Machine Learning Engineer Nanodegree

Capstone Project

Edgar Valente
April 14th, 2019

I. Definition

Project Overview

In the city of Vitória (state of Espírito Santo, Brazil) there are numerous public health facilities. In these, the number of patients that don't attend to their medical appointments has reached nearly 30% from 2014 to 2015. This pattern is similar all over the country, so it isn't a local issue. This indice represents nearly R$20 million of public cash being wasted by appointments that result in no-shows, taking into account operational costs, SMS sent, confirmation call and employee's time. That's at least R$1 million monthly waste.

Problem Statement

This problem will be addressed mainly as a classification problem, where I'll try to predict if a patient will show up or not on his appointment based on demographic data (age and gender) and appointment characteristics, such as day of the week which it is scheduled and number of days from registration to scheduled appointment.

As viewed in this article, this problem has been addressed before in the San Carlos Clinical Hospital, in Madrid, having demographic data, patient history and classes for medical appointments. Unfortunately, the dataset used in this case isn't as complete, but it should still be possible to predict no-shows with a certain accuracy.

Additionally, a segmentation method through unsupervised learning will be applied in order to find the characteristics of people who show up and people who do not. Care will be taken so that the groups have high variance regarding the target label and each group will be mainly composed of either people who do or do not show up. Having said characteristics might lead to a possible recommendation of any feature that is mutable at the time of appointment.

Metrics

The prediction model shall be evaluated through f1-score, so that it takes into consideration both precision and recall, assuring that an imbalanced distribution of the target label won't have a bias on accuracy. f0.5-score will also be taken into consideration, prioritizing precision, so that it minimizes unnecessary actions that would be taken in order to assure that the patient would show up.
Both metrics are derived from the fbeta-score, which depends on precision and recall:

- Precision: measures how well your predictions are made for the positive label. It is the rate of true positives against all positive predictions;
- Recall: measures how well your predictions actually hit the positive label. It is the rate of true positives against all positive original data;
- F-Score: f1-score is the harmonic mean between precision and recall.
  fbeta-score gives flexibility to this equation by weighting either precision or recall. When beta is closer to 0, it prioritizes precision (which is our case for beta=0.5), and when it is higher than 1 (up to infinity) it prioritizes recall.

These metrics follow the formulas below:

$$\text{Precision} = \frac{tp}{tp + fp} \qquad \text{Recall} = \frac{tp}{tp + fn}$$

$$F_\beta = (1 + \beta^2) \cdot \frac{\text{precision} \cdot \text{recall}}{(\beta^2 \cdot \text{precision}) + \text{recall}}$$

For the attempt in recommendation, something similar to the elbow method will be applied, so that the number of clusters with highest variance among the target label will be chosen. The difference being that the elbow method takes into account explained variance of each cluster, so that adding one more cluster is unnecessary, and I wanna choose a number that maximizes segmentation of the target label.

In this case, we are gonna take into consideration the standard deviation, which is a measure of variance that weights higher values by squaring their distance to the mean:

$$s = \sqrt{\frac{\sum_{i=1}^{N}(x_i - \bar{x})^2}{N-1}},$$

To measure the efficacy of the recommendation, a test set will be used to predict the target label both before and after the recommendation, and the difference of ratio between predicted show ups and no shows will be used to evaluate the recommendation.

II. Analysis

Data Exploration

The dataset used in this project has 300,000 observations and 13 variables (300000, 13). There are two numerical variables (age and days_to_appointment), two labeled categorical (gender and week_day), and other nine binary, containing 1s and 0s representing true and false, respectively.

It is important to notice that the target label (no_show) is slightly unevenly distributed, having around 70% for show-ups (1) and 30% for no-shows (0). The two numerical variables have been checked for skewness, which might need further transformations. Of these, "*days_to_appointment*" seems to be skewed to the left, as shown in the graph below, and will be dealt with when clustering with power transformation and possibly removing outliers through IQR (Interquartile Range).

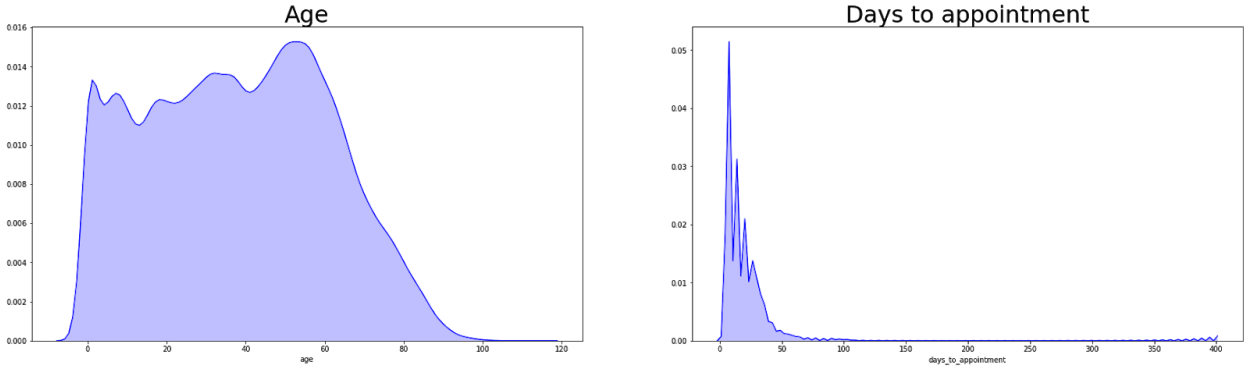

Fig. 1: distribution of age (to the left) and days_to_appointment (to the right).

The variables are more thoroughly described as follows:

- **age**: integer, age of person to make the appointment;
- **gender**: categorical, gender of person to make the appointment, allowing either male or female;
- **week_day**: categorical, the day of the week of the scheduled appointment;
- **days_to_appointment**: integer, days from appointment registration to scheduled appointment;
- **diabetes**: categorical binary, whether the person has diabetes;
- **alcoholism**: categorical binary, whether the person is an alcoholic;
- **hypertension**: categorical binary, whether the person has hypertension;
- **handicap**: categorical binary, whether the person has a handicap;
- **smokes**: categorical binary, whether the person is a smoker;
- **monetary_help**: categorical binary, whether the person receives monetary help from the government, called 'Bolsa família';
- **tuberculosis**: categorical binary, whether the person has tuberculosis;
- **sms_reminder**: categorical binary, whether the person has received an SMS reminder for the appointment one day before it;
- **no_show**: categorical target label, indicating if the person showed up or not on the appointment

Below is a sample of the dataset with its most promising features:

age	gender	week_day	days_to_app	no_show
5	f	thursday	1	0
69	f	friday	4	0
16	f	tuesday	5	1
8	m	monday	14	1
7	m	thursday	15	0

Table 1: sample of dataset with only four explanatory features and target label “no_show”.

Exploratory Visualization

Among the numerical variables (age and days_to_appointment), there are no obvious correlation between days to appointment and show up status. Age, on the other hand, has a clear distinction, where older people seem to attend to their appointments more frequently.

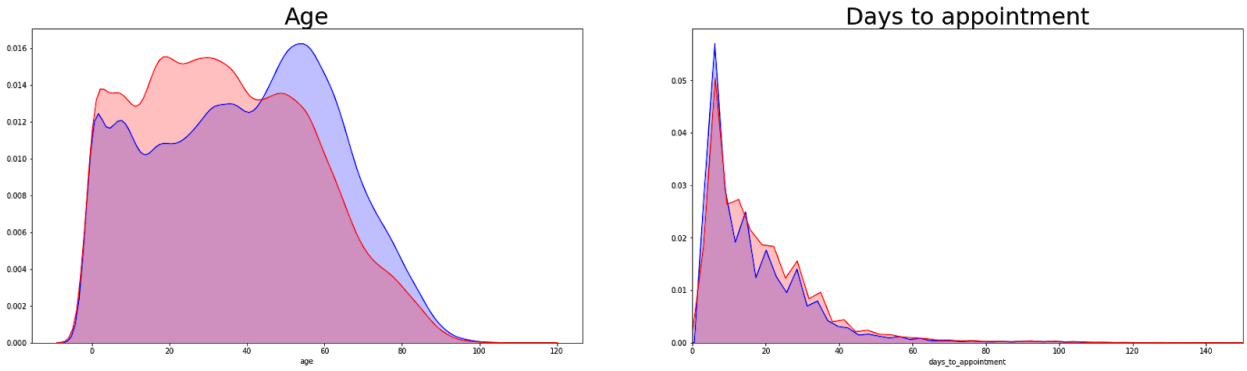

Fig. 2: distributions of age and days to appointment over the week. Blue represents show ups and red no shows.

Extending that visualization to days of the week, in order to find if there's a specific day where people seem more likely to attend, the result is quite similar, with the exception of saturday, where there doesn't seem to be much difference between the age of attendees.

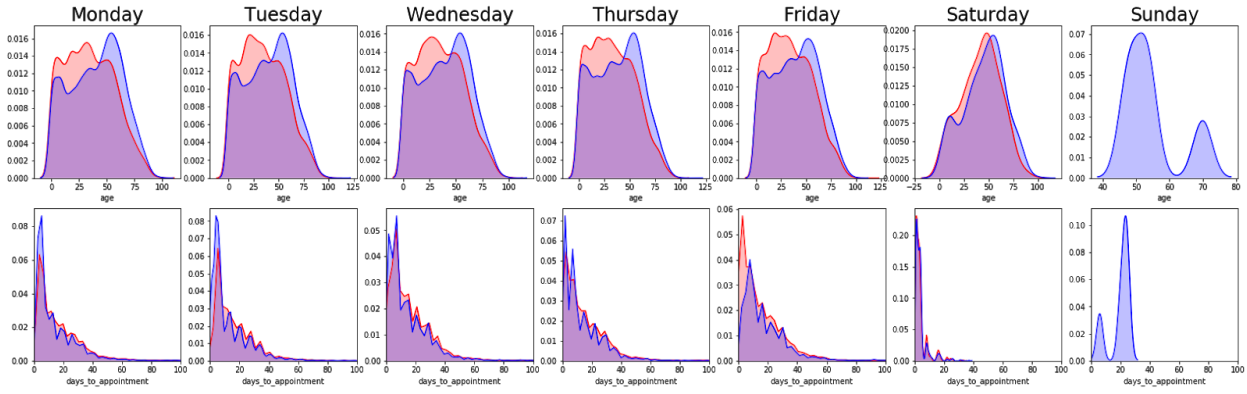

Fig. 3: distributions of age (above) and days to appointment (below) per show up per day of the week, where blue represents show up and red no show.

Comparing both numerical variables and the attendance status, we find an interesting pattern, where people tend to make later appointments on specific periods of their life,

specifically between ages 0-10 (infants) and 50-80 (older people), probably meaning that these periods are where they care the most about periodic appointments. Also, there seems to be a more frequent amount of show ups at higher scheduling, which could indicate that age and days_to_appointment, together, might be a good indicator if a person is gonna attend.

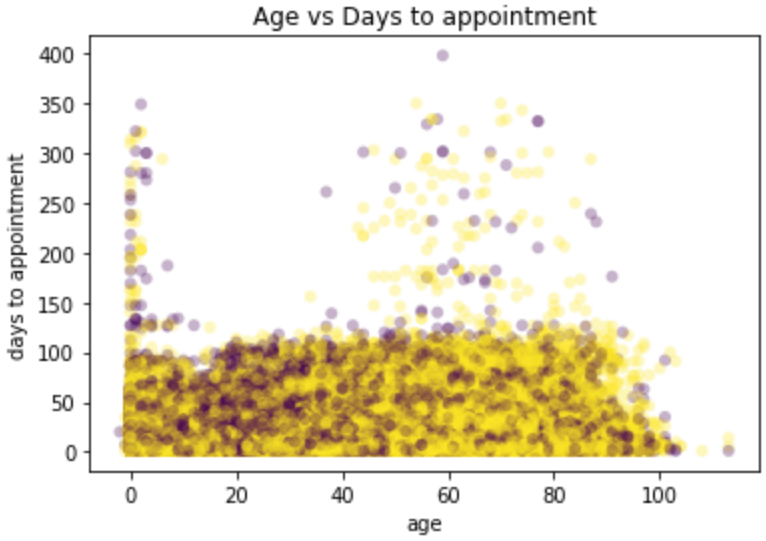

Fig. 4: Age vs days to appointment, where purple dots are no shows and yellow are show ups.

Algorithms and Techniques

Three algorithms will be used:

- **Logistic Regression:** finds a function that fits a sample into either one or the other classification, similarly to using binomial regression;
- **Extreme Gradient Boosting:** an ensemble of trees where every built decision tree is considered a poor predictor, barely predicting better than a random guess. Every new predictor is built in order to weight more heavily the errors on the previous tree so that, in the end, we have an ensemble of weak learners specialized in their own parameters;
- **Gaussian Mixture Model:** a probabilistic model that assumes that every individual is part of a larger gaussian distribution with varied shapes, returning the probability of each individual belonging to each one of these distributions;

Logistic Regression is gonna be used solely as a basis for benchmarking.

The main classifier for prediction will be Gradient Boosting, specifically from the *xgboost* python package, which has been used in many *Kaggle* competitions for classification, and its results are among the best of its category.
The following parameters are gonna be checked for tuning the model:

- max_depth: maximum depth of each generated tree. Low value will avoid overfitting;
- learning_rate: rate at which the model updates each prediction based on the last tree. Low value will avoid overfitting;

For model selection, GridSearchCV from the module *scikit-learn* will be used, checking permutations of previously mentioned parameters while performing cross validation. Also, within the *xgboost* package, the parameter *early_stopping_rounds* will be deployed, where it stops training if the evaluation metric - in our case, f1-score - hasn't improved for the last *n* trees' predictions. This method allows us to avoid overfitting while selecting the model with best metric.

For the unsupervised learning problem, Gaussian Mixture Models from *scikit-learn* will be used, since it allows clusters with varied shapes, contrary to the hyperspherical clusters formed by other algorithms like KMeans. The inputs will be appropriately preprocessed for its application. The clusters centers are gonna be extracted from the model in order to attempt recommendation.

Benchmark

A Logistic Regression algorithm will be applied to have a base metric so that we can use it for benchmarking our final selected model. Also, GridSearchCV already cross-validates models with its hyperparameters permutations.
The results from previously mentioned article are gonna be used as a basis for comparison, since the problems are quite similar, even though the data varies slightly, having its final prediction error of around 28% (equivalent accuracy of 72%).

III. Methodology

Data Preprocessing

A few transformations had to be made in order to apply both models:

- One hot encoding (supervised and unsupervised);

- Outlier removal (supervised and unsupervised);
- Feature selection (supervised and unsupervised);
- Power transformation (unsupervised);
- Normalization (unsupervised);

Both categorical variables - *week_day* and *gender* - have been one hot encoded. For outlier removal, the Interquartile Range method was applied on *days_to_appointment* after power transformation specifically for this purpose. Since it is skewed to the left, only higher values (to the right) were removed using this method, taking out 96 exaggerated samples from the dataset, which all had *days_to_appointment* values higher than 224. More samples could have been removed, but it is also intended to have clusters formed by these individuals.

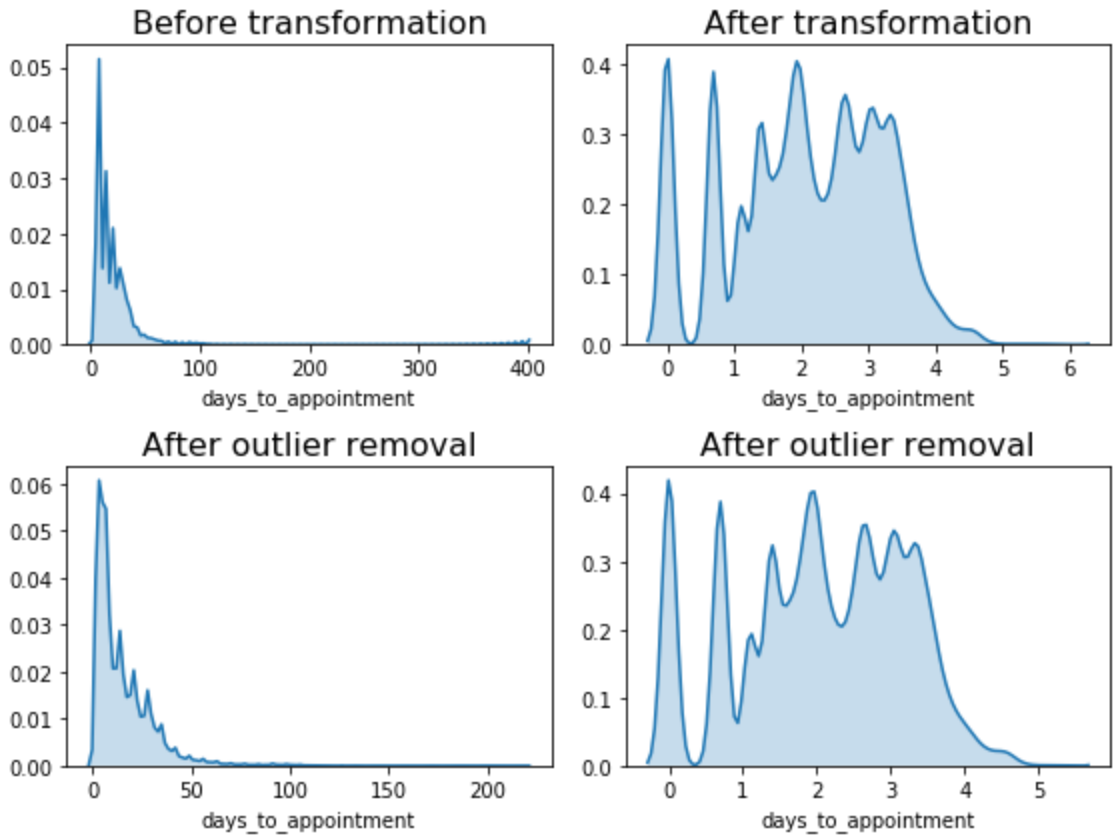

Fig. 5: distribution of days_to_appointment before and after log transformation and after outlier removal.

In order to more accurately segment clusters, feature selection had to be applied, so that each group is more likely to describe its target label.
Two methods were used for this purpose:

- Information Gain from xgboost model. It takes into consideration how much information is obtained from the data after tree splits at one specific feature.
- Mutual Information from sklearn. It segments each feature and compares its contained information with the segmented target label. The higher this metric, the more likely it is for both features to have their segments clustered together.

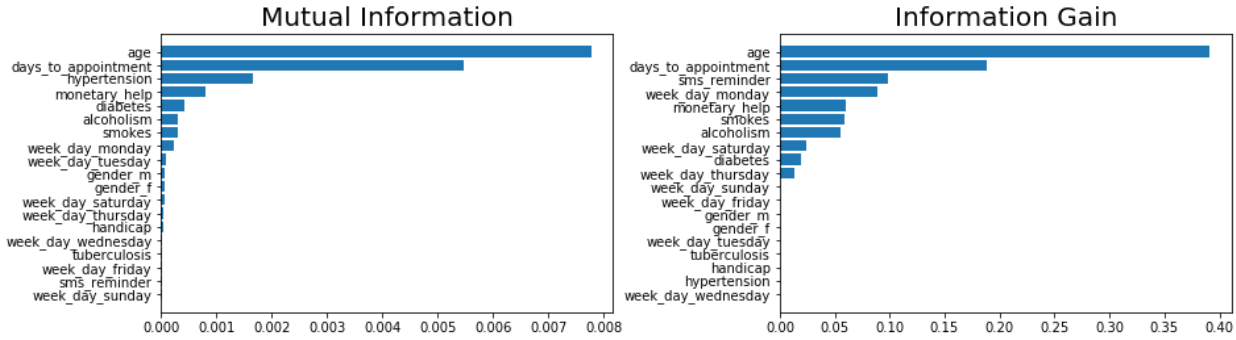

Fig. 6: feature importances calculated with Mutual Information (left) and Information Gain (right).

Since it is desired to apply both supervised and unsupervised learning on the same data, only features with high values on both metrics were chosen, those being **age**, **days_to_appointment** and **monetary_help**. Also, the categorical variables *gender* and *week_day* are not being used after feature selection, so there is no need for further one hot encoding.

Tree ensembles like the one used for classification don't need normalization, since they are not based on sample distances. Gaussian Mixture Model, on the other hand, considers euclidean distance for clustering, hence the need for normalization on both numerical variables, being applied with *scikit-learn*'s MinMaxScaler and transformed to values in the range between 0 and 1. Power transformation (log, in this case), however, is only needed on the variable *days_to_appointment*, previous to normalization, to adjust its skewness into a normal distribution.

Implementation

The preprocessing step resulted in two sets of data, all maintaining the original proportion of target label, which will be used separately for supervised and unsupervised learning:

- Train and test data split into classifier features and target label (supervised);
- Train and test data with all features, with log transformation and normalization (unsupervised);

That said, three models have been made:

- Logistic Regression model for benchmarking;
- Extreme Gradient Boosting for main prediction model;
- Gaussian Mixture Model for clustering and further recommendation attempt;

For supervised learning the first set of train and test data was used (in notebook *"supervised"*), where the features **age**, **days_to_appointment** and **monetary_help** were used, with **no_show** being the target label, while the positive label 1 means no show while 0 means show up.

The Logistic Regression model has been made using solver *lbfgs*, which is an optimization algorithm that uses low amount of computer memory. It wasn't expected to be a good model, but for benchmarking instead. Its accuracy score measured 0.7, seeming a good result, but accuracy isn't the desired metric. Its f1-score was 0.005, which is terrible for predicting no shows. If we consider 0 (show-up) as being the positive label, this value is 0.82, but this isn't the objective in this project. This is because it predicted 99% of the test set as showing up, while there were actually only 70% of them.
The refinement of this model wasn't in the scope of this project, since it's not intended to be its main model. But after such poor result in predicting no shows, an attempt to find better hyperparameters through GridSearchCV has been made.

Originally, f1-score was gonna be prioritized since it describes a more concise model. Given such results, the focus is gonna be to optimize f0.5-score, where at least we wanna be sure that the model is finding people who won't show up, so that we can take actions to prevent that, if applicable. Since f0.5-score weights precision more heavily, it's an appropriate approach.

This problem has been addressed when training the second model - extreme gradient boosting (XGBoost). It was cross-validated using GridSearchCV with 3 fold cross-validation and experimenting with hyperparameters *max_depth* (3, 5, and 7) and *learning_rate* (0.3, 0.5 and 0.7) to select the best ones and using f0.5-score for model selection, building each model 50 trees.

Since the result seemed to be considerably better with higher max_depth and learning_rate, I decided to manually iterate over each of these arguments with even higher values to verify how the f1-score is raised along with each parameter, which ended up being the highest possible (max_depth 11 and learning_rate 1.3).

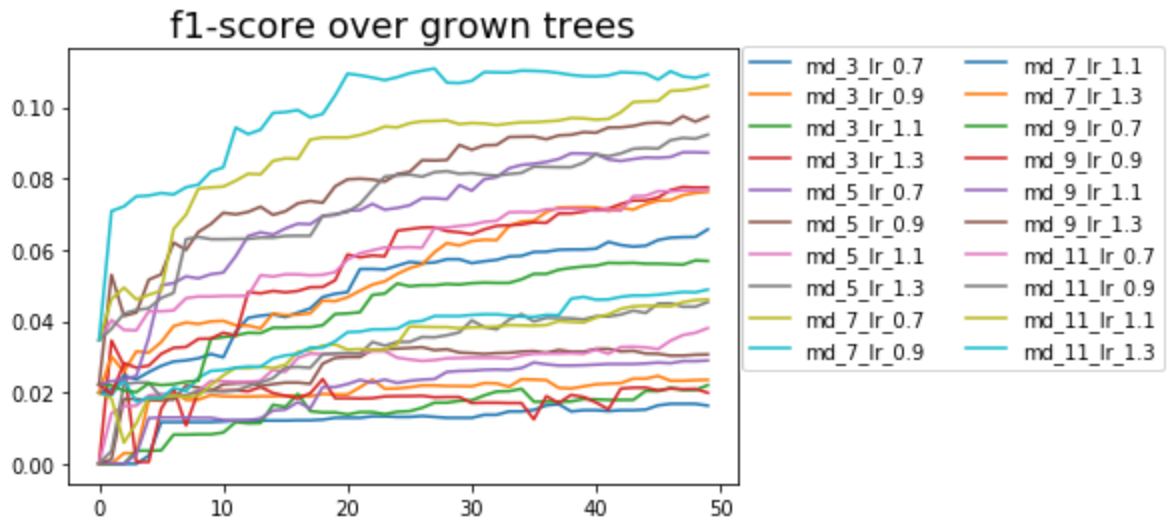

Fig. 7: f1-score measured over each model along each grown tree.

Within the best model, the *xgboost* package also allows an evaluation function to select the best model trained while growing the trees, selecting the metric with lowest value. Since high f0.5-score is desired, the custom function supplied to the algorithm is the inverse of this metric. The model was then applied taking into consideration found parameters, which were learning_rate 0.9 (chosen over 1.3 since values higher than 1.0 might lead to overfitting) and max_depth 11, building 2000 maximum trees and early stopping if the evaluation metric *inverse f0.5-score* hasn't improved over the last 200 built trees. Its results are reported in the table below:

	PREDICTIONS						
	show_up	**no_show**	**total**	**error**	**f1-score**	**f0.5-score**	**precision**
show_up	49.703	2.465	52.168	4,73%	0,81	0,74	0,70
no_show	21.246	1.562	22.808	93,15%	0,12	0,20	0,39
total	70.949	4.027	74.976	48,94%	0,46	0,47	0,55

Table 2: results of xgboost model stopping at tree 1007 and using f0.5-score as early stopping parameter.

The training stopped at the 1007th tree, but f0.5-score hasn't improved much since the 200th tree, as seen below:

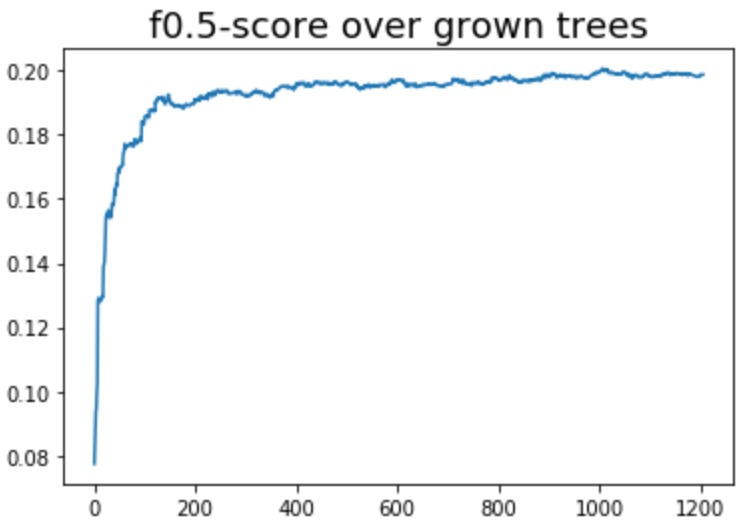

Fig. 8: f0.5-score measured over grown trees.

The unsupervised learning approach is applied on the second set of train data (in notebook *"unsupervised"*, so that the same individuals would be used. This dataset is, however, log transformed and normalized, as mentioned in the "Data Preprocessing" section.
The Gaussian Mixture model was used, and the number of clusters desired was selected based on the standard deviation of the its target label centroid values. One model with n clusters was created for each iteration from n=2 to n=24. The standard deviation values, however, were all close to 0.5, which was unexpected. That means every cluster has a centroid "no_show" dimension of either 0.0 or 1.0. So, even though the values vary from 0.47 to 0.49, I decided to arbitrarily choose a value of n where the standard deviation was 0.49 and, in this case, n=11.

The recommendation attempt, however, didn't have a decent basis for validation, since the prediction model that would be used for it was not good enough. That said, the significance of the recommendation would be hard to measure. This loss makes the unsupervised learning approach unutilized.

Refinement

There was one last attempt to get a better prediction model by creating a custom loss objective function. By default, the classification loss function doesn't distinguish its penalty for 1 or 0 predictions. Since I am attempting to focus on precision, I want the model to be right when it predicts ones. In this case, I've attempted to penalize its errors on these predictions by multiplying them by 1.3 with the function described below:

```python
def custom_objective(y_true, y_pred):
    grad = y_pred - y_true
    hess = y_pred * (1.0 - y_pred)
    grad = np.where(grad < 0.0, 1.3 * grad, grad)

    return grad, hess
```

Attempts were made with penalties of 1.1, 1.2, 1.3 and 1.4. This value was chosen based on the best obtained metrics without learning penalties. Starting from 1.4, the model loses its learning stability. The hyperparameters used were still max_depth of 11 and learning_rate of 0.9.

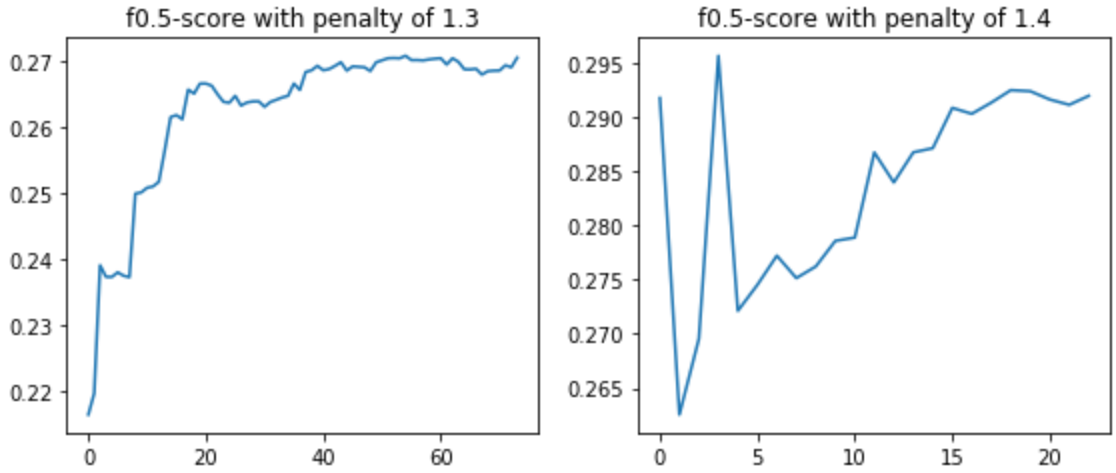

Fig. 9: f0.5-score over grown trees with penalties 1.3 and 1.4.

Its results described a better f0.5-score (0.27 over previous 0.20) and even f1-score (0.18 over 0.12), as shown on the table below:

	PREDICTIONS						
	show_up	**no_show**	**total**	**error**	**f1-score**	**f0.5-score**	**precision**
show_up	47.947	4.221	52.168	8,09%	0,80	0,74	0,70
no_show	20.065	2.743	22.808	87,98%	0,18	0,27	0,39
total	68.012	6.964	74.976	48,03%	0,49	0,50	0,55

Table 3: results of xgboost model after penalty in loss function of 1.4, with f0.5-score as early stopping parameter.

IV. Results

Model Evaluation and Validation

The results obtained over development are described in table 2 and, finally, in table 3. There are a few considerations to be made:

- After refinement, the values predicted on training data during training validation had an f0.5-score close to 0.34, against 0.27 found on the validation test set. This might indicate a slight overfitting, which couldn't have been properly addressed. It was still much better than any previous model.
- Even with overfitting, the model seems to be concise against unseen data, as observed from results from mentioned tables. Since its focus was made on precision, the only certainty desired for this method was to correctly assert that a patient isn't gonna show up. With these results on the test set, it can be assured that, from every 10 predictions of no-shows, nearly 4 won't really show up.

Justification

The problem described ended up being a bit different from the mentioned article, having less data to focus on. The former used accuracy as a validation metric, while this problem used f0.5-score, because of the lack in predictability. Having to focus on precision, the model as tuned to predict well patients that don't show up on their appointments, even though it misses a lot of them. This approach was chosen so that actions taken by the hospital wouldn't be taken lightly, since any explicit action would also cost money.
With a precision of nearly 40%, there are still gonna be 60% wasted approaches to avoid a patient from not showing up, which would still be better than 100%. That said, there's certainly room for improvement, both in data as in algorithmic approaches.

V. Conclusion

Free-Form Visualization

The prototype for recommendation function has been made, based on cluster centers. It hasn't been improved, however, due to the lack of predictability from the classification model and, hence, the lack of significance for validation.

It was an attempt to calculate the euclidean distance between a new individual and the cluster centers, but lacking two dimensions: the feature to be recommended and the target label.
Since the clusters were neatly segmented when considering the target label, all other features from each cluster describe a type of patient. With the cluster centers and extracting its recommendation feature, we can estimate what good recommended value would be for a new individual based on solely the present variables.
The function takes as parameters the following data:

- data: dataset of sample individuals to attempt a recommendation;
- centers: centers of every found cluster;
- rec_index: index of recommendation feature on the dataset;
- Input_indexes: list of indexes of present features;
- label_index: index of target label on the dataset;

Of course, for practical use, there wouldn't be a rec_index or label_index in a real case dataset, but these wouldn't be needed anyway. They are used in this example for the sole purpose of validating what data we have.

```
def calc_distance(a, b):
    """
    Calculates euclidean distance between two points
    """
    return np.sqrt((np.array(a) - np.array(b))**2).sum()

def calc_distances(a, centers, indexes):
    """
    Calculates euclidean distances between one point and several
    """
    return {i: calc_distance(a[indexes], j[indexes]) for i, j in enumerate(centers)}

def distances(individuals, centers, indexes):
    """
    Calculates euclidean distances between two arrays of points
    """
    return np.array([calc_distances(np.array(j), centers, indexes) for i, j in individuals.iterrows()])

def recommend(data, centers, rec_index, input_indexes, label_index):

    individuals_distances = distances(data, centers, input_indexes)
    valid_clusters = [i for i, j in enumerate(centers) if j[label_index] > 0.75]
    recommended_values = {k: v[rec_index] for k, v in enumerate(centers) if k in valid_clusters}

    recommendations = []
    for individual_distance in individuals_distances:
        valid_distances = {k: v for k, v in individual_distance.items() if k in valid_clusters}
        recommended_cluster = min(valid_distances, key=valid_distances.get)
        recommended_value = int(recommended_values[recommended_cluster])
        recommendations.append(recommended_value)

    return np.array(recommendations)
```

Fig. 10: attempt at recommending a variable for getting a better target label.

Reflection

The project was promising, and the problem is a real one. The data provided is, however, not sufficient for the desired solution. It was expected to have a decent prediction score based on demographic data and days of the week. The former had real value (age specifically), as seen in the feature selection section, but the second has been completely ignored, since it had no patterns.
When implementing, however, the data showed its unexpected lack of predictability. The focus on precision seemed the only feasible solution, and so it has been attempted, with reasonable success.
A recommendation was intended to be made, but with a bad model, there is no significant way to validate its results.

Improvement

The mentioned article had a few interesting data points that lacked in this project, and the results would probably be much better had the hospital provided them:

- History of patient schedules (dates and status of previous appointments);
- Type of medical appointment;
- Specialty of appointment;

Having this data, the whole approach could be repeated. There could then be patterns between days of the week and attendance. A recommendation attempt could be made on this variable, as on the hour of scheduled appointment.