**Santander Product Recommendation Engine**

Our aim was to build a recommendation engine using Item-Item Collaborative Filtering algorithm and to beat the benchmark score provided by Kaggle.

I**tem-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 don’t need all similarity scores between customers. And with fixed number of products, product-product similarity matrix is fixed over time.

We used this algorithm because from our research we found 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 couldn’t 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.