Research.md

Recosys Research

Definition of Problem

Given the following:

- A set of customers, C
- Corresponding customer metadata, $\overrightarrow{M_c}$
- A set of banking products, B

We wish to learn the following:

- A function $f : (c, \overrightarrow{M_c}) \twoheadrightarrow B \times [0,1]$ which assigns $\forall c \in C, \forall b \in B$ an associated probability $P_b \in [0,1]$ that customer c would use the product b : $f((c, \overrightarrow{M_c})) = \{(b_i, P_{b_i}) | b_i \in B\}$
- A function $g : (c, \overrightarrow{M_c}) \rightarrow \sigma(B)$ which maps each customer-metadata pair to a subset of the permutations of $\sigma(B)$: $(b_0, b_1, ..., b_n)$, $n \in dim(B)$, where the ordering of the permutation corresponds to the likely customer preference ranking: $g((c, \overrightarrow{M_c})) = (b_0, b_1, ..., b_n)$

Note that given f we automatically get g by ranking according to the absolute probabilities given by f.

Customer metadata primarily consists of transaction history in natural language and amount transacted, but may include additional metadata (demographics, social profile, etc...). We conjecture that transaction sequences contain more information and predictive power about customer state, and therefore susceptibility to recommendation. It is for this reason that we will want to go beyond classical collaborative filtering techniques which are unable to model sequential information.

Recommendation Systems Background

Recommendation systems primarily take two approaches to predicting relevant recommendations: collaborative filtering, and content based filtering.

Collaborative filtering works by calculating similarity metrics between users, then making recommendations according to similar user's preferences.

Content based filtering works by finding similarities between items, and then predicting new items that a user will like according to the similarity content of their previously liked items.

In practice these systems work in very similar manners. The recommendation system designer decides on relevant user features (which items liked, time spent looking at X product, etc...) and defines a large (often sparse) matrix of these user-item interactions. The former SOTA method involves factoring this user-item matrix into separate lower dimensional User and Item matrices, which are then used to calculate new predictions.

Formally, let the sparse user-item rating matrix M with $dim(M) = (u, i)$ where (u, i) denote the number of users and items respectively. Letting k = latent dimension, we approximately factor M with two lower rank matrices U, I such that $M = UI$, and $U \in \mathbb{R}^{users \times k}$, $I \in \mathbb{R}^{k \times items}$. Then we can fill in our sparse matrix (i.e. estimate user u's rating of item i by:

$$m_{u,i} = \sum_{j=0}^{k} U_{u,j} I_{j,i}$$

Interestingly, it has been shown that when d to many possible extensions, we propose to start there. Once we have transaction vectors that have encoded context and semantic meaning in relation to bank product usage, we will be able to use these va e used banking products.

So the flow is:

- Train transformer to learn encoder from transaction histories to latent space
- Train FC NN to take input latent vectors and output rankings (use softmax over all banking product categories to prefer the one actually used). Train only on data with banking product use.

We can also use the learnt latent embeddings to cluster users and then recommend products with a nearest-neighbor approach.

Going further, we can use a transformer with multiple tasks to get rankings directly out of the transformer decoder: In addition to the usual transformer decoder head outputting the next sequence element (a transaction in our case), we can add another FC head which outputs rankings over banking products when such training information exists (i.e. when the training sample includes a banking product as the next transaction item). This will nicely regularize the transformer and help to condition the decoder on the task we really care about: banking product ranking.

Training details

Once we have pre-trained out transformer to produce the latent vectors for transaction sequences, we feed these vectors to a fully connected (FC) neural network (note, this NN can be combined as a "head" on top of the transformer in a multi-task manner for complete end-to-end learning).

Because the vast majority of transactions will have no banking product usage, there is a strong class imbalance (i.e. most of the time the correct prediction is none). However if a customer *must be given a recommendation*, what should the ranking be? In this case a softmax with cross entropy loss gives a sharper answer, since we want to know about the *relative* ranking of different classes. However, a sigmoid layer with binary cross entropy loss is also desirable in that it will both allow for non-mutually-exclusive recommendations, and also encode something closer to an absolute probability of usage (keeping in mind the Absolute Probability Problems from above). Because we want a sharp relative ranking, but also an absolute probability (even given the caveats above), we propose to try both sigmoid and softmax final layers with cross entropy and binary cross entropy losses respectively (even trying both simultaneously).

There should be testing of various oversampling proportions between samples with and without banking product use. We imagine that examples with any banking product use will be so vastly underrepresented that they will need to be relatively largely oversampled if sigmoid + BCE is used. When softmax + CE is used, we only need to train on examples with banking product use, which will be much faster. (In the case of multi banking-product use, we can still train softmax + CE with a valid probability distribution, something like (in the case of the first two products used): $\hat{y} = (0.5, 0.5, 0, ...0)$)

Clustering

Using the trained transformer encoder, we can use the encoder output as a kind of context vector in the style of Word2Vec. These context vectors encode transaction sequences *conditioned on banking product recommendations*, and so will already contain information about what banking products customers are likely to use.

Using these context vectors to represent users, we can do classical clustering and take a k-NN approach to recommending banking products according to the most used product by other users in the cluster.

It may even be possible to learn FC NNs as above for each separate cluster - this will need to be tested.

Long Term Model

If we need to get absolute probability that a customer will use a banking product, we have to understand that use is conditioned on recommendation/knowledge of product.

To this end, we propose first deploying the above-mentioned transformer based system to make recommendations, recording when a recommendation is made and what the result is. When our dataset of banking product use after recommendation has grown large enough, we can train a new policy network (represented by the FC given the context vectors) on *all* the data for which recommendations are made using sigmoid + BCE with no caveats, at which point the values will represent true probability of use.

Minimum Viable Product in One Week

If we had only a week to implement a simple neural recommender system, we propose to use a word2vec style encoder on banking product usage *only*. Using this encoder we can cluster uses simply based on their recent banking product usage, then recommend the cluster centroid preference.

If transaction data also exists which is correlated with the banking product use and there is time, take a pre-trained transformer (like BERT), finetune on the credit card transaction data, and then combine the transaction encoding with the banking product encoding as inputs to a FC NN which is trained to output recommendations (where the transaction embeddings used are only those around the banking product use).

Banking Product as Transaction

Throughout this note we have considered banking products as elements of customer transaction data.

It is also possible to treat customer transaction data (such as those from credit cards (CC)) as distinct from banking product usage/transaction. In such a case, we would propose two separate encoders, a transformer for the CC data and a light-weight recurrent network for the banking products. These encodings could then be combined into a fully-connected network for the final recommendation.

However, we would guess that treating them jointly in a single large transformer would outperform separate encoders long-term. We must remember to only train the recommender loss when appropriate (see above), whereas the CC next-transaction head in the transformer is trained on all available data (putting us in a multi-task setting).

Roadmap

Since the transaction data is already available, we should start by training a transformer to predict the next transaction in a sequence given the previous items. This will provide a powerful transaction sequence -> latent vector encoder.

After the transformer is trained, we should do some discovery to see how well customers can be clustered. If clustering works well, we can consider a k-NN approach to making recommendations.

Next we should experiment with training a FC NN given the transformer context vector to produce banking product recommendations, trying both softmax + CE and sigmoid + BCE heads. (Also consider making the FC NN another head of the transformer and only training this head on samples with banking product use, rather than a completely separate NN). Consider making a separate FC NN for each cluster.

Combining clustering, a powerful classical approach to recommendation systems, with a modern NLP methods for transaction embedding with a FC recommender policy head is likely to bring the best chance to achieve SOTA results for banking product recommendation. Along the way we will likely also produce a SOTA "transaction predictor" by using the full transformer in its originally intended manner.

Refs

Kernel-Mapping Recommender System Algorithms

Non-linear Matrix Factorization with Gaussian Processes

Matrix Factorization Techniques for Recommender Systems

What Recommenders Recommend – An Analysis of Accuracy, Popularity, and Sales Diversity Effects

Word2Vec

Sentence2Vec

Gru4Rec

Transformer

Netflix Prize

Neural Collab. Filtering

RecVAE

Bert4Rec

Performance Comparison of Neural and Non-Neural Approaches to Session-based Recommendation

Multi Task Learning Auxiliary Targets