Santander Product Recommendation Challenge

Problem Statement:

Santander Bank is one of the North America’s top retails banks by deposits and a wholly owned subsidiary of one of the most respected banks in the world: Banco Santander. It’s parent company, Santander Group, serves more than 100 million customers in the United Kingdom, Latin America, and Europe.

Santander Bank offers a lending hand to their customers through personalized product recommendation to support needs for a range of financial decisions. Under their current system, a small number of Santander’s customers receive many recommendations while many others rarely see any resulting in an uneven customer experience.

The challenge here is to predict which products their existing customers will use in the next month based on their past behavior and that of similar customers. With a more effective recommendation system in place, Santander can better meet the individual needs of all customers and ensure their satisfaction no matter where they are in life.

Data Description:

- We are given a dataset which consists of 1.5 year customer behavior data. Approximately there are 1 million customers and the timeline for the data is from January 2015 to May 2016.
- There are approximately 13.5 million records which gives us the information about the customer and the products purchased he/she purchased.
- There are 24 variables which gives us the information about the customer profile, along with this there are 24 flags which gives the information about the various products offered by the bank.
- If a customer own’s a particular product, it will be marked as 1 otherwise it will be marked as 0.
Evaluation Metric:

- The evaluation metric used is Mean Average Precision @ 7 (MAP@7):

$$MAP@7 = \frac{1}{|U|} \sum_{u=1}^{[U]} \frac{1}{\min(m,7)} \sum_{k=1}^{\min(n,7)} P(k)$$

Where
- $|U|$ is the number of rows (users in two time points)
- $P(k)$ is the precision at cutoff $k$
- $n$ is the number of predicted products
- $m$ is the number of added products for the given user at that time point.

Note: If $m = 0$, the precision is defined to be 0.

Exploratory data analysis:

Exploratory data analysis is done on the dataset for data cleaning, adjusting some features, and visualize various important features.

- In the age field, we find that in addition to NA, there are people with very small and very high ages. It’s also interesting that the distribution is bimodal. There are a large number of university aged students, and then another peak around middle-age.
- We then look into distribution of age based on gender. The Sexo field had NA values which were converted to gender H.
- ind_actividad_cliente field tells whether a customer is active or not. This field also contained NA values and those records were converted to inactive status. Count of inactive customers is almost constant throughout the year.
- renta (Gross income of the household) is missing a lot of values. Rather than just filling them in with a median, it’s better to fill the missing values by the median of renta for a particular province.
Similar group of people tend to buy similar products under similar conditions. Analyzing the product popularity among different segments of people in June 2015 can be very useful for our problem.

**Segment 3 customer’s product popularity**
Current Account > Direct Debit > Payroll Account > Payroll = Pensions

**Segment 2 customer’s product popularity**
Current Account > Particular Account > Direct Debit > Payroll Account = e-account > Taxes = Pensions

**Segment 1 Customers Product popularity**
Current Account > Direct Debit = Long-term deposits = e-account
**Our Approach:**

**Recommender Systems**

Recommender systems are tools for filtering and sorting items and information. They use opinions of a community of users to help individuals in that community to more effectively identify content of interest from a potentially overwhelming set of choices. There is a huge diversity of algorithms and approaches that help creating personalized recommendations.

Two of them became very popular: collaborative filtering and content-based filtering. They are used as a base of most modern recommender systems.

**Traditional Recommender Approaches**

**Content-based filtering:**

Content-based recommender systems work with profiles of users that are created at the beginning. A profile has information about a user and his taste. In the recommendation process, the engine compares the items that were already positively rated by the user with the items he didn’t rate and looks for similarities. Those items that are mostly similar to the positively rated ones, will be recommended to the user.

![Content-based Filtering Diagram](image)

This approach is not possible to meet the needs of our problem statement since it is difficult to define the products of banking (e.g. Credit account, Savings account, Payroll account etc.) with its content.

**Collaborative filtering:**

The idea of collaborative filtering is in finding users in a community that share appreciations. If two users have same or almost same rated items in common, then they have similar tastes. Such users build a group or a so called neighborhood. A user gets recommendations to those items that he/she hasn’t rated before, but that were already positively rated by users in his/her neighborhood.
User based approach:

![User based approach diagram]

Item based approach:

![Item based approach diagram]

These approaches are most followed by various organizations to recommend products for their existing customers while they do have a cold start problem with new customers. Decent number of kagglers tried collaborative filtering based approaches by clustering users based on their preferences and recommending the products most preferred in the respective cluster. However, this type of approach doesn’t consider the situation or context in which the user preferred that particular product.

**Algorithms behind these approaches:**

**Neighborhood models:**

Neighborhood methods use similarity functions such as the Pearson Correlation or Cosine Distance to compute sets of neighbor’s to a user or an item. Recommendations are then computed by using data from those neighbors.

**Latent Factor models:**

On the other hand, latent factor models such as Matrix Factorization (MF) solve the recommendation problem by decomposing the user; item matrix and learning latent factors for each user and item in the data. The underlying assumption is that both users and items can be modelled by a reduced number of factors. This approach has proven to be the most accurate method in isolation in different settings.

Although the simplified user; item recommendation model can be used successfully in many settings, it is not uncommon to find real settings in which additional variables come into play. For instance, there are many situations where time plays an important role in defining a user’s preference for an item. In this case, the two-dimensional matrix is turned into a three-dimensional (user; item; time) tensor. The set of variables that influence the user’s preference for a given item are referred to as context.
**Context-Aware Recommender systems** are categorized into three types; contextual pre-filtering, where context drives data selection; contextual post-filtering, where context is used to filter recommendations once they have been computed using a traditional approach; and contextual modelling, where context is integrated directly into the model.

We chose to implement the contextual modelling approach by directly integrating the context into the recommender model. This approach cannot be employed with the general neighborhood models. Few applications have been developed using contextual graphs, SVM’s etc. which are more practical with contextual information from social networks.

**Implicit vs Explicit Feedback**

**Explicit feedback:** includes explicit input by users regarding their interest in products or their experience.
Eg: Ratings and reviews for products.

**Implicit Feedback:** User preferences are inferred from more abundant implicit feedback which indirectly reflect opinion through observing the past user behavior.
Eg: Purchase history, browsing history, search patterns.

**MULTIVERSE RECOMMENDATION**

The main idea behind the use of **Tensor Factorization** is that we can take advantage of the same principles behind Matrix Factorization to deal with N-dimensional information.

**Matrix Factorization:**
CF techniques based on MF work by assuming that ratings provided by users on items can be represented in a sparse matrix $Y_{n \times m}$ where (n is the number of users and m the number of items). The observed values in $Y$ are thus formed by the rating information provided by the users on the items. The CF problem then boils down to Matrix Completion problem. In MF techniques the aim is to factorize the matrix of observed ratings into two matrices $U_{n \times d}$ and $M_{m \times d}$ such that $F = UM^T$ approximates $Y$, i.e. minimizes a loss function $L(F,Y)$ between observed and predicted values. In most cases, a regularization term for better generalization performance is added to the loss function.

**Tensor Factorization:**
The aim in proposing an N-dimensional TF approach for context-based recommendation is to model the context variables in the same way as the users and items are modelled in MF techniques by taking the interactions between users-items-context into account. Note that in contrast to SVD and HOSVD methods, in CF there is no need for imposing orthogonality constraints on the factors.
Higher Order SVD decomposition:

\begin{align*}
\text{Input } Y, \quad d \\
\text{Initialize } U \in \mathbb{R}^{n \times d_U}, \quad M \in \mathbb{R}^{m \times d_M}, \quad C \in \mathbb{R}^{c \times d_C} \quad \text{and} \\
S \in \mathbb{R}^{d_U \times d_M \times d_C} \text{ with small random values.} \\
\text{Set } t = t_0 \\
\text{while } (i, j, k) \text{ in observations } Y \text{ do} \\
\quad \eta \leftarrow \frac{1}{\sqrt{t}} \quad \text{and} \\
\quad t \leftarrow t + 1 \\
\quad F_{ijk} = S \times_U U_{i*} \times_M M_{j*} \times_C C_{k*} \\
\quad U_{i*} \leftarrow U_{i*} - \eta \lambda_U U_{i*} - \eta \partial U_{i*} l(F_{ijk}, Y_{ijk}) \\
\quad M_{j*} \leftarrow M_{j*} - \eta \lambda_M M_{j*} - \eta \partial M_{j*} l(F_{ijk}, Y_{ijk}) \\
\quad C_{k*} \leftarrow C_{k*} - \eta \lambda_C C_{k*} - \eta \partial C_{k*} l(F_{ijk}, Y_{ijk}) \\
S \leftarrow S - \eta \lambda_S S - \eta \partial S l(F_{ijk}, Y_{ijk}) \\
\text{end while} \\
\text{Output } U, M, C, S
\end{align*}

Advantages of Multiverse Recommendations:
1. No need for pre- or post-filtering.
2. Computational simplicity.
3. Ability to handle N-dimensions.

Graphlab’s Ranking Factorization Recommender:

Ranking Factorization Recommender trains a model capable of predicting a score for each possible combination of users and items. The internal coefficients of the model are learned from known scores of users and items. Recommendations are then based on these scores.

In the two factorization models, users and items are represented by weights and factors. These model coefficients are learned during training. Roughly speaking, the weights, or bias terms, account for a user or item’s bias towards higher or lower ratings. For example, an item that is consistently rated highly would have a higher weight coefficient associated with them. Similarly, an item that consistently receives below average ratings would have a lower weight coefficient to account for this bias.

The factor terms model interactions between users and items. For example, if a user tends to love romance movies and hate action movies, the factor terms attempt to capture that, causing the model to predict lower scores for action movies and higher scores for romance movies. Learning good weights and factors is controlled by several options outlined below.
More formally, the predicted score for user $i$ on item $j$ is given by

$$score(i, j) = \mu + w_i + w_j + a^T x_i + b^T y_j + u_i^T v_j,$$

where $\mu$ is a global bias term, $w_i$ is the weight term for user $i$, $w_j$ is the weight term for item $j$. $x_i$ and $y_j$ are respectively the user and item side feature vectors, and $a$ and $b$ are respectively the weight vectors for those side features. The latent factors, which are vectors of length $\text{num_factors}$, are given by $u_i$ and $v_j$.

The model is trained using Stochastic Gradient Descent [SGD] with additional tricks to improve convergence. The optimization is done in parallel over multiple threads. This procedure is inherently random, so different calls to create recommender may return slightly different models, even with the same random seed. The objective function we attempt to minimize is:

$$\min_{w,a,b,v,u} \frac{1}{|D|} \sum_{(i,j) \in D} L(score(i, j), r_{ij}) + \lambda_1 (\|w\|_2^2 + \|a\|_2^2 + \|b\|_2^2) + \lambda_2 (\|U\|_2^2 + \|V\|_2^2)
+ \frac{\lambda_{rr}}{\text{const} \cdot |U|} \sum_{(i,j) \in U} L(score(i, j), v_{ir}),$$

**Data preparation for model:** The dataset contains 24 product columns with 1’s at the purchased products making it a huge sparse matrix. This sparse matrix is converted into a list of products purchased using **THIS** function.

**Model:** The create method of ranking factorization recommender is provided with a Graphlab’s sframe containing user ID’s column, Products purchased column, and also side data i.e context variables are provided to the model. This recommender also takes parameters for various regularizations.

Training the model using only 1st month data (due to space and memory constraints):

Recsys training: model = ranking_factorization_recommender
Preparing data set.
Data has 1222935 observations with 690203 users and 24 items.
Data prepared in: 8.46824s
Training ranking_factorization_recommender for recommendations.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>num_factors</td>
<td>Factor Dimension</td>
<td>32</td>
</tr>
<tr>
<td>regularization</td>
<td>L2 Regularization on Factors</td>
<td>1e-009</td>
</tr>
<tr>
<td>solver</td>
<td>Solver used for training</td>
<td>adagrad (Adaptive Gradient Stochastic Gradient Descent)</td>
</tr>
<tr>
<td>linear_regularization</td>
<td>L2 Regularization on Linear Coefficients</td>
<td>1e-009</td>
</tr>
<tr>
<td>binary_target</td>
<td>Assume Binary Targets</td>
<td>True</td>
</tr>
<tr>
<td>side_data_factorization</td>
<td>Assign Factors for Side Data</td>
<td>True</td>
</tr>
<tr>
<td>max_iterations</td>
<td>Maximum Number of Iterations</td>
<td>25</td>
</tr>
</tbody>
</table>

**Result:**
Optimization Complete: Maximum number of passes through the data reached.
Computing final objective value and training Predictive Error.
   - Final objective value: 0.445319
   - Final training Predictive Error: 0.166249

**Model Evaluation:**
Evaluation of the model which is trained on the January 2015 purchase data on February 2015 purchase data gave the following precision and recall measures for various cut-off values (i.e. no. of products to recommend)

<table>
<thead>
<tr>
<th>cut-off</th>
<th>precision</th>
<th>recall</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.0427157688541</td>
<td>0.0213942943986</td>
</tr>
<tr>
<td>2</td>
<td>0.031316748286</td>
<td>0.0274873754431</td>
</tr>
<tr>
<td>3</td>
<td>0.0250316682991</td>
<td>0.0318707616771</td>
</tr>
<tr>
<td>4</td>
<td>0.0210005876592</td>
<td>0.035659813822</td>
</tr>
<tr>
<td>5</td>
<td>0.0183346523017</td>
<td>0.0393068046938</td>
</tr>
<tr>
<td>6</td>
<td>0.016269539667</td>
<td>0.0420585854933</td>
</tr>
<tr>
<td>7</td>
<td><strong>0.0146818525255</strong></td>
<td><strong>0.0446674039422</strong></td>
</tr>
<tr>
<td>8</td>
<td>0.0133398628795</td>
<td>0.0465346640344</td>
</tr>
<tr>
<td>9</td>
<td>0.0122428120579</td>
<td>0.0479978790477</td>
</tr>
<tr>
<td>10</td>
<td>0.011419079334</td>
<td>0.0499663302026</td>
</tr>
</tbody>
</table>

**Predictions and Recommendations:**
The built model has a recommend method which takes a test SFrame as input with same structure as the training SFrame and number of products to recommend.
```python
predictions = recommend.recommend (users = test_sframe, k=7).groupby("ncodpers", 
{"added_products": gl.aggregate.CONCAT("product")})
```

**The Quandary of Additional products:**
The problem statement of kaggle competition Santander Recommender Systems states to find what additional products a customer shall use. Since the initiation of competition, there has been a lot of discussion going on concerning what exactly does additional products means. An
understanding regarding this which shall be implemented is to generate new probabilities of owning the products from the probabilities given by the recommender by multiplying them with a filter matrix.

\[ \text{New Probabilities} = (1 - \text{products\_owned\_in\_may2016\_matrix}) \times (\text{probabilities generated by the recommender for June2016}) \]

This shall nullify the probabilities for the products already owned by a particular user and then one can choose top 7 products with higher probabilities.

**Validation sets:**

This recommender model needs to be tuned to our dataset by adjusting various parameters such as step size and other regularization parameters. With basic understanding of the problem statement and dataset, we can create two different types of validation sets:

1. Random split by user: Create a training set of 80% of users and a validation set of the remaining.
2. Split by time: Create a training set containing 14 months’ data and a validation set containing the 15\textsuperscript{th} month data.

**Model Parameter search:**

Graphlab provides with a model\_parameter\_search function which takes a tuple containing training, validation sets as input along with the type of model needed to create. This function schedules a job to find the parameters suitable for our dataset by evaluating the model at each step. Graphlab has an optimal way of performing the parameter search by using the local Async environment and running the job in background. Parameter search on a recommender uses precision and recall in case of implicit feedback and finally provides the precision and recall values for various combinations of different values of parameters. The values of parameters can be provided in 3 ways:

- **Manual search:** Manually provide a set of parameters to search for.
- **Random search:** Randomly searches for different values of parameters.
- **Grid search:** Searches through a range of values of parameters with fixed increments for particular parameters

**Code Repository:**

[https://drive.google.com/open?id=OB33Pc_oABKzSOl1Vlg2bnUzN1k](https://drive.google.com/open?id=OB33Pc_oABKzSOl1Vlg2bnUzN1k)
References:

1. Collaborative Filtering for Implicit Feedback Datasets; Yifan Hu, Yehuda Koren, Chris Volinsky.(Research paper behind the used ranking recommender)
2. Incorporating Contextual Information in Recommender Systems Using a Multidimensional Approach; Gediminas Adomavicius, Ramesh Sankaranarayanan.
3. Algorithms and Methods in Recommender Systems; Danar Asanov

Contributors:

GROUP-13
Chandra Mouli Kallu
Pruthvi Kiran Pokala
Ranjit Kumar Malloji
Sai Sandeep G