HelloFresh Data Science Project

Carlo Contaldi

contaldicarlo@gmail.com

www.linkedin.com/in/carlocontaldi/

1. Frame the Problem

I am provided with a choice over two tasks:

- predict the customer churn over time;
- predict the weekly demand for each product type.

I would address both tasks but, in order to maximize business efficiency, I still need to prioritize them and get relevant results on the most important one before tackling the other one. In absence of any prior reason that assigns different priorities to the tasks, I would choose the task based on the usefulness and impact of related results on the business strategy, and the related amount of innovation and pioneering work.

Based on Dr. McCarthy's analysis at "https://www.linkedin.com/pulse/hellofresh-has-bigger-customer-retention-problem-than-daniel-mccarthy" and assuming that such analysis is still valid at the current state of the business, HelloFresh's customers churn out more quickly than its main competitor (Blue Apron). I believe that, given HelloFresh's customer base, characterized by relatively faster expansion (better customer acquisition rate) and a relatively lower retention rate, it's more important to derive insights useful to retain customers on the long run rather than better understand the demand of the current, rapidly changing customer base.

For the sake of the project, I will address the prioritary task only, i.e. customer churn forecasting. The task is formulated in an open and simple way: it is unconstrained by a minimum target performance over some given control metric or any other satisficing metric. I consequently assume that my Statement of Work consists in the following:

- define a suitable target and control metric;
- identify and leverage as many as possible data patterns;
- model and evaluate a representative set of pipelines with the purpose of performance maximization.

1.1. Target and Control Metric Definition

Let us notice that the specification does not define the KPI ”customer churn over time”, nor does explicitly include the related column in the dataset: I need to formalize the target and extract it from the supplied dataset. By analyzing HelloFresh’s financial reports (in particular the ”Performance Measurement System” chapter in the ”Consolidated financial statement for the year ended 2017” at ”http://ir.hellofreshgroup.com/websites/hellofresh/English /3100/annual-report.html”), I noticed that the number of Active Customers is defined as the ”number of uniquely identified customers who at any given time have received at least one box within the preceding 3 months”. As a consequence, I will define the ”churn rate at time t” as $(n_{t-1} - n_t)/n_{t-1}$, where n_{t-1} is the number of active customers a churn period before t and n_t is the number of active customers included in n_{t-1} that are still active at time t. So as to start with a simplified problem, I will initially consider a `churn_period` of one month, then, in case I get significant results, I will evaluate my model with a `churn_period` of three months. The prediction problem can be reformulated as a binary classification task that, given a set of events or characteristics associated with a customer, outputs whether the customer will churn by the subsequent `churn_period` or not. As a control metric, I would adopt accuracy if the problem is balanced, otherwise the F1 score or the ROC-AUC.

2. Programming Environment and Execution Procedure

I used Conda 4.5.11 with Python 3.6.5 (latest versions) on a machine implementing Windows 10 64-bit to set up my PyData stack. I worked within a new environment defined and activated via terminal as in the following:

```
> conda create --name hellofresh python=3.6.5 numpy pandas matplotlib seaborn
  scikit-learn py-xgboost
> conda activate hellofresh
```

The main.py script executes all the code I wrote and saves the output in a logfile.

```
> cd project_folder
> python main.py
```

3. Data Science Workflow

Data cleaning and conditioning occurred throughout Exploratory Data Analysis and Feature Engineering phases.

3.1. Data Retrieval

I imported each .csv file as a Pandas DataFrame, and checked the import for correctness with `head()`, as coded in `import.py`.

3.2. Exploratory Data Analysis

All the work of this step is coded in `eda.py` and `visualize.py`.

As indicated in the given readme, I will assume that provided data is the only available data (no external data allowed).

3.2.1. Boxes

The boxes dataset contains more than 4 million entries.

In regard to missing values, I observed that only 5 entries have a missing value on the started_week field. I retrieved such entries and noticed that they have the same subscription_id. I retrieved all entries having that specific subscription_id and noticed that there are entries with a filled started_week field. If the started week of a subscription remains constant through time, then I can safely fill the above-mentioned missing values with the date indicated in the other records. I tested the immutability of the started_week field with respect to subscription_id and proceeded with missing values imputation. A preprocessing step can be set up in the production pipeline so as to take care of analogous missing value problems with new data.

Each entry is characterized by a unique box_id. I assessed the uniqueness and non-nullity of such field. Each entry is also characterized by a subscription_id, which identifies a customer. Given that there are slightly more than 1 million of unique subscription ids, I observed that 4 boxes on average have been delivered to each subscriber.

The started_week field is stored as a string of the type yyyy-Wxx, where yyyy represents the year and xx the week number. I don't have any information useful to convert such string into a date. If it was possible, I would search for this information through suitable sources. In this case I will attempt to infer a conversion scheme. By observing the unique values of such field, I noticed that years are in the range [2012,2017], whereas weeks are in the range [1,52]. Given that a year contains 52 weeks plus one day (addressed with the leap year), I can assume that the week number is 1-indexed (e.g.: W02 is the second week of the year). I converted the string into a datetime64 value, by addressing the conversion to a 0-indexed week encoding and by setting monday as the day of each started_week.

Update: I finally noticed that such week encoding is explicitly used in HelloFresh URLs such as "https://www.hellofresh.com/menus/2018-W40/classic-menu". I can derive the correct conversion strategy by matching such weeks with the timedelta reported in the pages themselves. First of all, I observed that week numbers range in [1,53], and that each week starts on Saturday. I finally implemented, tested and applied the correct conversion strategy in the dataset. Here are reported some tests I considered to check for correctness: ['2018-W01', '2018-W02, '2017-W53', '2018-W53', '2016-W53', '2017-W01', '2016-W01', '2015-W53', '2014-W49', '2015-W31', '2016-W07']. I noticed that week 53 coincides with week 1 of the following year, excepting for the pair (2015,2016), where such weeks are distinct and consecutive. By considering this adjustment, I finally obtained a formula able to correctly convert each HelloFresh week in the dataset into a date that is the starting day of such week.

I assume that the started_week of any subscription should precede or be equal to the delivery dates associated with the subscription, because each customer needs to be subscribed in order to get a delivery. I tested the consistency of my reasoning by assessing whether started_week values precede or are equal to delivery dates: I observed that there are cases in which this is not true. I analyzed in-depth such anomalies and observed that: their count is around 1000, i.e. one over 5000 the overall number of entries; the anomalous timedelta reaches a maximum of 110 days but a maximum of 5 days up to the third quartile.

Such a consistent and variate anomaly makes me wonder on the process that is responsible of filling the `started_week` field. I would investigate more about this so as to decide whether we can leverage this predictor. In absence of further information, I can conclude that a timedelta up to a week may be related to some mechanism involved with the filling of this field I am not aware of, and that a deviation higher than one week entails a problematic entry, that may hinder model training and introduce too much noise. I finally removed the so-defined problematic entries: I believe that by doing so I would get a data series that possibly holds useful information and predictive power, even if not theoretically consistent.

3.2.2. Pauses

The `pauses` dataset contains almost 6 millions entries. It has no missing values.

By analyzing the timedelta between `pause_start` and `pause_end`, I observed that it is generally around one week, and that there are 26 anomalous entries having `pause_start` not occurring before `pause_end`. By observing such entries I noticed that they have the same values for `pause_start` and `pause_end`. By assuming an error on the year value of `pause_start` or `pause_end` and by opportunely adjusting it, the resulting entries would present the same pattern as the others. Still, even if I would be able to impute the correct dates, I decided to drop these entries because I don't know a way to generally address such a problem at production time, with new data.

3.2.3. Cancels

The `cancels` dataset contains around 1 million and a half entries. It has no missing values.

3.2.4. Errors

The `errors` dataset contains more than 400 thousands entries. It has no missing values. I renamed the field `hellofresh_week_where_error_happened` as `week`. By observing the distinct values assumed by `week` I noticed the presence of the erroneous value `0000-W00`. Given that there is a single error entry presenting such value and that there is no way to recover such information, I dropped it.

Analogously to the `boxes` dataset, I observed some discrepancies in the timedelta `[week, reported_date]`, where week represents the starting day of the week in which the error occurred. This timedelta should not be negative because an error can be reported only while or after experiencing it, but a great deal of entries in the dataset (more than 90 thousands, i.e. more than 1/5 records) describes the opposite pattern. The anomalous timedelta reaches a maximum of 727 days but a maximum of 2 days up to the third quartile. I applied the same conclusions and choices I made for the `boxes` dataset to this case as well.

In regard to compensation types, I noticed that possible values include `full_refund` and `full refund`: I fixed this trivial discrepancy by redefining `full refund` as `full_refund`. At this point, I can still observe three different refund compensation types, i.e. `full_refund`, `partial_refund` and `refund`. By observing the generic statistics of entries characterized by these fields, the `refund` type looks like an exception rather than a rule: there are only around 2500 entries over more than 400 thousand entries, i.e. one over 170. I need to understand whether such refund type can be aggregated to the full_refund or partial_refund series, so as to reduce the number of possible categorical values (especially with a rare value such this one). If refund entries present a similar enough distribution with respect to one of the others, I would incorporate those entries in that group. In order to evaluate this, I plotted the compensation amount densities in terms of refund types with Kernel Density Estimation. I can notice that partial_refund can be approximated to a univariate gaussian distribution centered in 30, whereas full_refund looks like a very steep and defined bivariate distribution centered in 60 and 70. The 4 spikes in partial_refund make me think that most of partial refunds involve amounts nearby 10, 20, 35 and 50, whereas almost all of full refunds involve quantities higher than 55. Unfortunately, the refund set presents a bivariate distribution centered in 35 and 70: I can conclude that entries collected as refund probably involve both full_refunds and partial_refunds. Assuming that refund entries are too few to be considered as a separate class, I will distribute the related samples to full_refund and partial_refund by discriminating with respect to an amount threshold: entries having an amount lower than 55 will be assigned to the partial_refund group, whereas the remainder will be assigned to the full_refund class. I recognize that I used the whole dataset to derive the 55 threshold: I should instead include this preprocessing step in the model pipeline and operate on the training data only.

I also noticed the presence of 3 entries having the sorry compensation type. I assume that the count is too low to consider relevant such categorization or the samples themselves: I consequently dropped these entries.

All dates have been converted from strings into datetime64 values. The delivery_date ranges from 2013-01-07 to 2017-12-01, whereas the started_week ranges from 2012-08-27 to 2017-11-27. It seems that the boxes dataset has been acquired by extracting all deliveries occurred in a predefined time range, then the other datasets have been built by considering all related entries matching with all distinct subscription_ids in the boxes dataset. This would also justify why the boxes.started_week value can be outside of the sampled date range. I assessed this assumption by evaluating whether all subscription_ids in pauses, cancels and errors are a subset of the subscription_ids in boxes. This resulted to be true at the beginning of the cleaning process, thus I will consider boxes as the reference dataset. Given that after I dropped some entries from boxes this is not true anymore, I removed from the other datasets entries not having a match with boxes on subscription_id. On the other hand, delivery occurrences may happen at different dates with respect to other events (pause starts, cancellations, etc.): as a consequence, the index of the final DataFrame will encompass all relevant dates in the datasets, i.e. boxes.delivery_date, pauses.pause_start, pauses.pause_end, cancels.event_date, errors.week. I plotted the Event Density Over Time, and noticed that the majority of events occurred from 2015 till the middle of 2017, at an increasing rate. It's also important to take into account the extremes of the distribution: the way the dataset has been acquired could have removed partial information on the extreme ranges, in a way that the remaining information does not describe anymore the process that generated it. For instance, let us think again that box deliveries are in a restricted range with respect to the other dataset time ranges: training our model over samples related to the extreme ranges, that involve all possible events excepting for deliveries, would just contaminate and confuse the model. Based on this assumption, since I observed very sporadic events on the most remote extreme, I removed the first percentile of the entries, based on their date. The most recent extreme, on the other hand, should be truncated up to the last delivery date in the `boxes` dataset, because churn labels assigned based on events occurring at posterior dates would not be significant. In particular, events outside of the range `[2014-10-20, 2017-12-01]` have been removed from the dataset.

I plotted the distribution histograms of relevant categorical predictors: product types, channels, cancellation event types, error compensation types. In particular, I considered channels as properties characterizing customers, and thus I counted them by removing all duplicates in the `boxes` dataset (where multiple entries may be related to a single subscriber). We can notice that:

- the majority of products are of type 1 or 2, but other types are present in the dataset in a non-negligible way;
- `channel16` is reported in 37% of the totality of subscribers, whereas `channel15` involves 26% of the entries; the remaining 34 channels are more or less distributed across the remainder of the population (37%);
- most of event types in `cancels` are `cancellation` events (87%);
- most of errors are solved with a credit compensation (65%); most of the remainder (29% of the population) is addressed with a full refund.

I intend to convert all these categorical features in a one-hot encoding fashion. Based on the general imbalance of such feature distributions, it is likely that less represented one-hot encoded values will not have enough predictive power to be significant for training our system. As a further consideration, I can say that preprocessing the dataset with some dimensionality reduction technique would probably have a positive impact on the final results.

3.3. In-depth Problem Formulation & Feature Engineering

At this point I can confirm my preliminary thoughts about problem formulation: this can be seen as a binary classification task where our model, provided with an event associated with a customer, outputs whether the customer will churn by some subsequent timedelta or not. I initially considered a timedelta of three months, but I will formalize my experiments by providing the `churn_period` as a parameter, so that I can easily apply different timedeltas (such as 1 month). In order to tackle such task I first need to build a proper dataset by using the given data. All work of this step is coded in `extract.py` and `visualize.py`.

The focus is on the customer: each sample will consist of an event or event set associated with a specific `subscription_id` and timestamp. Based on the considerations of section 1.1, I define a ”churned customer at time t” as a customer that is not active at time t, i.e.

that did not receive any box in the timedelta `[t-churn_period,t]`. The label of a sample at time t will then be a binary value that describes whether the customer identified by the `subscription_id` of the sample is not an active user at the time defined as `t+churn_period`. I thought about considering reactivations or unpauses as events associated with an active user, but HelloFresh flexible subscription model allows to continuously pause the subscription: it is reasonable to consider a user who behaves like this for a long time as a churned user anyways. The box delivery remains the only robust indicator of an active user, given the available data.

I will be able to derive a churn value only for events happening up to a `churn_period` before the end of the date range of available data: we can't use events in the last `churn_period` for training or testing, but only to assign labels to the previous events.

The final DataFrame will be indexed by `date` and `subscription_id`, where an entry describes a customer that is going or not going to churn based on a particular event or event set among deliveries, pause_starts, pause_ends, cancellations, reactivations, errors.

I have a choice on the temporal granularity of the final dataset:

- Consider a single event per entry. We may have more events happening within a single day, and we should possibly consider a model able to capture the temporal relationships across ordered simple events, in relation with the target sequence. The training and testing samples should be the sequences of events associated with each customer.

- Consider an aggregation of events per entry, i.e. the events that happened in some specific timedelta for some specific customer. An entry alone possibly has enough information to discriminate over the answer, so in this case we would be able to use a memory-less model.

Given that I have not enough time available and experience in implementing temporal models (and I would probably need more time to work on it also for this reason), I chose to proceed by aggregating events per customer over predefined time ranges and using simpler models. The temporal aggregation introduces the granularity parameter: it defines the timedelta which a sample refers to, as well as the time discretization strategy to apply to our dataset. The granularity can reasonably vary from a week to the `churn_period`. Given the granularity definition, the churn label of a sample associated with a timedelta starting at t will then be a binary value that describes whether the customer identified by the `subscription_id` of the sample is not an active user at any time within the period `[t+granularity,t+granularity+churn_period]`. I will refer to a generic timedelta having length equal to a `granularity` period as `deltaG`.

The `boxes.box_id` feature does not contain any useful information for the task at hand: it will be dropped. Given the relatively high number of customers as compared to deliveries (4 boxes per customer on average), I won't consider the `subscription_id` as a feature, but only to join information for the final dataset. The `errors.reported_date` theoretically has no informative power for the task at hand: it is the customer himself that provides us with the error temporal location, through the week field. The reported_date field will be dropped.

3.3.1. Aggregation strategies

- The number of boxes received by a customer in deltaG can be considered as our first, relevant numerical feature.
- We can consider the timedelta [started_week, deltaG.start] and the channel as further, customer-related features.
- I would like to aggregate the information on the products (type of received boxes). I can think of two strategies:
  - redefining the above-mentioned number of boxes as an array of numbers, one per type;
  - considering the mode among the boxes delivered in deltaG to the customer.

  The second strategy is preferable because it does not introduce possibly sparse, lowly significant arrays, but may introduce bias if the mode is not really representative of the set. I evaluated this aspect in this way: I grouped boxes entries by subscription_id, then I calculated the number of boxes of the most frequent type divided by the total number of boxes, for each customer. Based on the resulting statistics as well as the Kernel Density Estimation plot of this quantity over the boxes dataset, I can conclude that associating each customer with its favorite product type introduces a negligible amount of bias in the aggregated data.
- Entries will be provided with the paused binary field: if a pause period intersects deltaG, then the field is set.
- If a cancels event happens within deltaG, then the related cancelled or reactivated one-hot field is set.
- The errors.compensation_type field identifies a categorical value that can be one-hot encoded. Multiple errors of different types in the same deltaG will result in activating multiple bits in the encoding.
- I will include the compensation_amount as a single, numerical feature. It is defined as the sum of the compensation amounts of the errors experienced in each deltaG by each customer.

3.3.2. Implementation

I have been able to devise an efficient enough way to acquire event occurrences (pauses, cancellations, reactivations, errors) for all deltaGs and customers only after some attempts. I first implemented two too slow methods, where the fastest one took 6 hours to process just one tenth of the pauses dataset. My third implementation finally ensured a fast acquisition (in the order of minutes). In regard to churn labeling, I started by setting all entries as `churned`. For each `deltaG`, I unset the `churned` values of all its entries matching with those entries temporally located by the next `churn_period` and characterized by a positive `n_boxes` value. For instance, with a granularity and `churn_period` of 1 month, the entries that are set as not `churned` at month x involve the customers who had at least one delivery during month $x+1$. As a consequence, all entries in the last `churn_period` result as churned and can't be used for training or testing.

3.3.3. Last Touches and Considerations on the Final Dataset

An important prerequisite for a dataset to be used for churn prediction is an homogenous churn/no-churn distribution over time with respect to `granularity` and `churn_period`. First of all, I removed from the dataframe all entries that have been used only as references for churn labeling, i.e. entries temporally located in the last `churn_period` (i.e. starting from date `2017-11-13` with the baseline values I applied for `churn_period` and granularity). I plotted the churn rate over time as a stacked histogram with bins large as the chosen granularity, before and after truncation. In my baseline case, by considering a `granularity` and a `churn_period` of 28 days, I can observe that the task is balanced on average and over local time ranges as well. Given such a balanced classification task, I will evaluate my models based on the accuracy metric.

3.4. Train/Dev/Test Splitting

Given our large dataset size (more than 3 million entries) and assuming an homogeneous label balancing, I am confident that a 99/1 split between training+validation set and hold-out test set is a fair choice. Moreover, I need to evaluate the model in view of its productionalization. I must assume that all phenomenons that ultimately impact on the churn rate vary over time: what is needed here is a model that is able to keep up with them; possibly, a model implementing online learning. In order to simulate its execution at production time, I will consider as a test set the last percentile of the entries in terms of dates; I will assume all previous entries as labeled and available to us, ready for the training/validation loop, possibly tackled with a k-fold cross-validation with grid search; the test set will be touched by our system only at the end, for the final and actual performance evaluation. With a `churn_period` and a granularity of one month and a 99/1 train-test split I obtained a churn rate of 0.59 in the training set and of 0.64 in the test set.

3.5. Preprocessing

Before any preprocessing step, I should assess the presence of multicollinearity so as to possibly remove redundant columns. The multicollinearity phenomenon may affect the final performance based on the task type and the used model, but it should be assessed regardless of its direct impact: reducing the number of features leaves unvaried or possibly improves the performance, if we can assume that excluded features contain redundant information. I will add this step to the pipeline after a first, simplistic baseline evaluation. I one-hot encoded the two categorical features `channel` and `box_type` and respectively obtained 26+6 additional columns in their place. Given the sparsity and the large size of the resulting matrix, the preprocessor converted the output into a sparse matrix. I need to normalize over numerical features, i.e. `n_boxes`, `amount` and `from_subscription`. Before doing so, it's better to address outliers: replacing too high values with a reasonable threshold entails a better normalization, less prone to wash out relatively low values and less prone to make them negligible. Scikit-Learn's `RobustScaler` can address such problem by scaling the data according to the interquartile range. By fitting and applying the `RobustScaler` with its default values I observed that the parameter `with_centering=True` possibly transforms 0s in values different from 0s: this may not be what is needed here, given that the information ”no delivered boxes” or ”no compensation amount” would be conveyed more weakly after such transformation. This may have a bad impact especially with models where feature values equal to 0 do not participate in the training (e.g., neural networks).

In this first iteration I won't deal with this problem. I then fitted and applied a `MinMaxScaler` to rescale all numerical values in the range $[0, 1]$. I won't explore any dimensionality reduction approach in this iteration.

3.6. Training & Validation

I generated and saved as a figure the confusion matrix associated with the results of each iteration.

I applied the non-parametric Wilcoxon signed-rank test with a threshold of 0.05 to identify a statistically significant difference between the results of two iterations.

3.6.1. Iteration 1

In this first iteration, I will apply a simple model that enables online learning, in particular a Logistic Regression model trained with Stochastic Gradient Descent (SGD). In all of my experiments I will apply a 10-fold cross validation over the training set, and finally score my model on the test set. This baseline achieved validation and test accuracies respectively of 81% and 83.9%. It is not a relatively high accuracy as compared to a random baseline for balanced binary classification (50%) or the majority class (61%), but I can conclude that the resulting model has predictive power for the task at hand. Given that a superficial preprocessing and a simple, untuned model yielded significant results, I will immediately try to vary on the train/test split ratio, so as to better assess the generalization capability of our models. I believe this is a prioritary action to make based on task and available data. The used test set is temporally located within a single month, a too narrow period as compared to the timedelta of the whole dataset, i.e. more than 3 years. What concerns me is that, even if labels are balanced over the whole dataset, a too temporally narrow test set may still not be representative of the whole population over time; as a consequence, it may happen that our model works well for the population sampled over a specific time period, but not for the others.

3.6.2. Iteration 2

The proportion of samples temporally located in the last month is 0.055, whereas that related to the last two months is 0.108. I will attempt a 90/10 train/test split, so that almost all entries of the last two months are encompassed by the test set. I achieved a validation/test accuracy of 81%/81%. Based on my previous considerations, I will proceed with the 90/10 train/test split.

3.6.3. Iteration 3

Let us focus on multicollinearity. I do not know if robust methods to identify sets of multicollinear features exist, but there is a qualitative approach based on Variance Inflation Factors that may help. I attempted to apply a method leveraging the function `variance_inflation_factor()` of the package `statsmodels.stats.outliers_influence`, but its execution takes too long on our dataset. Given that I do not have enough time available, I will apply a simpler method able to identify pairwise correlations only. I visually checked the correlation matrix over all features to look for values close to 1, indicating variable pairs that are highly correlated with each other. Excepting for the diagonal, it results that just two pairs of variables have positive values, specifically 0.54 and 0.67. These correlation values are small enough to not be worried about: I am now confident enough in proceeding with the next iteration.

3.6.4. Iteration 4

At this point I will take into account again the applied preprocessing steps, in particular the `RobustScaler`. Two over three numerical values in the dataset (`n_boxes` and amount) are characterized by a very significant 0 value, that means respectively no deliveries and no compensation. Rescaling by also subtracting the median would probably de-nullify values, and this may have a negative impact on the informative content of the data series. By setting `with_centering=False` I obtained a validation/test accuracy equal to 81%/81.13%; I evaluated the statistical significance of the result with respect to iteration 2 results and assessed a very small but statistically significant difference; I will consequently confirm this change in my pipeline.

3.6.5. Iteration 5

At this iteration I will tune our Logistic Regression model by means of grid search and cross validation. I focused my grid on an exhaustive exploration over regularization and learning rate. In particular, I applied an `elastic_net` model encompassing pure L1 and L2 penalties and their mixtures (`l1_ratio = [0, 0.05, 0.1, 0.2, 0.5, 0.8, 0.9, 0.95, 1]`), and a variety of learning rates equal to the powers of 10 in the range $[10^{-6}, 10]$. I set a predefined low number of iterations (5) so as to prevent a too long execution time. The number of cross validation folds has been set to 5. The resulting validation/test accuracy is 81.08%/81.05%, providing a slight but statically significant improvement with respect to the previous iteration. The best model resulted to be characterized by a learning rate of 0.001 combined with a pure Lasso regularization method.

3.6.6. Iteration 6

Now that I took the best out of my model, I can attempt to improve the results by applying some dimensionality reduction technique. My hypothesis is that, even if we already have a relatively low number of dimensions (56), given that there are many categorical values whose frequency in the data is very low, then there are too sparse columns not conveying any significant information. I consequently applied PCA at the end of the preprocessing pipeline by letting the method itself filter the strongest components, based on a variance threshold of 0.99; this resulted in a reduced space of 26 components. Unfortunately, despite the small entity of removed information (from the linear point of view adopted by PCA) in terms of explained variance, the adjustment entailed a statistically significant reduction in the final performance. With a validation/test accuracy equal to 81.86%/80.89% I decided to not leverage dimensionality reduction techniques.

3.6.7. Iterations 7-8

Let us now apply a more complex model to our problem. Gradient Boosted Decision Trees identify a class of complex supervised models based on Gradient Boosting. One specific implementation of this model, `XGBoost`, is very popular in the data science environment especially because it dominated Kaggle competitions over the last two years. I decided to apply this model to our task because of its good generalization capabilities, its relative insensitivity to tuning thanks to its strong regularization mechanism and its online learning capability.

Even this model is based on SGD, and is one of the few shallow learning models able to calibrate the needed number of training epochs before overfitting over the validation set, as it happens with deep learning models. I tested such property in this project over the last two iterations; first, I combined this autoregularizing mechanism with cross validation, grid search and early stopping; then, during the execution of such procedure I observed the average number of epochs needed by the algorithm to properly train without overfitting; finally, in the last iteration, I directly instantiated the model based on the best parameters yielded by the grid search and XGBoost autoregularizing process. The validation/test accuracy pairs achieved in both cases are the same not only on average, equal to 81.25%/81.53%, but also from a statistically significant point of view; this endorses the auto-tuning capability of the system. The statistically significant performance difference yielded by this model as compared with previous iterations models promotes it as my best model so far.

4. Conclusions

Based on the minimal performance improvement I achieved with respect to my simplistic baseline model, it is likely that better models for the task at hand still need to be identified and leveraged. There are still several explorations, adjustments and improvements that can be made on this business case, but unfortunately I do not have enough time available till expected submission time. The first action I would take at this point would be to carefully inspect representative sets of samples that my best models misclassified, so as to possibly identify and address generalizations and patterns over the sources of error.