Scuola universitaria professionale
della Svizzera italiana

SUPSI

University of Applied Sciences and Arts of Southern Switzerland
Department of Innovative Technologies

Data challenge 3 - Project 1

CUSTOMER CHURN PREDICTION

Amos Colombo
amos.colombo@student.supsi.ch

Carlo Grigioni
carlo.grigioni@student.supsi.ch

Nathan Margni
nathan.margni@student.supsi.ch

Examiner: Sandra Mitrović
SUPSI, Lugano Switzerland

07/11/2022

Abstract

The purpose of this project is to build a model to predict the churn rate of a shop. The available data set contains orders of an online shop for the years 2010 and 2011. The initial data set was not suited to train a model on it, since a target feature characterizing churners needed to be defined and generated. Features were also relative to orders and not to customers. The approach adopted is to extract additional features, relative to the customers, from the ones of the orders. Time series analysis, Natural Language Processing and Graph analysis are the techniques used to generate features. Feature selection is then performed exploring filter, wrapper and embedded methods. Some different classification models where then implemented. Random Forest Classifier give us outstanding results.

Table of Contents

List of Figures

1. Description of the problem

In this section we explain and discuss our definition of churn and its implications.

1.1 Churn definition

The definition provided of churn provided by the Cambridge Dictionary is:

> *The number of customers who decide to stop using a service offered by one company and to use another company, usually because it offers a better service or price.[1]*

In our setting the task of defining which customers are churners is not trivial, since we are not provided with a clear label. After iterating on it our operative definition is the following:

> *A customer is a churner if, given a time t, looking at all transactions made in the previous 6 months, the customer did not appear in any of them. Otherwise, it is not a churner.*

The interval of 6 month is large enough to avoid that most seasonal customers are labeled as churners.

1.2 Churn ratio

The data set is unbalanced with 3506 customers who are not churners and 2373 churners. **Expand with data**

2. Data set

Data is from an online shop. It contains information about orders made from December 2009 to December 2011.

2.1 Data set structure

It is made of 2 tables:

- Year 2009-2010: 525461 orders from 01-12-2009 to 09-12-2010
- Year 2010-2011: 541909 orders from 10-12-2010 to 09-12-2011

2.1.1 Columns

- Invoice: unique code identifying every order
- StockCode: code identifying every product
- Description: text characterizing every product
- Quantity: number of items of a product in an order
- InvoiceDate: date and time of the order in the format "DD-MM-YY HH:MM"
- Price: total price of the order
- Customer ID: code identifying the customer who purchased the order
- Country: country from where the order was made

3. Pre-processing

We cleaned the dataset starting from an initial 1067371 rows and ending with 787490 rows therefore we removed 279881 rows which is just over 26% of the data. Let's now take a look at how we pre-processed the data in detail.

3.1 Cleaning the data

Dropping duplicates present in the dataset to ensure the orders are unique, most probably duplicates are the result of a system error which generates two times the same row. This is although our assumption that must be verified with the client.
We noticed there were circa 20k rows with None value for the customer, since our focus is on the costumers we cannot get any useful information from this rows therefore they were dropped.
We now have 787490 rows all non-null. During exploration of the dataset it came to our attention many rows did not reference actual purchases form customers. Some columns referenced adjustments warehouse movements ex... We decided to drop this rows as they cannot offer any additional insight in the customers behaviours. In order to do so we converted Stock code and Invoice into strings to more easily work on them, we then dropped all Stock codes containing the following strings "Adjust, M, D, Test".
Stock codes containing the letter C referred to cancelled orders so we kept them as they can prove very insightful in the prediction of churners. We created a dataset through a groupby based on customer ID with only the cancelled orders so we could compute features based on the cancelled orders only.
We then did a groupby based on customer ID with all the remaining data. This allows us to look at each customers information more easily.
Finally we merged the customer based datframes together in order to have all the information about users orders both cancelled and standard grouped together in an easy to access dataset.

4. Feature extraction

4.1 Time series analysis

For the time-series analysis we decided to use tsfel which is a library that given data with a temporal dimension computes various features. For the reference of all the difference features computed by the library look at the following link https://tsfel.readthedocs.io/en/latest/descriptions/feature_list.html. Tsfel analyses the dataset looking if the data is adequate (all features must be numbers, more than 1 row is necessary) then it runs some preprocessing of it own and finally it computes the feature extraction. In order for all this process to work we had to generate a dataset for each customer and then call the library to compute the features on the customer based on his purchases.

4.1.1 Preparing the dataset for time series analysis

We removed all the cancelled orders, dropped the Stock Code, description and country as mentioned before this was necessary because tsfel can only work with numbers. For the same reason the invoice date was converted into a numeric timestamp. So the final extraction was done on the following features: Quantity, invoice date, price, customer ID and total price.

4.1.2 Creating the customer datasets

Now all we have to do is divide the dataset by customer ID and create a new dataset for each unique customer, which will contain all their orders. To do this we create a list containing each unique customer ID. We then create a dictionary containing the dataframe relative to the customer and check if the customer has made more than one order. If the customer has made less than one order he will be removed as we cannot compute time-series features in this case. We will then compute the features with tsfel *time_series_features_extractor* function and save the dataset as a file.

4.1.3 Merging all the datasets

We now have a dataset for each customer with the time-series features computed, only thing remaining to do is merge them together.

4.1.4 Some cleaning

tsfel generates a lot of features coming from many domains, naturally so some of this features are not relevant in our case. In order to lighten the dataset and improve computing speed for the next processes we do a small data-cleaning removing all the features which have more than 80% null values.

4.2 Natural Language Processing

4.2.1 Initial data

The Description column contain a text describing the product in few words, it have a lot of unique descriptions (as the stockcodes) therefore high cardinality, we also doesn't have any information about the similarity between products, for that this data is not very suitable for our final model and therefore we want to extract more functional and compact information.

4.2.2 Tokenizing text

Before doing any manipulation on the text we had to tokenize it, meaning that we splitted the text in smaller units (words) removing some irrelevant and potentially confounding terms for the NLP model that we want to create e.g. colors, materials, quantities; those terms would aggregate very different products as if they are similar for example by only having the same color and we don't want this to happen. The output of this process is finally a list of cleaned words for every description.

4.2.3 Pre-trained embedding model

To create clusters on the description we have to find an embedding vector for each word, to do it we firstly trained a word2vec model from our descriptions to estimate about a word's meaning based on their occurrences in the text, but unfortunately our text data was insufficent and the clustering bad. We then changed strategy by using a pre-trained model, "word2vec-google-news-300" [2], with already 3 billion running words in the embedding, this model return a much more precise 300 dimensions vector about the position in the embedding for a word (the neighboors of a word are the synonyms) then for each description we computed the mean of this vector from each word in it, to finally have a 300 dimensions vector with numerical variables for each description that estimate the position of every product in the embedding.

4.2.4 Clustering

We computed then the cosine similarity matrix from those vectors (clusters) because should be better with text clustering, then we computed the PCA with 50 components (from the 5000 columns of the cosine matrix) and trained a kmeans model, here we had some troubles in finding the best number of clusters, this because generally we have to pick the k value when the cost stop decreasing the most and the Silhouette stop increasing the most, and as we can see from the below graph those points are at a very high k value (about 1000 or more), those high values would not reduce that much the high cardinality of the products and therefore we choose a middle ground value between a very low k value (better for simplicity and cardinality reduction) and a high k value (best inertia cost and Silhouette score), the chosen number of clusters was then 500. We have to say that even with that many clusters the Silhouette score is about 0.25 that is generally a bad score (max is 1) this because there are few words on the descriptions to work with.

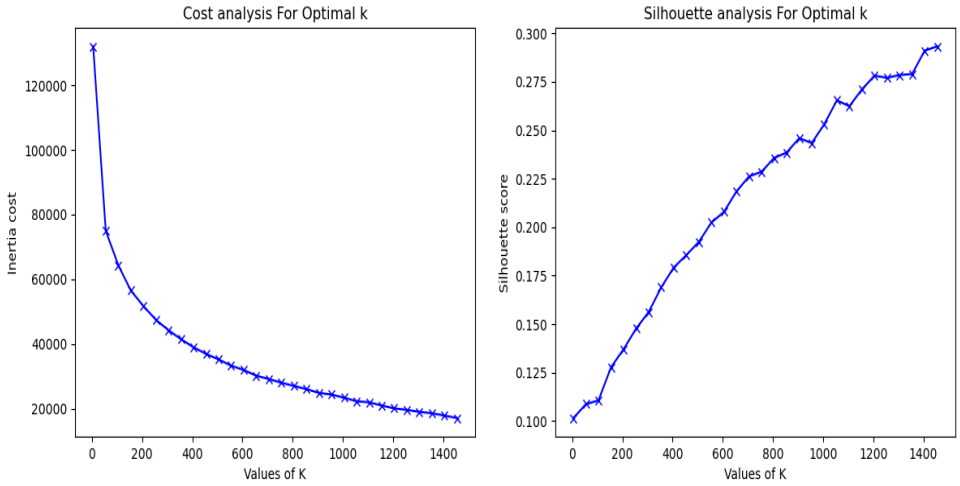

Figure 1: Cost and silhouette analysis

4.2.5 Results

Finally we have a category and a vector for each product that we use as feature by computing some aggregations:

- Count of the purchases by category (cluster) for each customer
- min, max, mean of all the vector purchases for each customer

We also performed a PCA reduction in two dimensions to visualize the products in a two dimensional space both for vectors and cosine similarity (without cluster hue because would be very confusing).

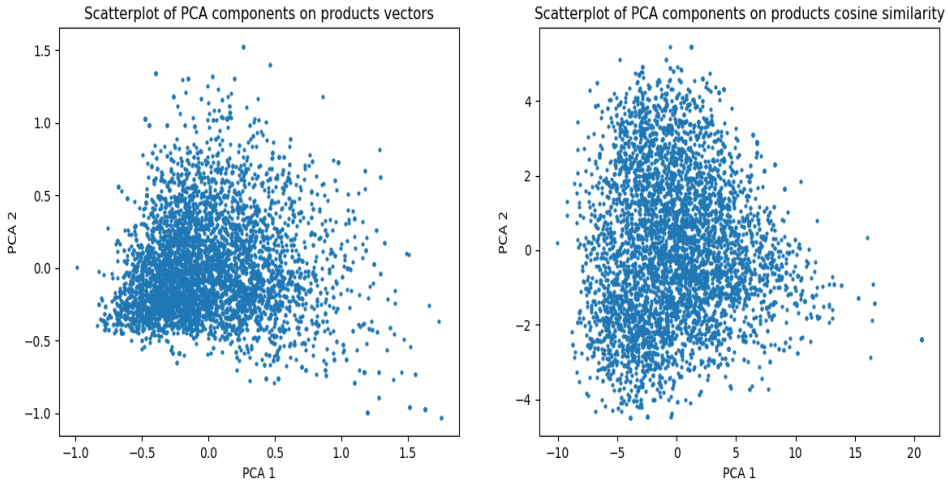

Figure 2: PCA components on products

We can see that on the cosine pca plot the points are a little less dense, but we don't have very isolated groups in both cases making difficult to separate in clusters, but of course we use more than 2 PCA components to train the model and it manage to create clusters in some way.

4.3 Graphs

Graph analysis is an approach used to extract information from data made of linked entities. In this data set products are related between them by customers who bought two products and customers are linked by the common products they bought.

4.3.1 Graph structure

We built the graph of products, which has as nodes the products. They are linked if there is a customer who bought both products. Edges are weighted by counting how many customers bought both products.

4.3.2 Data leakage risks

Extracting features from graphs is a process where data leakage can occur, if not handled correctly. Data leakage is essentially the introduction of information about the target, which should not be legitimately available in the training set[3]. In our case, the risk arises when, by using the original data set containing every order to generate a graph, and then extracting features from it, our test data gets contaminated from nodes and edges which belong to test data.

Solution We proceeded to perform a train-test split on customer IDs, with a test size of 0.20, then generated the train set and the test set by taking every order relative to that customer. We then extract graph features for the training test, using only the orders contained in the training set. For the training set instead, the features are computed using the whole graph, since data from the train set is always legitimately available.

4.3.3 Centrality measures

The features we were interested in extracting were related to how central the products bought by a customer were in the product graph. For this purpose we used some centrality measures:

Degree centrality It computes the number of links a node has with its neighbors. It relies on the idea a not referenced by many nodes is more important. It does not take in account weight. We used the implementation of Degree Centrality in the NetworkX library.

Betweenness centrality It computes how many shortest paths are there between nodes u, w in the graph which contain node v. We used the implementation of Betweenness Centrality in the NetworkX library.

Closeness centrality It computes the length of the shortest path between node v and any other node in the graph. More important nodes have shorter paths to reach the rest of the network. We used the implementation of Closeness centrality in the NetworkX library.

Eigenvector centrality Starting from the idea behind degree centrality it also takes into account the importance of each node, by recursively updating the importance value and including it in the importance computation of neighbor nodes, assuming a node with important neighbors will be more important than a node with less important neighbors. We used the implementation of Eigenvector centrality in the NetworkX library.

PageRank It is a variant of Eigenvector centrality, originally created as an algorithm to rank web pages by Google. It works by counting the number and quality of links to a page, in order to determine a rough estimate of how important the website is [4]. The same approach can be applied to any entity in a graph, looking at its links with neighbor nodes, their importance, and the weight of the edges. In particular, when running PageRank, a Random Surfer is simulated. We used the implementation of PageRank in the NetworkX library.[5]

4.3.4 Extracted features

After computing centrality measures we had importance values for each product. We then needed to transform those values to features relative to costumers. In order to do this for each customer and for each centrality measure we computed the mean of the node importance for the list of products a customer had bought.

5. Feature selection

During the feature extraction phase over 2000 features were generated. To make models less computationally expensive and easier to understand we performed feature selection and noticed some important difference in the relevance of features.

Additional pre-processing Before performing feature selection we needed to drop not numerical features. Descriptions, cluster descriptions and dates had not usable data types.

5.1 Filter methods

We used filter methods to rank features by relevance in predicting the target. These methods are fast, which is important when working with such high dimensional data. Methods we used include chi-square and mutual information. Other common methods such as ANOVA (Analysis of Variance) did not fit our case since we are performing binary classification.

5.1.1 Chi-square

Based on the χ^2 statistical test, it is a uni-variate filter that measures divergence from the hypothetical distribution where the feature is independent from the target value. It requires normalization of the features. We used the implementation from scikit-learn. According to this test the best features were some of the ones extracted by Time Series analysis.

5.1.2 Mutual information

It is a common uni-variate method, it measures the decrease in entropy considering a single feature at a time. The higher the gain, the more important the feature. Its drawback is that it does not consider redundant features, since working independently on each feature. We used the implementation from scikit-learn.

5.1.3 Fisher Score

It ranks better features with similar values in their instances of the same class and different values to instances from different classes. We used the implementation from skfeature-chappers.

5.2 Wrapper methods

Wrapper methods were not our first choice for feature selection, since they are generally slow. They use a classification algorithm as a black box evaluator to he best subsets of features.

5.2.1 Recursive Feature Elimination

It is a multivariate method, since it considers all features together. It trains the selected model (Logistic Regression) recursively and removes the least important feature, according to the model. We expected the method to be computationally inefficient. It required much time due to the high number of features generated during feature extraction. We used the implementation from scikit-learn.

5.3 Embedded methods

Embedded methods were a good choice for feature selection, since they can model feature dependencies and select subsets of features, without iterating like wrapper methods.

Additional pre-processing Before using these methods some pre-processing was needed. Most features were of type float64, which is too large for the scikit-learn implementations of the feature selection methods. We updated their type to float32. Some infinity values were also present. We decided to set them to the maximum finite value.

5.3.1 LASSO

Least Absolute Shrinkage and Selection Operator (LASSO) is a shrinkage method, often used both for regularization and feature selection. It works by trying to minimize a cost function, thus automatically selecting the most relevant features. We used the implementation from scikit-learn.

5.3.2 Embedded Random Forest

It works by measuring how each feature decrease the impurity of the split in decision trees. The average over all trees in the forest is the measure of the feature importance. This process naturally happens when building the model. We used the implementation from scikit-learn.

6. Dimensionality reduction

6.1 Dimesionality reduction methods

We used two dimensionality reduction unsupervised methods to visualize the position of the costumers in 2 dimensions colored by the target value "churn" to visually see how well the features variance in two dimensions separates the label. We used the classic method PCA then we also tried the TruncatedSVD method that should work better with sparse data, we applied the methods on the product vectors and graph/timeseries features separately because we saw that visualizing all the features together was separating badly the label.

6.1.1 Reduced components from products features

Figure 3: Reduced components for products

We can see that the products vectors features didn't separate very well the label thought there are some more red regions (customer that churned) meaning that those features will still bring some information on the target.

6.1.2 Reduced components from graph/timeseries features

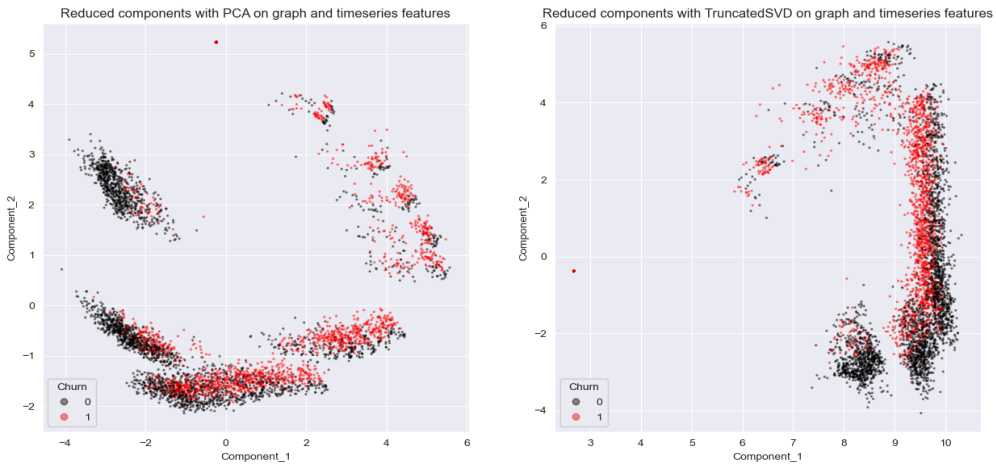

Figure 4: Reduced components for graph and timeseries features

On the others timeseries/graph features the two components create more distant and isolate groups of customers that separate better the label (some groups have mainly black or red points) in other mixed groups a little alteration within the group also separate well the label Therefore we can expect that on the model the graph and timeseries features will be more important than the the products ones on predicting the label.

7. Results

We decided to train various models and see the results, and the results obtained are really promising. We decided to use the feature selection of lasso and manually removed the time since last purchase (this was done due to the high correlation with the target feature "churn"), now let's see the models and their results. Lasso kept 354 features which is a lot of features but taking into account we had 2741 features.

7.1 K neighbors classifier

The resulting f1 score with K neighbors classifier was 0.838 which is a really good result and much better compared to when we gave the raw dataset, resulting in an f1 score of 0.668

7.2 Logistic Regression

The resulting f1 score is 0 since the model is just randomly guessing which is tied as the lowest f1 score amongst our models.

7.3 Tree decision classifier and Random forest

This two models gave very similar results and gave the best results, for decision tree 0.982 f1 score and random forest 0.99. A very big improvement from the raw dataset giving 0.634 with decision tree and 0.703 f1 score with random forest.

7.4 Support vector machine

The resulting f1 score is of 0.668 which is not a big improvement from the raw dataset, f1 score of 0.562

7.5 Neural network

As a neural network we used the sklearn MLPClassifier resulting in an f1 score of 0 meaning the model is randomly guessing. Building and testing various structure of a dedicated neural network might have brought better results, a thing to test in the future.

8. Critical reflection

This project gave us a better idea about feature extraction, its opportunities and challenges. Facing a big data set with features we were not able to train a model with, required us to explore different approaches. The need to define and generate our target feature ourselves was a topic of discussion and we reached a solution after some trails and iterations. Feature selection was then important to understand which of the feature extraction approaches we tried was the most useful.

8.1 Problems

A problem we found in almost every phase of the project was computational expensiveness of methods, which combined with the large amount of data, made computations time consuming. In particular time-series, graph generation and wrapper methods for feature selection were particularly taxing on this point.
Data leakage was a risk we had to deal with, as explained in a more detailed way in section 4.3.2 and as discussed in class with the lecturer, when generating graphs. Since it is such a subtle risk, it was important to see it in practice.

8.2 What could have been done

Feature extraction from graphs When extracting features from graphs we decided to generate a graph with products as nodes and customer orders as edges. The possibility of generating another graph with customers as nodes and common product orders as edges was available too. We decided to not take this path given the computational expensiveness of this process.

Models Given the requirements of the project, that did not expected models to be the most important topic and since we already obtained good results, we therefore missed the opportunity to improve them with hyper-parameter tuning could be interesting. Also a neural network model could have been built trying out different structures and we think some very good results could come out of it.

9. Personal conclusions

Data set The data set presented many challenges: its large size, which made not making mistakes function critical due to long computational times, the presence of many out of scope data (adjustment and post invoices were not real purchases from customers). Working with subsets of data and exploring the data set were essential to be able to work on the task.

Feature extraction Extracting features was the hardest part of the project. Time Series analysis is an example, given the problems we faced with libraries. However seeing they helped greatly improve the accuracy of the models was exciting and rewarding. Graph generation was also intriguing at the level of data manipulation, exploring ways to shape data correctly with reasonable time complexity. The power of centrality measures made us think about other implementations of this tool. It was also a good opportunity to see the data leakage problem in practice. The NLP part was also difficult because we the text was very short and many confounding words were present (adjectives, words with more meanings, overused words etc.) and unfortunately was not much relevant on our target, but it was still interesting to work with text and clustering with this data.

Team work We are happy with our task split, since everyone had something to work on in a balanced way. We also have complimentary interests, so everyone ended up working most of the time on the topic he was more interested in.

References

[1] "Cambridge dictionary," *Cambridge University Press*, 2022.

[2] "How to cluster documents using word2vec and k-means." (), [Online]. Available: https://dylancastillo.co/nlp-snippets-cluster-documents-using-word2vec/.

[3] S. Kaufman, S. Rosset, and C. Perlich, "Leakage in data mining: Formulation, detection, and avoidance," vol. 6, Jan. 2011, pp. 556–563. DOI: 10.1145/2020408.2020496.

[4] "How google search works," *Google*, 2011.

[5] NetworkX. "Pagerank." (), [Online]. Available: https://networkx.org/documentation/stable/reference/algorithms/generated/networkx.algorithms.link_analysis.pagerank_alg.pagerank.html.