**Introduction**

Ready to make a down payment on your first house? Or looking to leverage the equity in the home you have? To support needs for a range of financial decisions, [Santander Bank](https://www.santanderbank.com/us/personal) offers a lending hand to their customers through personalized product recommendations.

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.

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.

A host of different of different methods and algorithms were built, and are described in more detail below.

# Decision tree Engine

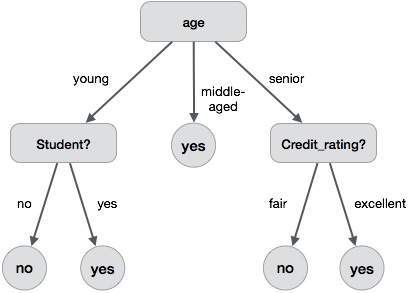
One of the simplest, easiest to comprehend algorithms out there, Decision trees are generally the starting point of most predictive systems, and give a good idea of the concept of predictive modelling.

## 1.1 Setting up the system

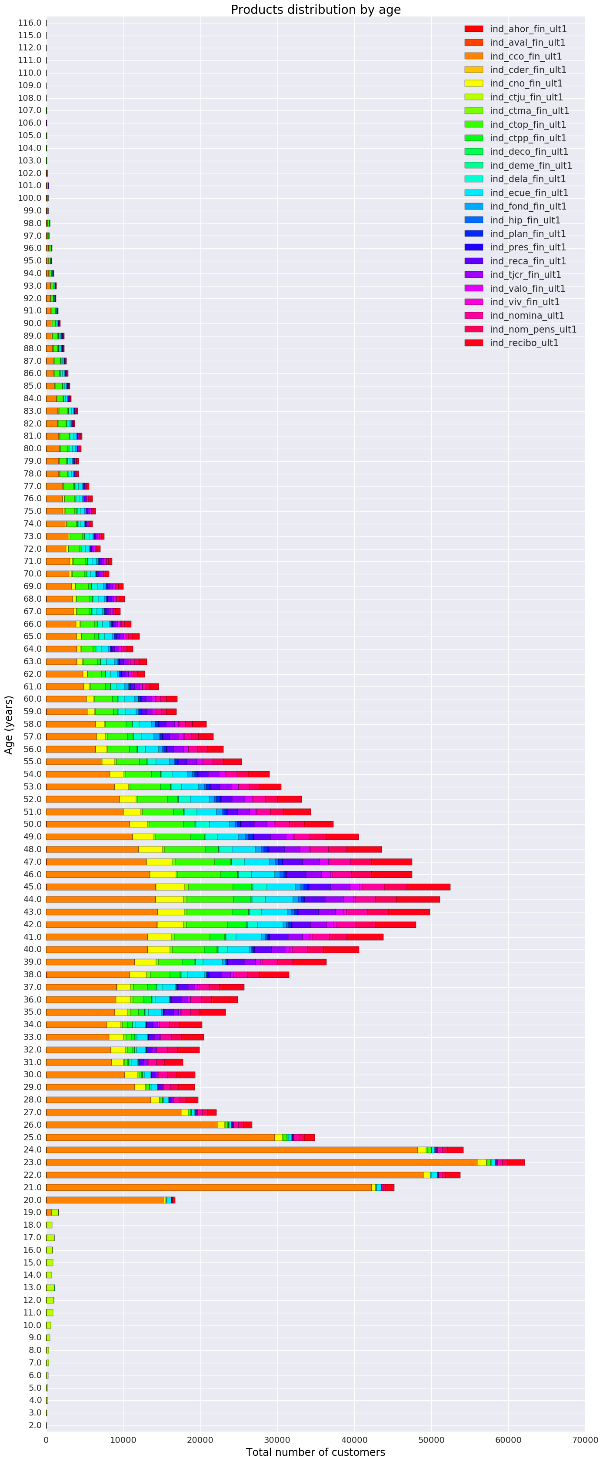
The first major step in Machine learning (ML) algorithms is data exploration, with the aim of exploring the features and identifying the most important.

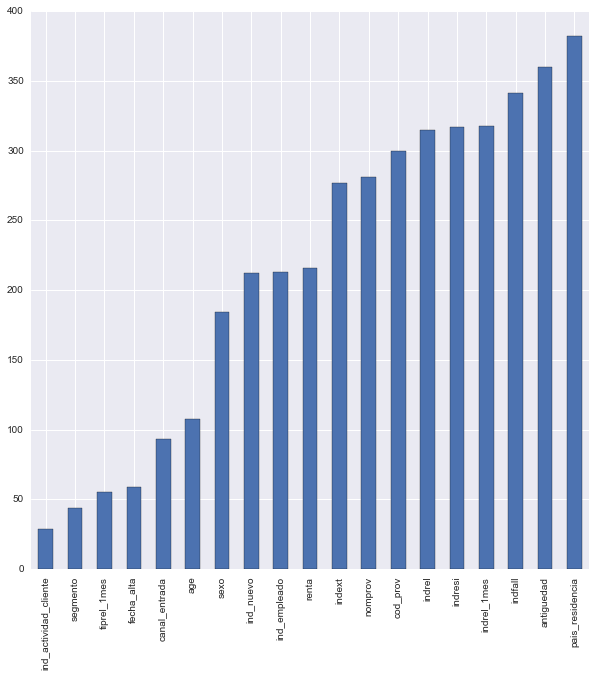
A Feature is defined as an individual measureable property off a phenomenon being observed, in our case, a feature describes an aspect of a customer we are analysis, an example being a customer’s age.

The use of features in ML decision trees is paramount, with the algorithm basing all its predictions on features. The way features are used is shown in an example decision tree is shown in Figure (1), where the vertices labels are the feature parameters.

**Fig (1): A visual representation of a simple decision tree similar to the ones built by our model to predict products.**

Finding most important features involved data engineering and cleansing, specifically null value imputation, followed by the use of **sklean.selectKbest** library to determine the most influential data features on what the customer is likely to buy.

**Fig (2): Graph displaying products owned split by customer age. This is an example of some of the data exploration performed in order to find the most important features to use.**

**SelectKbest** library involves looking at the target variables present, and the correlation between those and the features inputted, giving a score of the most influential ones. The results of this analysis are displayed in figure 3.

**Fig (3): a graph representing the importance of the features on the total number of products bought by customers, in a negative scale, meaning the lowest valued features are the most influential.**

With the most important features discovered, work underwent on producing a simple working decision tree model, with the plan of using the predicted probabilities for products as a measure for our recommender engine.

## 1.2 Results

The engine managed to produce relatively accurate predictions based on the training set provided (~85% accuracy). Returning the probabilities of predictions and formatting them into an accepted format was deemed too inefficient for the task. Therefore once the code was completed (alterations meant automatic cleansing and engineering of both training and testing data sets), work shifted towards a more suitable prediction engine.

# XGBoost

Notice that this recommendation engine is based around a publicly available script provided on Kaggle by author SRK.

XGBoost is an algorithm that has recently been dominating applied machine learning and data science competitions. It is an implementation of gradient boosted decision trees designed for speed, and most importantly, incredible performance.

## 2.1 Defining the problem

Recommendation engines are generally split into two categories. They can rely on the properties of the items (content-based filtering) or on the similarity of products between users (collaborative filtering). For this algorithm, a hybrid model was incorporated to achieve maximal results, blending both the characteristics of the customers and the similarity of the product they purchased.

## 2.2 Setting up the system

Initial work involved importing the script and relevant programs needed to run the recommender engine (XGBoost is a notoriously difficult package to install on windows).

## 2.3 Data cleaning

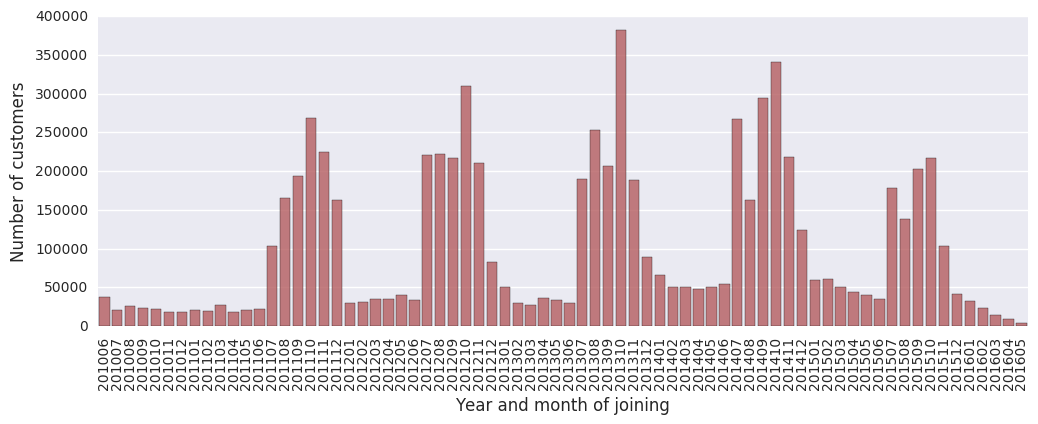
XGBoost manages only numerical vectors. As most of our data were categorical variables, we transformed them in an ingestible input for the algorithm via the one-hot encoding method.

The numbers of records processed through the ML algorithm were minimized by only looking at the customers that have purchased products within our specified date range, rather than all customers. Since the purpose of this project is to predict products customers are most likely to buy, rather than likelihood of them owning a product, it makes sense to only process products purchased.

## 2.4 Feature Engineering

There seems to be two important determinants in the performance achieved in a machine learning solution: features engineering and choice of algorithm.

A blend of the customers defining features, such as Age, Activity and Region were used as the typical features. The customer’s product list from the same time the previous year were also listed, in order to capture seasonality. Seasonality was an important factor in this project, and is shown appropriately in figure 4.



**Fig (4): A graph of number of customers and what month they joined. It is important to note how a spike develops yearly around June, displaying a seasonality trend in our dataset.**

In addition to seasonality (which for June in Spain seemed to be an influential factor since this is the month tax returns are due) we also wanted to portray time series trend effects, specifically how the customers buying patterns fluctuate leading up the to the months we want to predict.

This was achieved by adding in lag\_product features, comprising of a list of product ownership per customer dating back the previous months.

**Fig (3) Diagram showing the total number of customers, split by month and year. As clearly visible, there is strong seasonality with number of customers spiking after June each, a strong pattern that we wanted to capture within our algorithm.**

## 2.5 Algorithm

Short for “extreme gradient boosting”, It is based on the tree ensemble model, which is a set of classification and regression trees (CART). Our problem is a classification one, as the target variable is a set of 22 binaries, representing whether a product is recommended or not. While a classifier is a method for assigning predictions (the target variable) to some input variables by recursively partitioning the training data space and fitting a simple prediction model within each partition, XGBoost algorithm takes into account a multitude of trees in order to improve the performance.

XGBoost differentiates itself from the other tree ensemble models by the way it is trained. The training happens via the optimization (minimization) of an objective function, made up of two parts: the training loss, which measures how well the model fits on the training data, and the regularisation term, which measures the model complexity and helps to smooth the weights assigned to the trees in order to avoid overfitting.

This optimisation underlines what is known as the bias-variance trade-off. The majority of data science problem solving rely on the assumption that a model shall have an adequate prediction ability, even if it is at the expense of precise predictions on the training data. This avoids overfitting (low bias), the phenomenon which impedes the model from having good generalisation by fitting too closely to the training data.

As the objective function includes functions as parameters, it cannot be optimized with traditional methods. Instead, the model trains additively. Trees that most improve the model (chosen by the exact greedy algorithm) are added to the predictions during many rounds until the regularized objective function (multiclass negative log-likelihood) is minimised.

To let place for other trees and prevent overfitting, two methods are used: the shrinkage parameter (learning rate), which applied to the new added trees to the predictions reduces the individual impact of the trees, and column subsampling, which uses only a subsample ratio of the column for each split.

## 2.6 Parameters

Outlined here are the parameters we specified for our algorithm.

### 2.6.1 Learning task parameters

* **Objective:** ‘multi: softprob’

Specifies the learning task and the corresponding learning objective. We chose ‘multi: softprob’, which stands for multiclass classification using the softProb objective (returning a matrix of classes and probabilities for each customer)

* **Eval\_metric:** ‘map@7’

Chosen validation measure. We choose the same as the one used in the Kaggle problem, the multiclass negative log-likelihood.

* **Seed:** ‘seed\_val’

Value used to generate reproducible results.

### 2.6.2 Booster Parameters

* **Eta:** 0.05

Learning rate. A lower rate gives the model a better generalisation but must be offset computationally by doing more rounds to capture the residuals.

* **Max\_depth:** 6

Maximum depth of a tree. A too high depth might lead to overfitting.

* **Num\_class:** 22

Number of classes to classify by; it’s the number of target variables basically.

* **Min\_child\_weight:** 2

Minimum number of customers to split the tree further.

* **Subsample:** 0.85

Fraction of observations to be randomly sampled for each tree.

* **Colsample\_bytree:** 0.9

Subsample ratio of columns for each split, in each level.

* **Num\_rounds:** 100

Number of rounds.

## 2.7 Performances

The engine in its default configuration returns a score of 0.0261 on Kaggle, which is good enough for a top 1000 place finish on the leader boards.

The final model achieved a score of 0.0303, achieving a 170th place in the leader boards, a top 10% finish.

Regarding the parameters tuning, many values have been tested, but there were no clear trends whether augmenting or diminishing the value of a parameter jointly or not would be beneficial to the performance of the algorithm.

## 2.8 Advantages

1. **Regularization**

As the objective function comprises a regularization term working as a penalty term, it prevents the model of overfitting.

1. **Parallel processing**

This makes the algorithm very fast.

1. **High flexibility**

The user can build custom optimization objectives and evaluation criteria.

1. **Handling missing values**

The algorithm can by default impute or skip through null values, minimizing engineering and formatting efforts.

1. **Tree pruning**

The algorithm makes splits up to the Max\_depth specified, starts pruning the tree backwards, and remove splits beyond which there is no positive gain, resulting in the simplest possible method.

## 2.9 Future improvements

* **PCA Analysis**

Perform a PCA analysis to only keep features accounting for most of the variance.

* **K-fold cross-validation**

Implement a k-fold cross-validation, with k=5-10. This is a solution to avoid overfitting on the test set, which can still happen, if the parameters of the model are tweaked until the estimator performs optimally.

In this approach, the training set is split into k smaller sets. A model is then trained using k-1 of the folds as training data. The resulting model is validated on the remaining part of the data (i.e., it is used as a test set to compute a performance measure such as accuracy).

* **Add other features**

Think of other features that might improve the performance. (e.g. Lag\_product correlation measures.)

* **Neural networks**

Experiment a multi-layered neural network model, which might yield improvement regarding performance.

# K-Nearest Neighbours

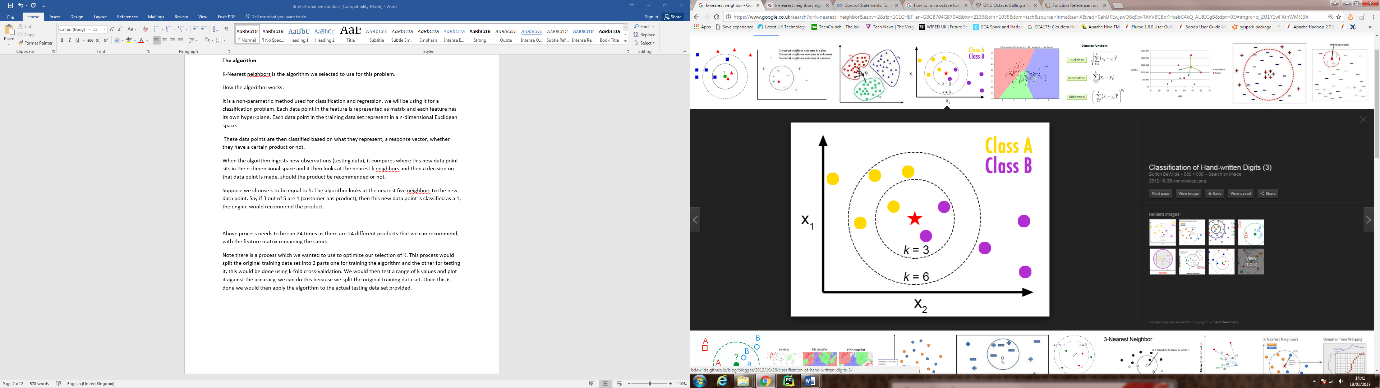
A very popular algorithm in regression problems, K-NN finds many uses in predictive models, especially in clustering methods. One of the simplest models, however a powerful and widely used tool for data scientists.

## 3.1 The algorithm

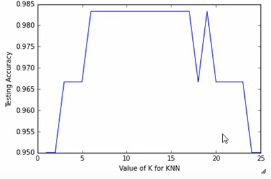
k-nearest neighbours is a non-parametric method used for classification and regression; we will be using it for a classification problem. Each data point in the feature is represented in a matrix and each feature has its own hyper-plane.

These data points are then classified based on what they represent, a response vector, whether a certain product should be recommended or not.

When the algorithm ingests new observations (testing data), it compares where this new data point sits in the n-dimensional space and it then looks at the nearest k neighbours and then a decision on that data point is made, should the product be recommended or not.

Suppose we have two features x1, x2 and the yellow and purple coloured dots represent the different classes (response vector). If a new data point is inserted, namely the red star. The value of k determines how many neighbours we will consider in order to classify this new data point. If k is 3 then we if clear that the new data point should be classified as purple (the majority of the 3 neighbours is purple), a similar process is used to make a decision when k is 6. This explanation is shown visually in figure 5.

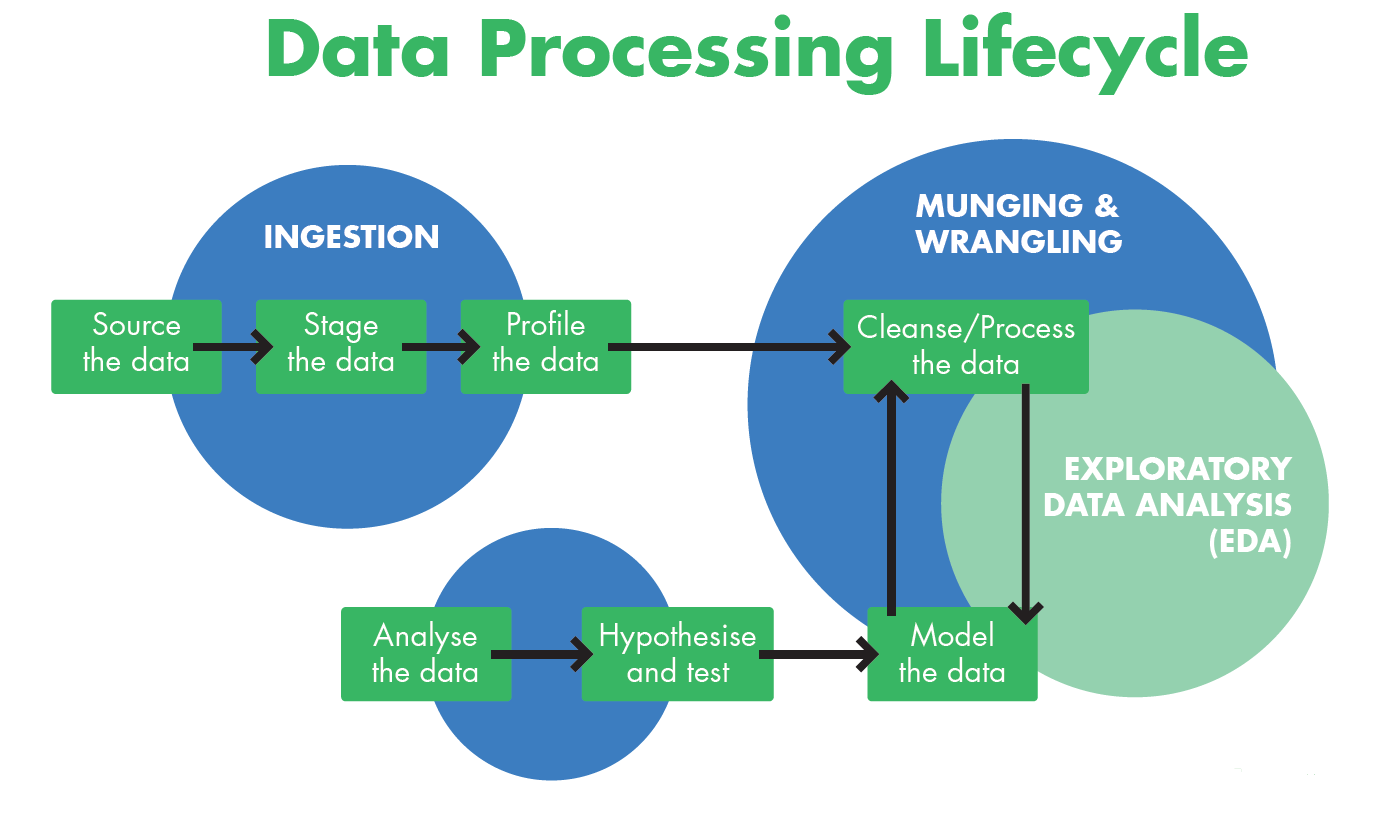
**Fig (5): A visual explanation of the K-NN method for categorizing data points.**

Above process needs to be run 24 times as there are 24 different products that we can recommend, with the feature matrix remaining the same.

Note there is a process that we wanted to use to optimize our selection of K. This process would split the original training data set into 2 parts one for training the algorithm and the other for testing it, this would be done using k-fold cross-validation. We would then test a range of k values and plot it against the accuracy (figure 6). Once this is completed, we would be apply the algorithm to the actual testing data set provided.

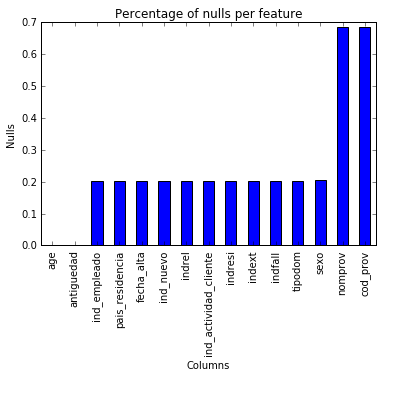
**Fig (6): results plot for K-NN cross validation.**

## 3.2 Data preparation

**Fig (7): the data processing lifecycle used in our algorithm production. A standard procedure in most data science applications.**

Tools used for this model:

* Apache Spark - Initial choice
  + The volume of data volume did not require a distributed environment.
* IPython
  + Easily deal with large volume of data.
  + One tool does it all - Data engineering, data visualisation and data analytics.
  + Easy to debug.
* Python libraries
  + Pandas – Data manipulation.
  + Scikit-Learn – Machine Learning library, train and test models in a speedy manner



Data Munging and Wrangling:

* Appropriate feature selection – we used the columns which had the least amount of nulls
* Reduce duplications – use the latest activity of the customer
* Impute nulls and outliers
* KNN works at an optimum level when the feature matrix consists of numbers between 0-1

## 3.3 Future improvements

**Fig (8): graph displaying the ratio of nulls in each feature columns. This is the basis on which we based our feature selection on.**

Due to time and resource limitations, the actual algorithm could not be implemented. A future major improvement could be obviously to introduce a working model.k

There was one interesting way we found of how to select the data for our feature set, if we had more time this is something we want to implement. We believe this would make our recommendation engine more accurate and versatile.

Our current model is based on entering the response vector, which is n by one vector. The idea and hope was to then get the responses of all the 24 results and combine them together and give a set of products that should be recommended to the customer. To further this model, the use of Support Vector Machine, would have allowed us to have multiple classification on all the products, so instead of having a n by 1 vector, the response would have been a n by m vector, where m is the number of products (24).

Further to the above, the idea of weighting the products is something we wanted to look into. The idea is that when a data point is represented in the n-dimensional hyper plane, the proximity of how close the data point is within a cluster will be given a weight, the closer it is to the centre the higher the weighting.

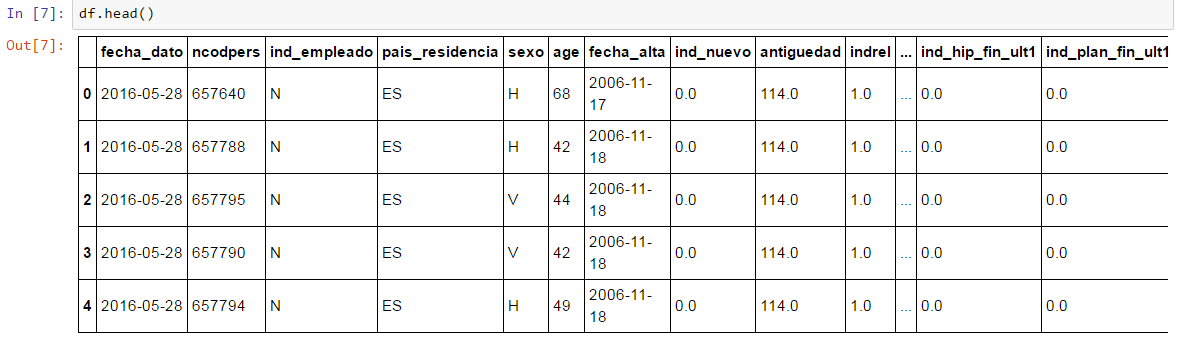
Another idea we had was to understand the historical relationship between purchasing patterns and the customers, we had this data but initially we ignored it, for simplicity and only used the most-up-date data. In future, we would like to look into these data to avoid recommending product that the customer cancelled in the past.

# Ranking factorization

The GraphLab toolkit provides a unified interface to train a variety of recommender models and use them to make recommendations.

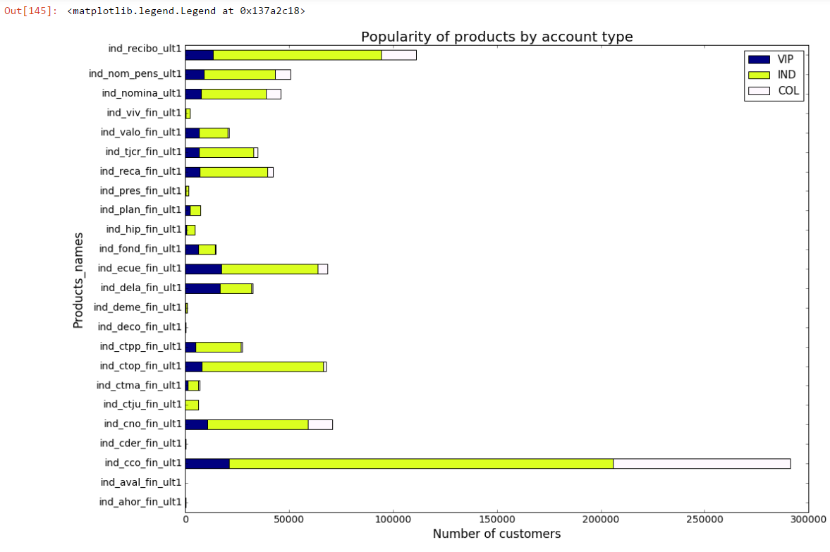
## 4.1 Method

1. Data ingested in CSV format and explored using python Dataframes as in figure1. Python was a suitable choice due to the large number of rows in the dataset, with Excel or other packages proving obsolete at handling such large data amounts.

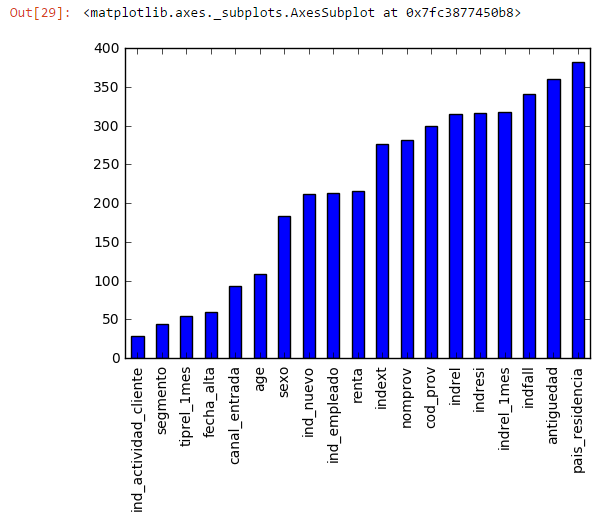


**Fig (9): A sample of the data ingested into pandas DataFrames**

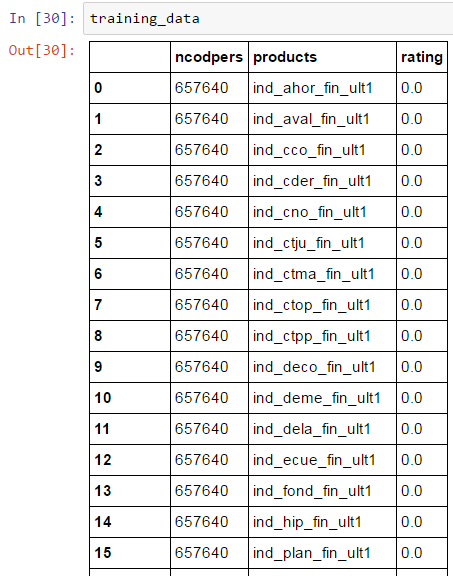
1. Data sets were cleaned and explored to find most popular products and the most important features for classifying customers.



**Fig (10): Graph representing the most popular products by account types VIP, individual and college (above). Graph representing scores of the most important features based on correlation (below).**

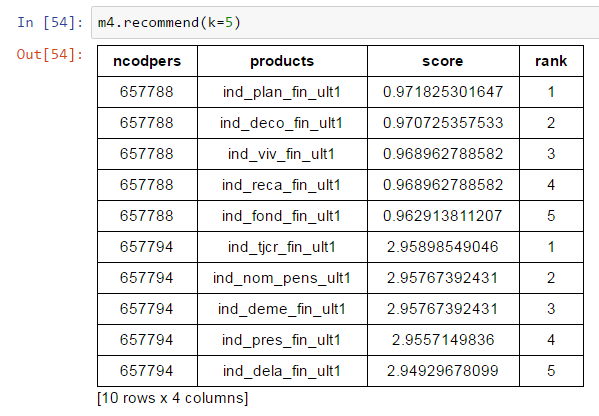


1. Data manipulation underwent in order to provide data into the right format for GraphLab algorithms. One of the main setbacks of the GraphLab interface was the stringent limits placed on the data formats required for the algorithms to work, meaning this step was the majority of the work for this model.



**Fig (11): A sample of the engineered data within DataFrames, to the required GraphLab standard.**

1. It was noticed that the products recommended were not specifically tailored to individual customers. This may have been because of the binary format of the ratings of the products, the large dataset and the number of zeros. To test the algorithms we built a recommendation engine on a much smaller dataset of 55 rows, with randomized ratings figures ranging from 1-5 using item-item similarity recommendation. This resulted is more personalized recommendations.



**Fig (12): A sample of the results provided by the ranking factorization model after tuning.**

1. This concludes the algorithms on GraphLab are not very effective with binary datasets.

## 4.2 Conclusion

GraphLab provides good package for machine learning and building recommendation engines, however when data takes a binary format it is not effective. The program is not very versatile with data structures, as the data had to be heavily manipulated for the algorithms to run. The results are not in the format where they are ready to submit to Kaggle. Although this was not one of the aims, as we believed it would not be possible to submit past deadline, the recommendation output from GraphLab could not be manipulated. This further shows lack of versatility of the GraphLab package. In the aim of understanding the python GraphLab package, this project proved to be a success, it was important to expand and look at new approaches in this project.

# Jaccard similarity collaborative filtering

The Jaccard index, also known as the Jaccard similarity coefficient is a [statistic](https://en.wikipedia.org/wiki/Statistic) used for comparing the [similarity](https://en.wikipedia.org/wiki/Similarity_measure) and [diversity](https://en.wikipedia.org/wiki/Diversity_index) of [sample](https://en.wikipedia.org/wiki/Sample_(statistics)) sets. The Jaccard coefficient measures similarity between finite sample sets, and is defined as the size of the [intersection](https://en.wikipedia.org/wiki/Intersection_(set_theory)) divided by the size of the [union](https://en.wikipedia.org/wiki/Union_(set_theory)) of the sample sets.

## 5.1 item-item collaborative filtering

Instead of finding customers that are similar, we tried find items (products) that are similar. Once we have an item similarity matrix, we can easily recommend alike items to customers who have purchased any item(s) from Santander. This algorithm is far less resource consuming than user-user collaborative filtering. Hence, for a new customer the algorithm takes far less time than user-user collaborate filtering, as we do not need all similarity scores between customers. In addition, with fixed number of products, product-product similarity matrix is fixed over time.

## 5.2 Algorithm

We used this algorithm because from research we decided that it was the best option, with regard to the structure of the data and the capability of our machines. Initially we used the Cosine Similarity function in the scipy library in Python. This is because many other recommendation engines we looked also used this function. After further research, we found that the Jaccard similarity function was best suited to our data because it is much better at relating Boolean rating like the purchase of a product. This justification for changing algorithm was reinforced by the score achieved using Root Mean Square Error as Jaccard reduced the error by approximately 50%.

**Jaccard Similarity:**

* + Similarity is based on the number of users which have rated item A and B divided by the number of users who have rated either A or B
  + It is typically used where we don’t have a numeric rating but just a Boolean value like a product being bought or an add being clicked

**Cosine Similarity:**

* + Similarity is the cosine of the angle between the 2 vectors of the item vectors of A and B
  + Closer the vectors, smaller will be the angle and larger the cosine

At first, attempting to load the whole dataset into our Python script caused the computer to crash. To remedy this issue we used only one month of data to make sure the engine actually worked. Then we used PySpark on our VM to create an RDD containing the last record for each customer with only his or her ID and Products that they do and do not have.

However, we had difficulty achieving a score from Kaggle at first because there were too many customers within the training set of data. We could not use the test set of data, as it was incompatible with the engine that we had built. Hence, we filtered our results against a list of customers required from the test set of data. This gave us our final results which we were able to submit.

We are happy to say that we successfully beat the benchmark score and overcame the issues that we encountered. However, we are still looking to improve the code as it takes approximately 9 hours to run. Although, this may be due to the large dataset, capability of our machines and the limited amount of knowledge about machine learning and algorithm optimisation between the two man team.

# Conclusion

This project returns a good insight to the number of methods of available these days to produce accurate prediction and recommendation systems. The main points to take away from here is that while the output is similar, a lot of thought has to go into what method and algorithm to go with, along with the business implications it has. One has to ask himself do the slight increases in accuracy of a model outweigh the time, effort and labour it takes to achieve such improvements? Maybe this project will help you decide the most efficient way.