Creating a Product Recommendation Engine for a Large Retail Bank

1. Business Problem

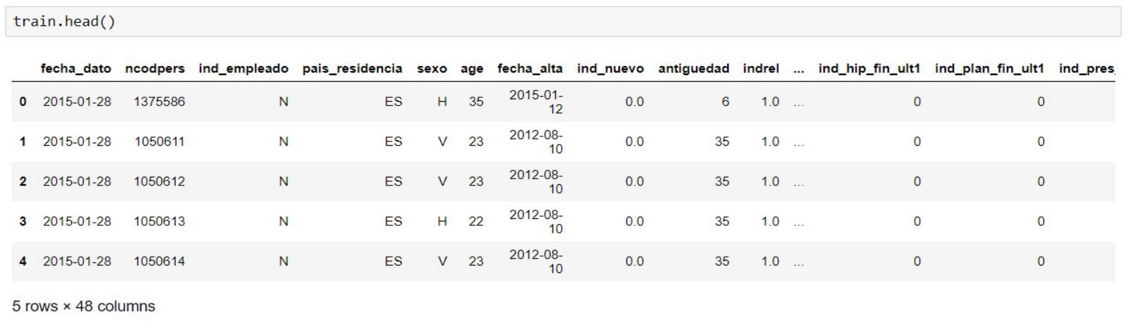
The problem to be solved in this project is how to create a model that predicts which products a bank's customer will buy next month given the available data on a bank's customers. The bank is essentially interested in implementing personalized product recommendations for all of its customers. Under their current system, a small number of the bank's customers receive many recommendations while most of their customers rarely see any. The bank is interested in having a more effective recommendation system in place so that it can better meet the individual needs of all its customers and ensure their satisfaction no matter where they are in life. This project is base on the Santander Product Recommendation Kaggle competition. The evaluation metric for the competition was the mean average precision at seven (MAP @7). Therefore the model should deliver a rank ordered list of up to 7 products for every customer in the test set.

2. Client

Santander Group is a multinational banking conglomerate based in Spain. Its chief holding is Banco Santander the largest bank in Spain, and although the data was provided with a disclaimer that this is not their real customer data, after reviewing it seems this data is at the least, based on the customer data of Banco Santander. The variety of products to be recommended in this data seem to be typical of a retail banking operation. The products include savings accounts, certificates of deposits, credit cards, mortgages and various other types of accounts. It's worth mention that although they named this a product recommendation challenge its specifically a product prediction challenge and although these two tasks do have plenty of overlap in their approach and implementation there are distinct analytical approaches taken for each.

3. Description Of Data

The data was available as two csv files labeled train and test. Each row is for one customer for a certain month. The test file is the customer records for June 2016 with the values for all the products missing. The train file is customer records for the previous 17 months with a column for each product filled with binary values to indicate whether the customer owned the product in that month. There are 24 different product columns and 22 columns that offer demographic data about the customers.



```
train.head()
```

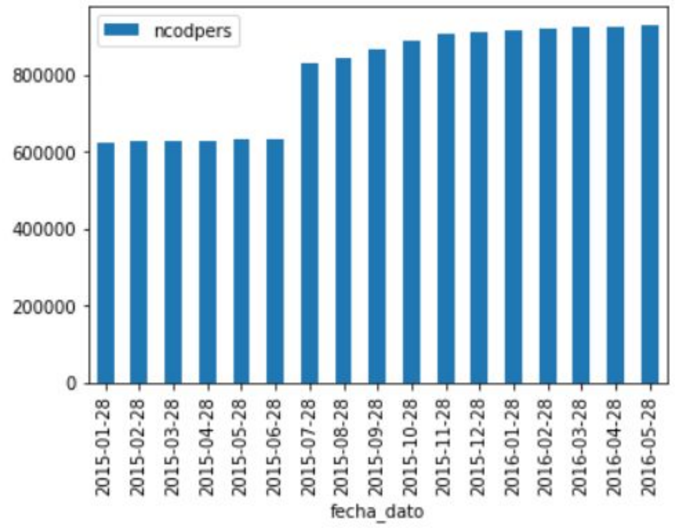
	fecha_dato	ncodpers	ind_empleado	pais_residencia	sexo	age	fecha_alta	ind_nuevo	antiguedad	indrel	...	ind_hip_fin_ult1	ind_plan_fin_ult1	ind_pres
0	2015-01-28	1375586	N	ES	H	35	2015-01-12	0.0	6	1.0	...	0	0	
1	2015-01-28	1050611	N	ES	V	23	2012-08-10	0.0	35	1.0	...	0	0	
2	2015-01-28	1050612	N	ES	V	23	2012-08-10	0.0	35	1.0	...	0	0	
3	2015-01-28	1050613	N	ES	H	22	2012-08-10	0.0	35	1.0	...	0	0	
4	2015-01-28	1050614	N	ES	V	23	2012-08-10	0.0	35	1.0	...	0	0	

5 rows × 48 columns

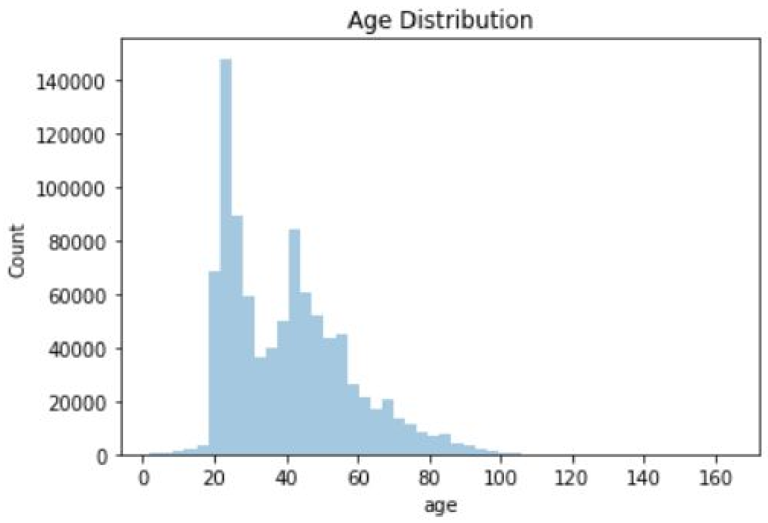
There were plenty of missing values in this data set. I approached the data column by column to decide how I should deal with this issue. First I noticed what looked to be certain rows that just had multiple missing values across many columns and I just dropped all those rows. For the registration method column and the column which flags for if the customer is a spouse of an employee I imputed the missing values as unknown since most values in these columns where missing. There were 2 columns for the province of the customer, one with the full name and one with an abbreviation, so I just dropped the abbreviation column and imputed the most common province name which happened to be Madrid. There was column for the type of customer with 4 numerical codes for each type and I decided to just replace these with strings for the name of the customer type. Then I imputed the missing values here with what was by far the most common type - primary. For the household income there was pretty good option for smartly imputing values which was to take the median income of the province the customer resided in. Similarly for the account type column I imputed missing values based on the income bracket of the customer. For gender and customer relation at end of month there weren't too many missing values so I imputed those with the most common value. Two of the product columns had some missing values so I just imputing those as 0 ie. the customer did not have the product. And for the customer seniority there were some negative values which I just replaced with the median.

4. Initial Findings

Since the data was sorted by month, each date being the 28th of the month in sequential order I wanted to see how the number of customers changed over time and how each month of data where different from each other.

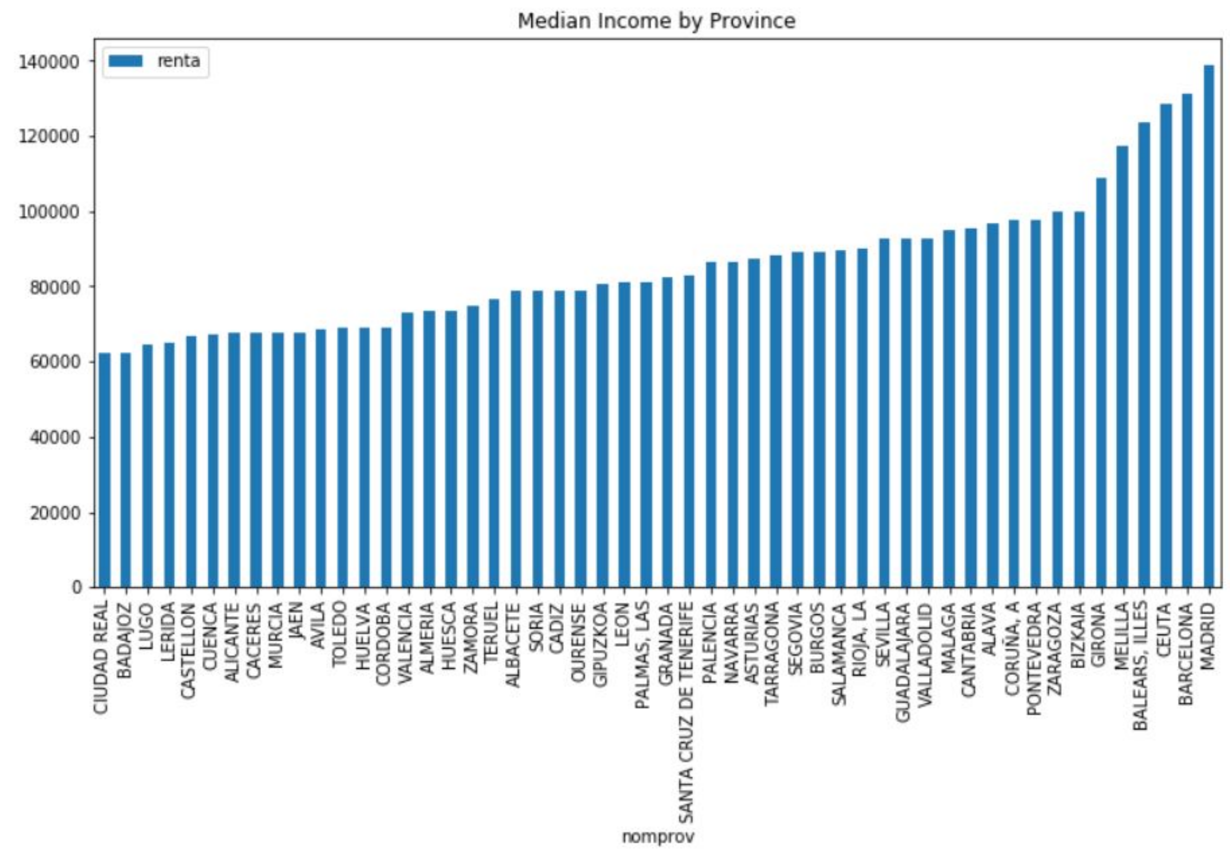


You can see that in June 2015 there was a large jump in the number of customers and then increased gradually from then. Upon further review there was a small amount of customers that dropped out over time, though for most part, most customers remained in the dataset after entering it and actually there are data points for all months for most of the customers in the data set. Now for analyzing some traits of the customer population.



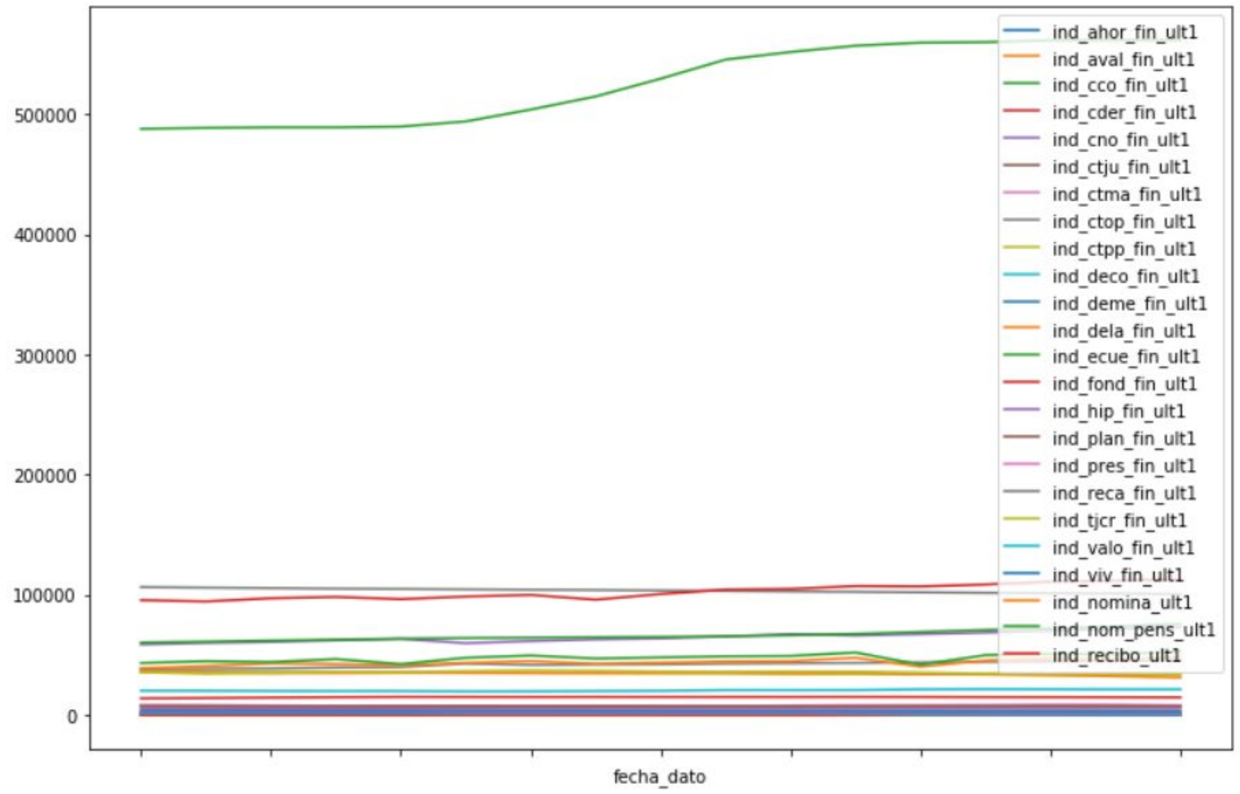
Looking at the age distribution there does seem to be a few outliers that may be due to error such as a handful of ages above 150 and probably a few too many young ages for this bank data. Also as seen in the histogram it appears to follow a bimodal distribution around the early 20's and mind 40's.

The next plot shows the median gross household income by province. This is relevant since I used the median income of the province to impute missing income for customers in that province. You can see Madrid and Barcelona have the highest median incomes. There are no units specified and these numbers are much higher than what the typical Spanish household income in Euros is, according any estimates I have seen online. However, the relative differences among provinces do seem to be vaguely accurate. So it does beg the question of how exactly this dataset was constructed and how much domain knowledge may be applicable vs how much prediction success here is a result of extensively data ming an arbitrary dataset.



One of the main things I had to grapple with in this dataset is how to identify when a customer bought a new product. It sounds simple enough, but the way the data was arranged actually made this less than straightforward. Each data row only said whether a given customer owned a given product in a given month, so to find out whether that product was purchased in that month I would have to reference the entry for that customer in the prior month. Combined with the fact there are between 6-9 million customers arranged in no particular order for each of 17 different months with 24 possible products they may have, hopefully you will see that identifying when customers bought a product required a careful approach.

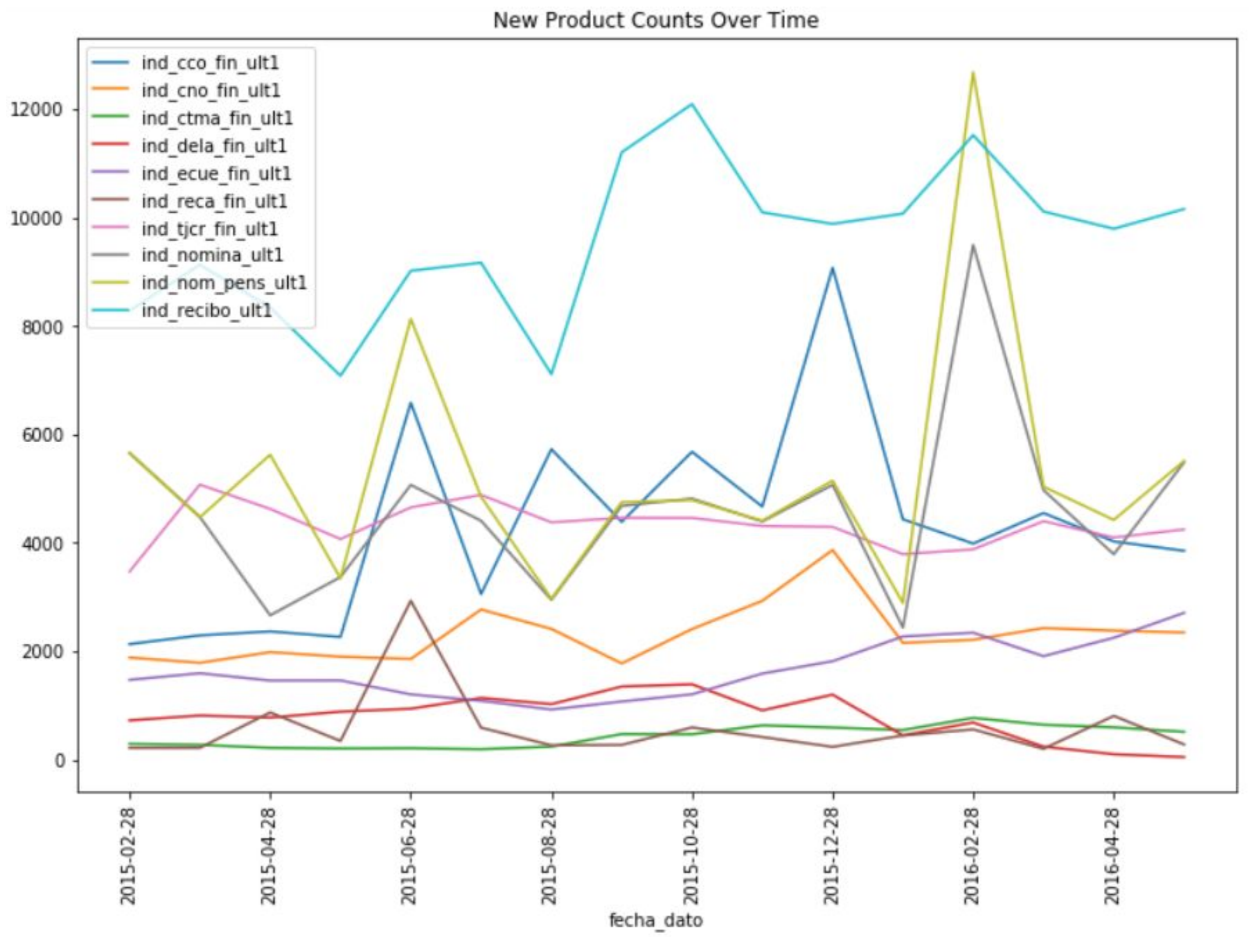
For example one of the first things I wanted to plot was the counts of new products over time so I grouped the products by date and aggregated by sum and quickly got this plot:



One thing you can see here is that there is one product that is much more common than all others ‘ind_cco_fin’, what is known as a ‘current account’ in British English or a checking account in American English.

However I realized that this just showed me the total count of each product for each month in the dataset and did nothing to differentiate products that were new for that customer that month.

Later I used a loop to iterate through sequential pairs of months, select only customers that were in both months and subtract the prior months’ product values from the current month. Then I filtered out all the zeros and negative one’s to get just rows representing a customer buying a new product. This allowed me to create the following plot where I used just the top ten products so that all lines will be distinguishable.



Much later on, after building, training and testing a variety of prediction algorithms I realized that the above chart might be the most important part of this project. That's because there seems to be a seasonal nature to what products Santander's customer buy and most of the machine learning models I built did nothing to capture this seasonality. When I started making models that focused on what customers did around the same time last year, their competition scoring results got significantly better. Later on I read on the public forum posts for this Kaggle competition that June is the end of the tax year in Spain, and if you look at June 2015 on the chart you can see its new product distribution is quite different from other months.

Pre-Processing for Modeling:

So in order to train some supervised learning algorithms and leverage the power of popular and successful packages such as Scikit-Learn it is necessary to 'massage' your data into the form of something resembling two matrices. Namely, the X-train and y-train matrices with X-train being a user by independent variable (feature) matrix and y-train being a user by dependent variable matrix. This is how I went about doing that for this dataset. Since I was only interested in building a model which predicted which *new products* a customer would have in June 2016 I made the X-train consist solely of instances where customer bought new products and the y-train those new products. To do this I looped through the 17 month train file and filtered out rows where there was a 1 in a product-is-owned column in the current month but a zero in the previous month. I did this by subtracting customers previous month row from the current month and then replacing all the negative ones with zeros to result in 446,000 rows with 24 binary 'is-new' columns as opposed to the original train file which has 13,647,000 rows with 24 binary 'is-owned' columns.

The 17 demographic columns that where present in both the train and test files had many missing values and had a mixture of categorical and numerical data. There were about 27,000 rows that had missing values for almost demographic features so I decided to just drop all of those. For the registration method column I filled the missing values with the string 'unknown' and did the same for the column indicating whether the customer was a spouse of an employee. There was a province code column that was redundant because of the accompanying province name column which I dropped. There were columns for customer type and customer type at the end of the month and each contain numerical codes in a mixture of string and int form. I converted these to strings of the customer types. For missing values of province name I replaced with the mode which was Madrid. For missing values of gross household income I imputed the mode of the province for which the customer belonged. Also for the customer's sex and seniority I imputed missing values with the mode and median respectively.

Feature Engineering:

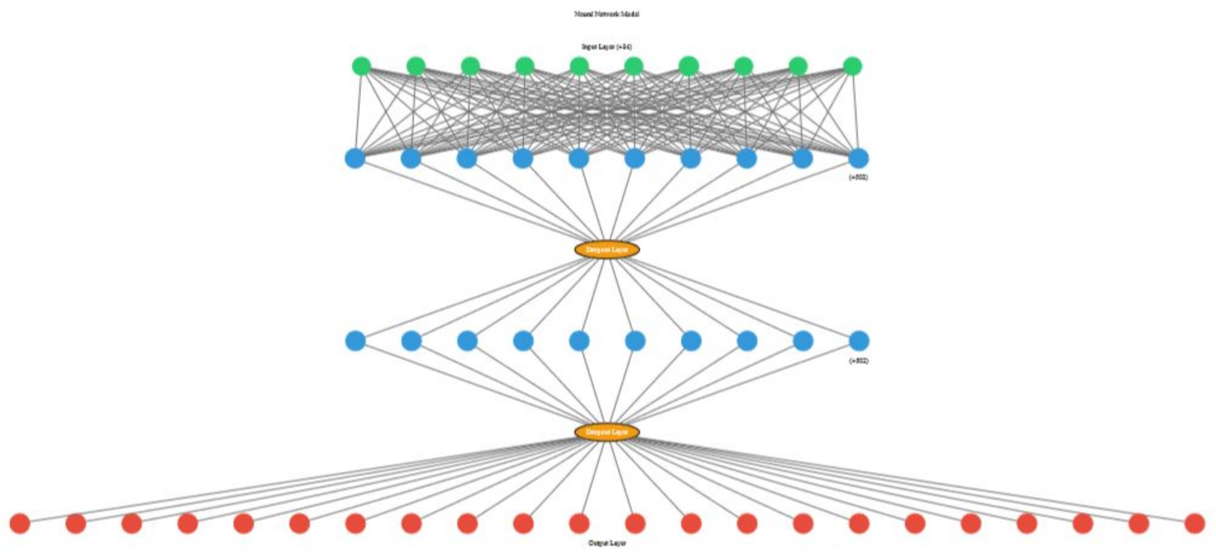
In an effort to distill more information from the demographic features I created a few myself. This included segmenting customers into age groups and income groups, binary values for residents of foreign countries or major cities, time customer has been with the bank in year. As it turned out later none of these were very useful at all.

5. Model Selection and Performance

Since this was a multiclass classification problem the two types of prediction algorithms I thought I would try were a tree based model (random forest, extra trees cfl., gradient boosted tree) and a neural network model.

5.1 Neural Network Multi-class Classifier:

So after performing data cleaning and feature engineering on both the train and test sets I decided to go with the TensorFlow library and use the Keras package on top of it as a high level API for developing a neural network classifier. Since a neural network is essentially just a series of layers of nodes that perform activation functions there are limitless ways to build one and the question of what constraints should be used as far as hyperparameters is a an emerging field of study. But for multi-class classification, a feed forward neural network known as a multilayer perceptron is an established choice. MLP's have at least layers including 'hidden' middle one and use backpropagation during training to tune the weights of the connections between nodes.



I used the ANN Visualizer package to create a graph visualization of my MLP model. Green is used for input nodes, blue hidden layers and red for output nodes. The yellow ovals represent the dropout layers used for back propagation and the connecting lines represent the weights. The two hidden layers actually contain 512 nodes both but this was left out by this was abbreviated by the visualizer.

In the Kaggle competition put on by Santander they were specific evaluating model performance by the mean adjusted precision at 7, (MAP@7) meaning they only cared about the top seven recommendations. This evaluation metric doesn't exist in the keras API and although I could have made a custom function and used it with train-test-split, I decided I would get much more meaningful feedback from the complete scoring of full contest submission. This model gave a score of 0.0205575. For reference the winning scores were approximately 0.0313 and the submission benchmark was 0.0042109

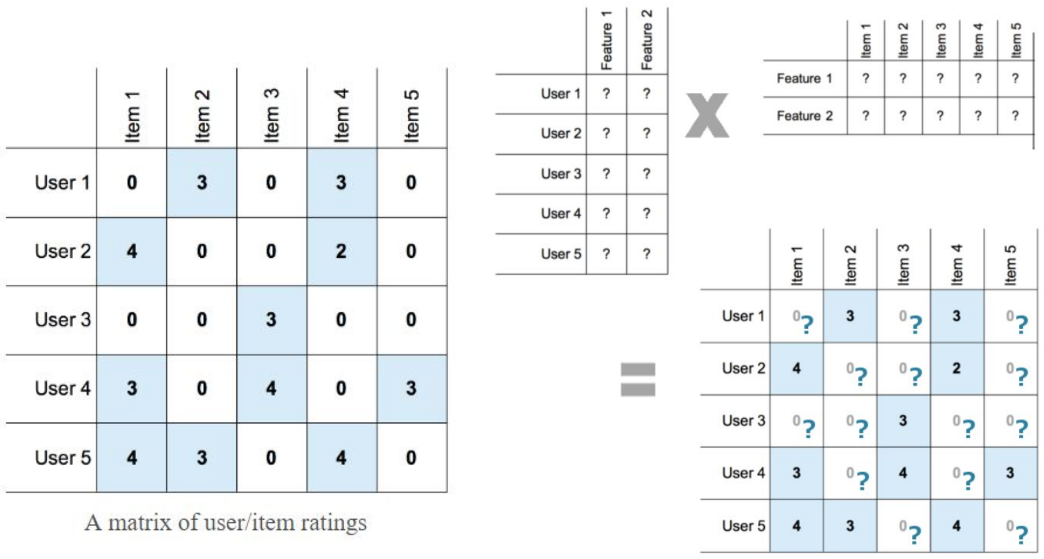
5.2 Random Forest Classifier:

Next I tried using a random forest classifier plugged in to the same processed train and test sets with the same logic for excluding recommendations for products the customer already has. The results were significantly worse even after trying hyperparameter tuning. This led me to think I had a lot of room for improvement as far as my feature engineering went and to consider model ensembling.

5.3 Collaborative Filtering:

I also started to research and learn about how the recommendations systems I encounter myself work, such when I use websites such as Amazon and Youtube. I really became interested in this topics as a I learned that algorithms for recommendations systems are almost like an entire separate field from what I learned so far in Springboard, combined with fact that I interact with, and to be honest, am influenced by recommendation systems on a daily basis.

The basic idea behind the latent feature based collaborative filtering (matrix factorization) that I used is that is that you can create a user by product matrix where the values that populate the matrix are some type interaction (ratings) data that you have on users and products. So you may not have data for all pairs of users and products in the matrix but you can approximate what the missing ratings are using some math called matrix factorization and an algorithm that numerically solves for a latent feature matrix. In theory these latent features could correspond to some real world traits that are shared by both users and items to varying degrees.



	Item 1	Item 2	Item 3	Item 4	Item 5
User 1	0	3	0	3	0
User 2	4	0	0	2	0
User 3	0	0	3	0	0
User 4	3	0	4	0	3
User 5	4	3	0	4	0

A matrix of user/item ratings

	Feature 1	Feature 2
User 1	?	?
User 2	?	?
User 3	?	?
User 4	?	?
User 5	?	?

X

	Item 1	Item 2	Item 3	Item 4	Item 5
Feature 1	?	?	?	?	?
Feature 2	?	?	?	?	?

=

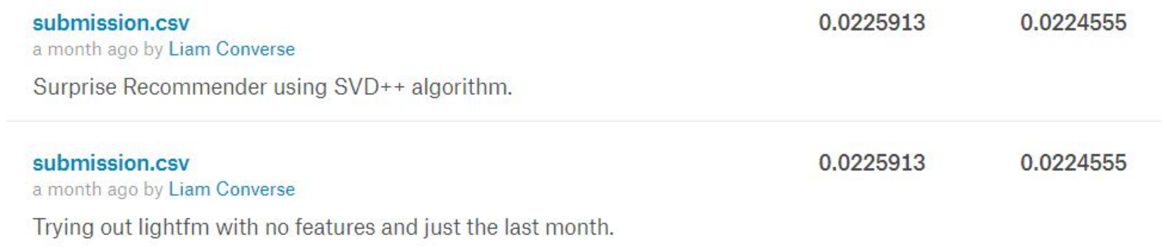
	Item 1	Item 2	Item 3	Item 4	Item 5
User 1	0?	3	0?	3	0?
User 2	4	0?	0?	2	0?
User 3	0?	0?	3	0?	0?
User 4	3	0?	4	0?	3
User 5	4	3	0?	4	0?

5.3.1 SVD with the Surprise! Python Package

I discovered a python package called Surprise! That was designed to be an easy to use scikit for recommender systems. One of the matrix factorization algorithms included in it was Singular Value Decomposition (SVD) which was made famous for its success during the Netflix Prize competition. So I tried it out, explored a range of hyperparameters, generated some predictions, and got a score of 0.023802, my best yet. Also within this package was the SVD++ algorithm which was designed to work better for instances of implicit feedback as opposed to explicit user ratings. The feedback that I had to work with was implicit, so supposedly SVD++ should have been a better choice but when I tried it out I found it to score slightly worse. And since there were many entries in the competition that scored at least 0.030, I knew I could do better.

5.3.2 Hybrid Filtering with the LightFM Python Package

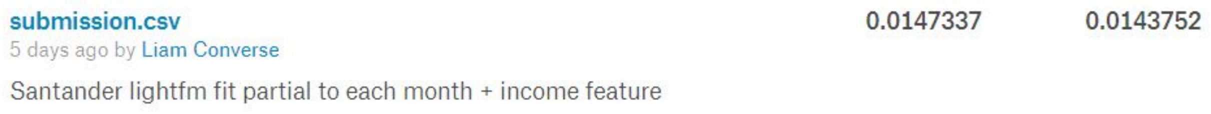
I discovered LightFM which on a paper appeared to offer many thing that should have made it superior to Suprise, and perhaps in slightly different use case would have been. Instead of just estimating latent features for user-item interactions LightFM allows you to estimate latent features for user-item metadata. The idea being that in some applications you have far fewer distinct metadata features than distinct items. This allows you to estimate a model with fewer parameters. Additionally LightFM has a fit_partial method that enables retraining of the model over time and can make predictions for new users. I first tested it out with just user and item interactions and noticed that it scored exactly the same as the SVD++ algorithm from before.



submission.csv
a month ago by Liam Converse
Surprise Recommender using SVD++ algorithm.
0.0225913 0.0224555

submission.csv
a month ago by Liam Converse
Trying out lightfm with no features and just the last month.
0.0225913 0.0224555

Then I tried adding some user demographic features to the model starting with income group. This only had the effect of drastically decreasing the score. Since in order to use this model, I had to do some considerable data preprocessing and it was not optimally compatible with windows as far as training time, I decided to try something else.



submission.csv
5 days ago by Liam Converse
Santander lightfm fit partial to each month + income feature
0.0147337 0.0143752

5.4 Neural Network Recommender with the Spotlight Python Package

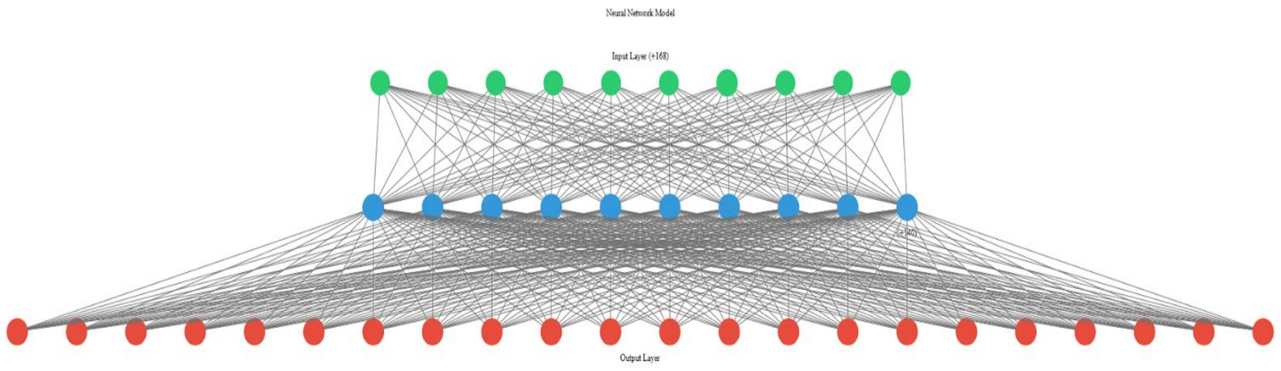
Much like LightFM, on paper the Spotlight recommender offered a number of benefits over every other model I tried. It runs using the Pytorch neural network framework which can make use of my computer's CUDA enabled GPU to accelerate training and prediction. Also Spotlight has in addition to factorization models, deep learning sequential models which look at recommendation as a sequence prediction task given the past sequence of user item interaction. But unfortunately I ran into issues with my hardware that made me back off, although spotlight and PyTorch are very promising options for building recommendation engines useful across multiple domains.

5.5 Producing Maximally Precise Predictions

As mentioned earlier, I eventually realized the importance of the seasonal nature of the new product data. This is because June 2015 had a significantly different distribution of newly purchased products. Reading back through the Kaggle competition forums a competitor by the name of "BreakfastPirate" completely changed the complexion of the contest when posted publicly that the reason he was able to achieve a MAP@7 of ~ 0.30 (topping the leaderboard at the time) was because he limited his train set to June 2015, and only instances of new products being purchased that month.

So I checked what score I would get if I simply recommended the most commonly bought products in June 2015, excluding products the customers already owned. Much to my dismay this entry scored higher than all of my previous machine learning models that required exorbendent computer memory and training times.

Producing recommendations this way garnered a MAP@7 on the order of 0.024. Getting to a MAP@7 was a matter of feature engineering, specifically 'lag' features that displayed what different products the different users who bought new products in June 2015 had in the months leading up to June 2015. This posed another multifactorial problem of how many months prior to include for each user and whether also considering products they owned in the year after June 2015 improved accuracy. What I settled on was using the 5 months prior to June as the lag features for both the train and test (evaluation) sets, the train set being for June 2015 and the evaluation set predicting for June 2016. Preprocessing the data this way I was finally able to create predictions that scored a MAP@7 above 0.030 using a Keras neural network classifier that can be seen below. The dropout layers were not used as overfitting isn't really a concern when you're just trying to get the highest score possible. Note there are actually 178 nodes in the input layer, not all are drawn in the visual.



Additional Considerations

One way to improve overall quality of recommendations would be to combine different algorithms and models together in an intelligent manner. For example since the SVD solves for latent features that it uses to compute predictions, these latent features should in theory express meaningful information across different subsets of users and can be extracted with the surprise package and added as features of a user for certain time windows when running tree-based or neural network classifiers to recommend product for specific time windows.

6. Conclusion

Ultimately, there is a large difference between building a prediction model for the purposes of winning a data science competition and building a prediction model at production scale in a commercial setting. In a competition the ratio of hours spent modeling to gains in precision reaches a point of diminishing returns that would not be wise in a business setting. And, in the end, winning models are likely to be overfitted in order to produce the highest possible score. As can be seen with this data, the model that is super precise this month can be pretty inaccurate the next month. I propose to the hypothetical client to use as a base model collaborative filtering with the SVD algorithm. The simplicity and ease of implementation of this approach provides strong, robust baseline recommendations that can then be improved upon. As seasonal changes in customer buying take effect, more complex data mining models can be back tested and then ensembled with the baseline recommendations.