last tnx $>$ 365 days) are taken out.

2 Technical documentation

2.1 Data preprocessing

To create sensible dataset to predict churn, I used temporal abstraction, as described in ExSum.

```
get subset before given time reference from transactions
get all users from this subset
for each user from subset:
    get most current transaction from subset
for each user from subset:
    look into transactions
    if there is transaction in one year after most current:
        append 0
    otherwise:
        append 1
```

This is wrapped in function ***create_churn***.

2.2 Feature extraction

Data have temporal dimension. Therefore, for each user I created:

- how much revenue he provided, and how many transactions he/she made in every of last N quarters.
- average price of the transaction overall (proxy for socioeconomic status)
- whether the transaction for given user were made mostly during working hours or in the evening.
- trend in the last three periods - was the user buying more or less than before?
- number of days from his last transaction

This is done in ***feature_extraction***

There was no need for *feature selection* in the case of *transactions.txt*, The processed data are still small enough($\sim$500kB), and the learning of the Random Forrest is still quick.

However, the other datasets are not reliable, (both are incomplete, and also user data cannot be trusted), therefore I decided that main analysis should be independent of these datasets, and should be used only for prediction improvement. The only necessary dataset for both prediction and modelling is *transactions.txt*, but user is free to choose another datasets as well.

users.txt is merged with transactions using ***process_users***, and creates columns indicating age, whether age is missing, male, and female. I did not add column indicating missing gender to escape dummy trap.

products.txt is merged with transactions using ***process_products***, and creates variables indicating most favourite category of given user, or whether his/her most favourite category was missing in the dataset.

All these functions are wrapped in function ***get_X_y_datasets***, which has following parameters, allowing that function can be used for both training and prediction:

- time_reference (pandas._libs.tslib.Timestamp), default Infer: timepoint, from which is whole dataset calculated if left as Infer, function takes timestamp which is one year smaller from maximal time stamp present in users dataset
- transactions (pandas.core.frame.DataFrame): raw dataset with transactions (5 columns) requires that column 'txn_time' is parsed as a date
- users (optional - pandas.core.frame.DataFrame): raw users table
- products (optional - pandas.core.frame.DataFrame): raw products table
- verbose (bool), default False: should function print reports?
- check_time (bool), default False: should function check, whether time_reference is at least one year?
- ndays_backward (optional - integer), default 365: how many days before timereference should be used to determine users present in the dataset?
- n_quarters (optional - int), default 8: how many quarters should be used to create temporal variables?

get_subset is helper function for creating subsets from matrices created by *get_X_y_datasets*. This was used during modelling and is also used in *clf.py*.

2.3 Model development & evaluation

First I looked at Naive Bayes, which unfortunately did not provide me with satisfying results. Afterwards, I tried **Random Forest Classifier from sklearn**. The results were pretty good with preset values, therefore I decided to perform several rounds of grid search. I was looking at

- n_estimators (number of trees)
- max_depth (number of layers)
- min_samples_split (how should be decisions in the tree made
- min_samples_leaf (minimal count of items in the end of the tree)
- max_features (how many variables should be taken into account)
- bootstrap (should training data be bootstrapped)

The objective function maximised was 'f1' score. During the grid search, I lowered the boundaries around the searched values in cca 4 iterations.

I evaluated the models on the basis of precision and recall and confusion matrix.

Regarding training and testing of the model, I had overall 5 years of data. They were processed followingly:

- I took first three years as training set
- Fourth year as testing set

I had to be sure, that anyone in the testing set is churned or not. This also allows for prediction in any given time. I also decided to train the models used for prediction only using these data as otherwise I would have no assurance that the models are not overfitted.

During analysis of the predicted data, I found out that with one model suiting whole data there is problem that model then analysis almost only on the basis of time from last txn.

This was big problem for high yielding users, 3 - 6 months from last TXN, who are going to churn. Therefore, I decided to split the dataset, on the basis of total revenue for given user, and create two models for each part.

While modelling, I decided to give higher weight for churn in 'rich' dataset (we really don't want to lose these people) and higher weight for not churn in 'poor' dataset (it was very imbalanced).

2.4 Potential deployment

For potential deployment I decided to create another python script, which can be run from commandline.

- Requirements: Python 3.6, pandas, numpy
- Installation: Just put contents of folder 'classifier' - *clf.py, data_manipulation.py* and *dual_models_all.pickle* in one folder
- Returns
  - prediction.csv in the same folder, with first column indicating row, second ID_user and third is prediction
  - optionally also *Xmatrix.csv*, matrix which was used for prediction
- Try with *'python clf.py path\to\transactions.txt'*

Usage:

```
usage: clf.py [-h]
              [--transactions TRANSACTIONS]
              [--products PRODUCTS]
              [--users USERS]
              [--refdate REFDATE]
              [--saveX SAVEX]

optional arguments:
  -h, --help          show this help message and exit
  --transactions      Address to transactions.txt
  --products          Address to products.txt (optional)
  --users             Address to users.txt (optional)
  --refdate           Date for which should be prediction made,
                      passed to pd.Timestamp().
                      If left, the most current date
                      in transactions dataset will be taken
                      (optional)
  --saveX SAVEX       If set to 1, script also saves
                      "Xmatrix.csv",
                      which was used for churn prediction
                      (optional)
```

3 Possible improvements

- Create automated learning script for customer. As this model is not online learned, it would need relearning, cca every year, to provide best results

4 Appendix

```
Includes transactions and products
Learned on year starting 3y from last date,
                ending 2y from last date
Tested on year starting 2y from last date,
               ending 1y from last date

Conf. Mat.
[[ 395  298]
 [ 258 1096]]

Low-yielding users - 70%
              precision    recall  f1-score   support

           0       0.60      0.57      0.59       693
           1       0.79      0.81      0.80      1354

avg / total       0.72      0.73      0.73      2047

--------------------
High-yielding users - 30%
[[331 148]
 [ 61 338]]
              precision    recall  f1-score   support

           0       0.84      0.69      0.76       479
           1       0.70      0.85      0.76       399

avg / total       0.78      0.76      0.76       878

--------------------
All
[[ 726  446]
 [ 319 1434]]
              precision    recall  f1-score   support

           0       0.69      0.62      0.65      1172
           1       0.76      0.82      0.79      1753

avg / total       0.74      0.74      0.74      2925
```