Predicting Hospital No-Shows

Dennis Wang

Data Science Institute, Brown University

GitHub[2]

1 Introduction

Patient no-shows present risks to healthcare organizations' revenues and hamper effective resource utilization. Additionally, they reduce the possibility of follow-up care, leading to poorer patient outcomes. Predicting which patients are likely to no-show to their appointments and why allows for outreach efforts to target these patients more effectively, as well as address issues that impede patients' abilities to make their appointments.

Healthcare No Show Appointment Data

The dataset was sourced from Kaggle and encompasses 107,000 medical appointments made in various Brazilian neighborhoods. The target variable "ShowedUp" indicates whether a patient showed to their appointment. The 14 other features include details about the patient, their health history, and their scheduling timeframe (i.e., when they scheduled their appointment and when they scheduled their appointment *for*). [1]

2 Exploratory Data Analysis

Each row in our dataset represents one appointment, uniquely represented by an "AppointmentID". However, I needed to know whether each patient, uniquely identified by a "PatientID", occurs more than once. If this was the case, our data would not be independently and identically distributed (iid). I compared the counts of unique values of PatientIDs against the total rows of the dataset and plotted the top 10 most-recurrent patients in the dataset:

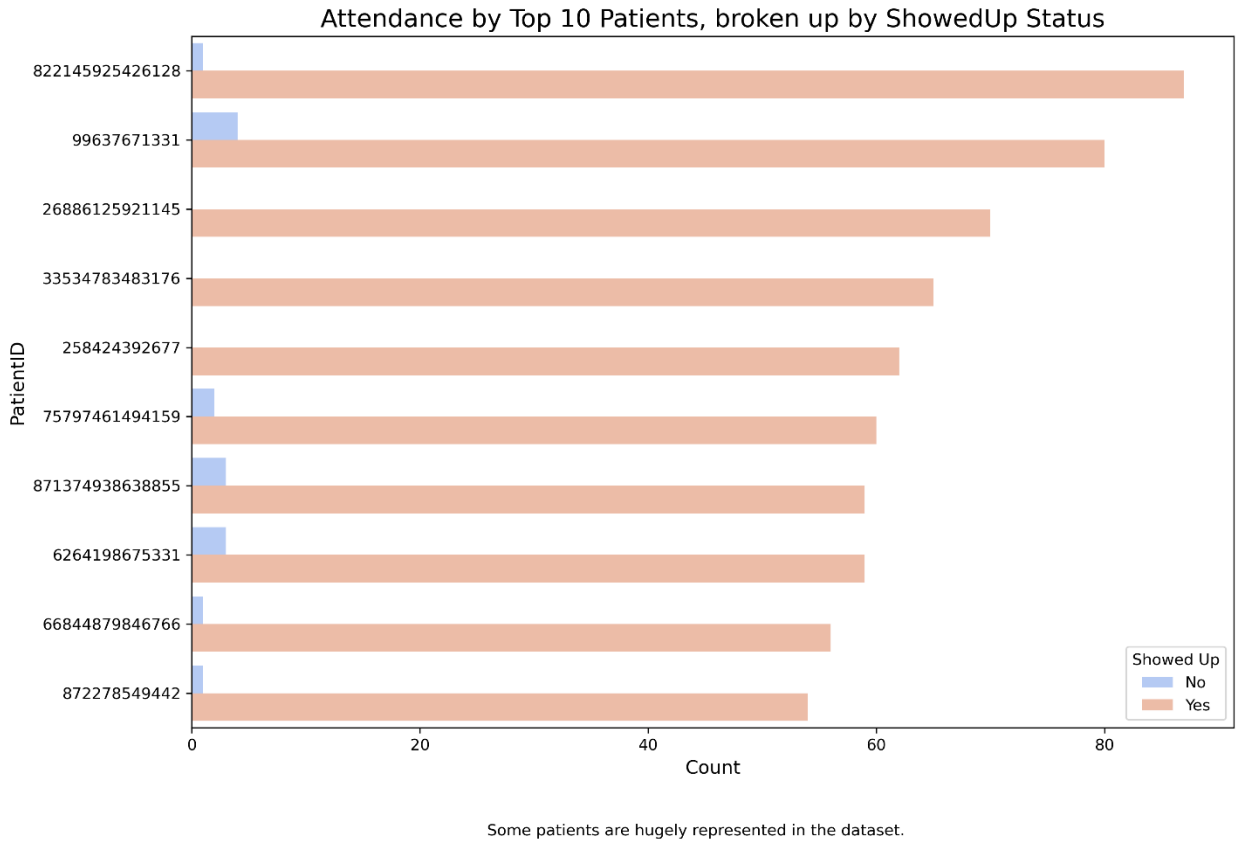

Figure 1. The top reoccurring PatientIDs in the dataset

My initial comparison showed about 60,000 unique patients in our dataset. Compared to the number of appointments in our dataset of 107,000, this in combination with the attendance plot, shows that we need to group by PatientID when splitting our data.

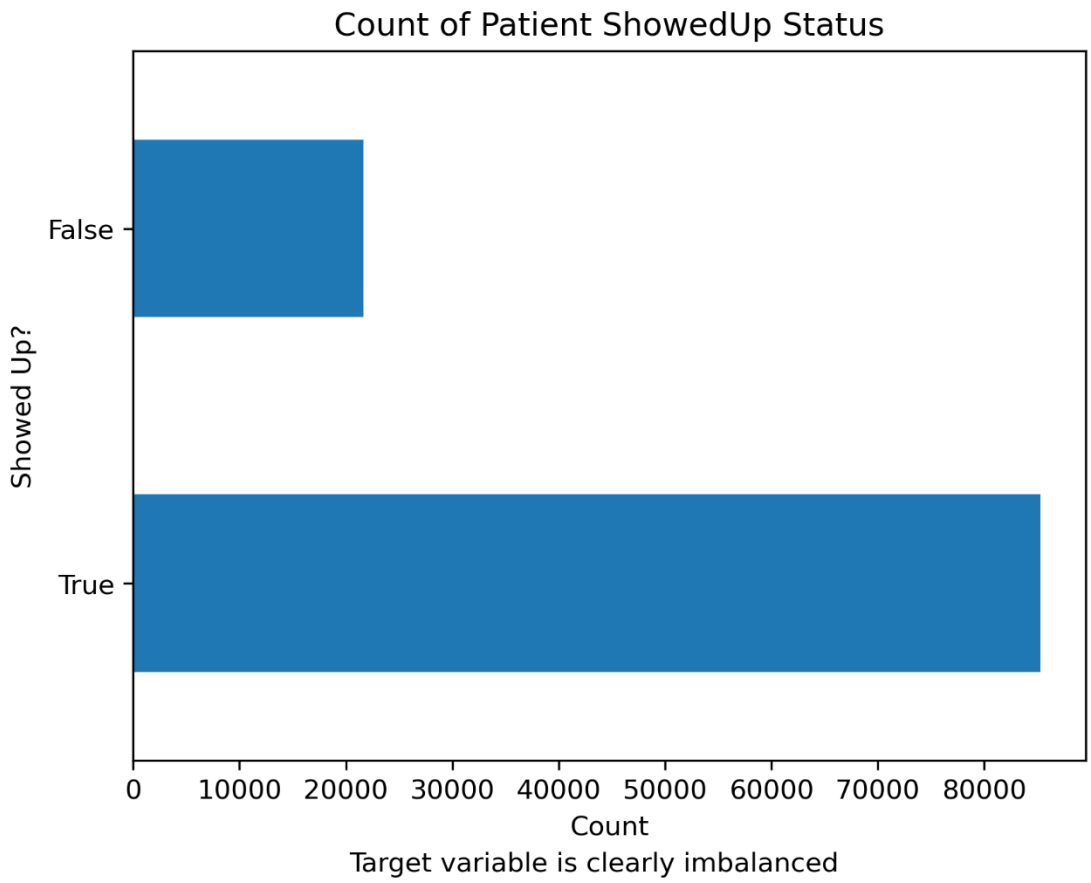

Figure 2. The distribution of values of ShowedUp

Visualizing our target variable “ShowedUp” reveals that it is clearly imbalanced. In this dataset, about 20% of the appointments have a no-show status. Thankfully, people tend to make their appointments most of the time, but this target variable imbalance means we need to stratify our target variable when splitting.

I also wanted to look at one of our features “DateDiff”, which represents the number of days between two other features: ScheduledDay, the day when a patient scheduled their appointment, and AppointmentDay, when the appointment was scheduled for. My hypothesis was that the longer this duration, the more likely the patient was to miss their appointment. To visualize this, I first compared boxplots of DateDiff grouped by ShowedUp status:

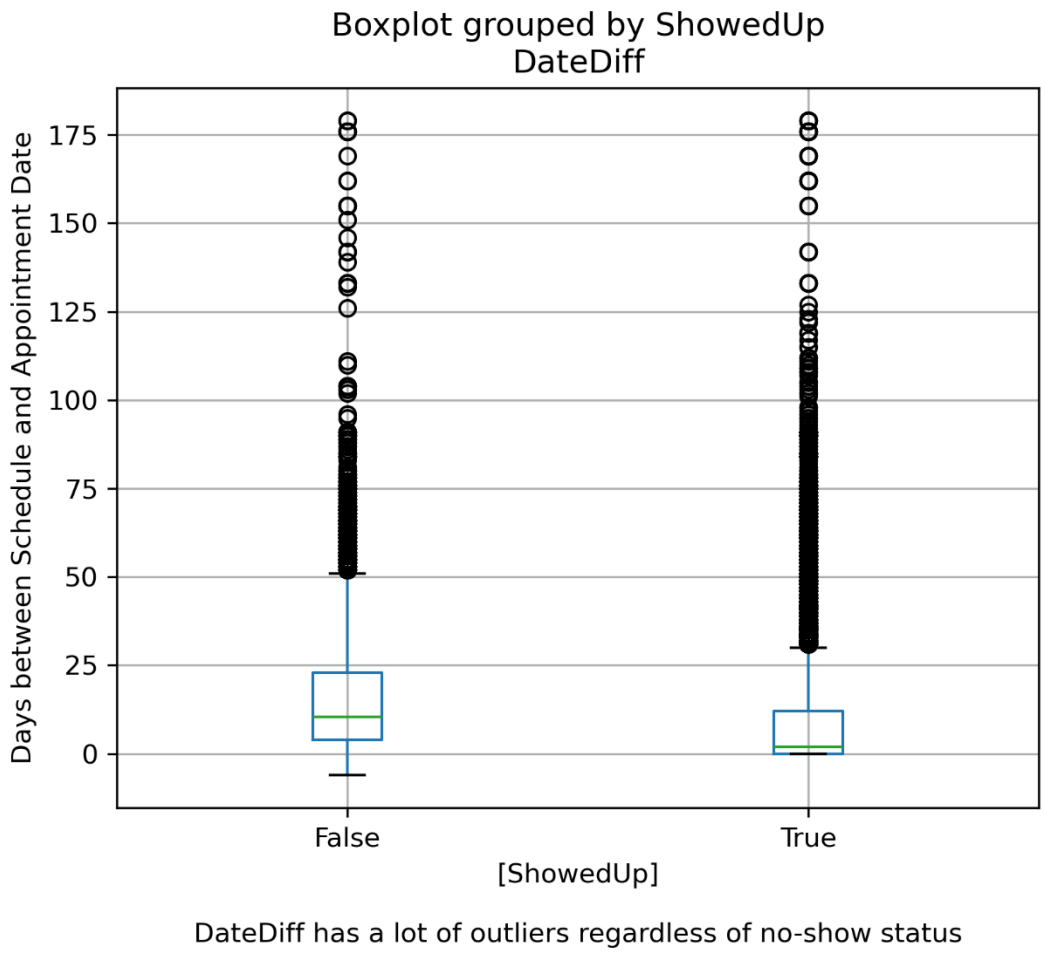

Figure 3. The distribution of the DateDiff feature, broken up by ShowedUp status

Appointments where the patient shows up have a lower median DateDiff compared to those with no-shows. Interestingly, both boxplots contain large amounts of outliers, with some reaching almost 6 months. Another notable feature is that there exists negative DateDiff values for the False boxplot, indicating that some appointments were scheduled in the system after the appointment had supposedly taken the place, though in these cases, the patient never showed. Perhaps it was a clerical error or some method of retroactive data entry.

The second way that I wanted to visualize DateDiff was to compare each value of DateDiff against the proportion of no-shows. My hope was that there would be a clear positive trend between DateDif and the proportion of no-shows.

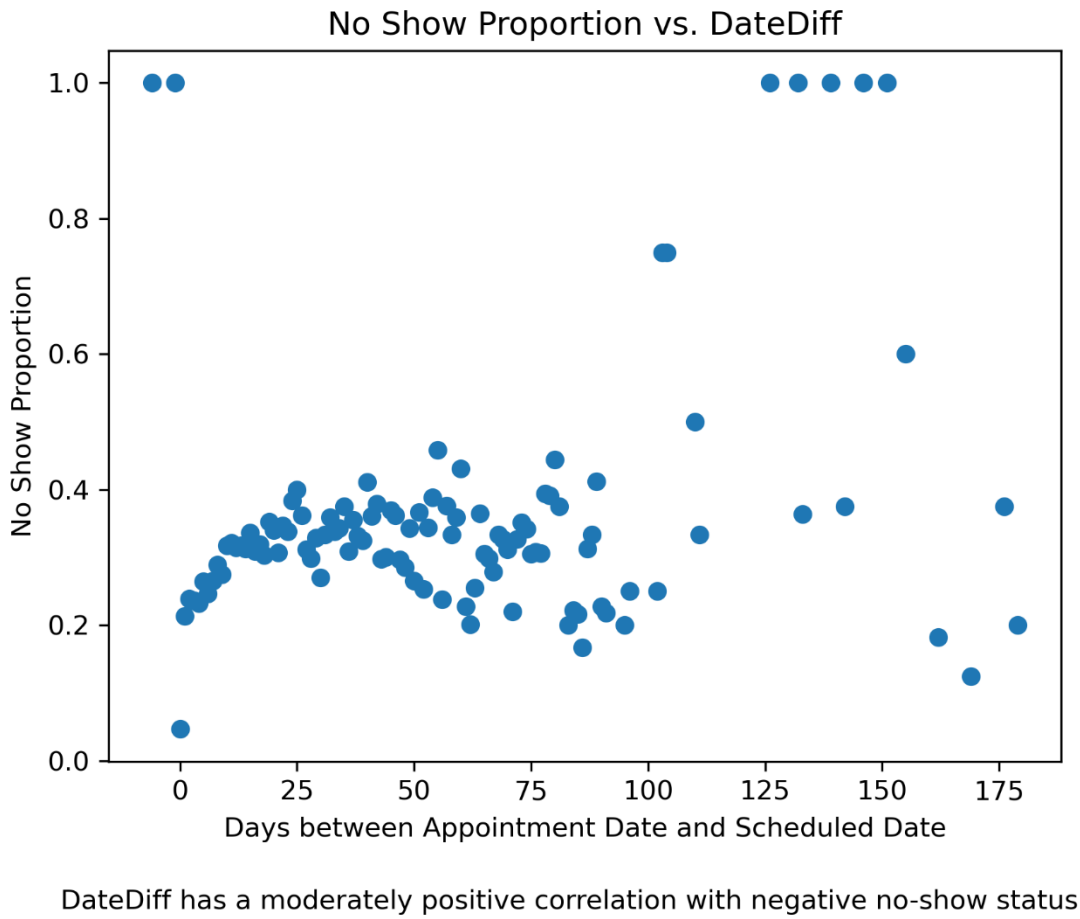

Figure 4. DateDiff against the proportion of no-shows

There appears to be a strong initial correlation that as DateDiff increases, so does the proportion of no-shows. However, after about 30 days, this correlation weakens considerably.

One final feature that I wanted to investigate was “SMS_received”, which is a binary feature indicating if a patient received a text reminder prior to the appointment. My hypothesis was that this would be a strong predictor and there would exist a positive correlation with ShowedUp status. I visualized this through two stacked bar charts: the first showcasing simply the raw counts, and the second showcasing a normalized version.

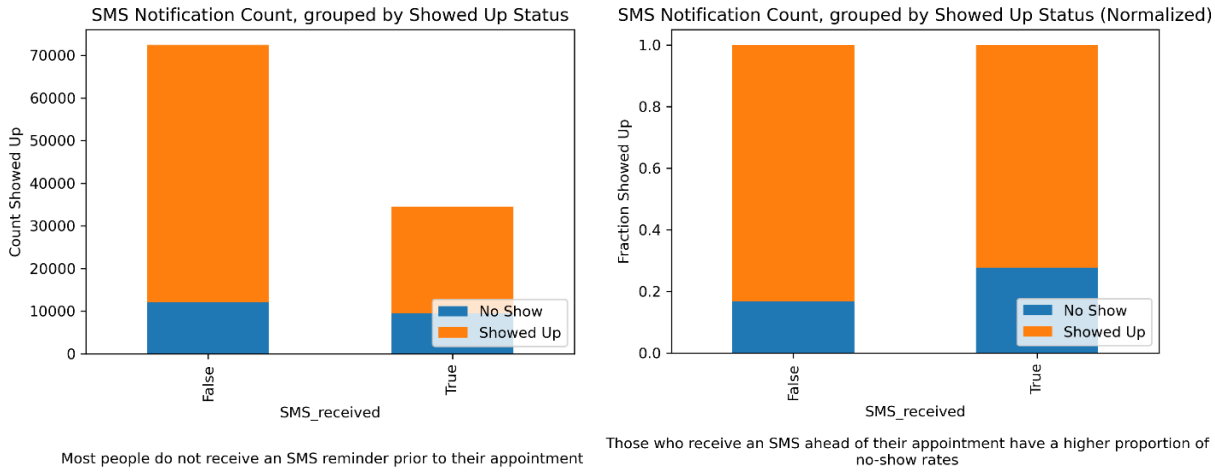

Figure 5. The raw counts and normalized distribution of values of the SMS_received feature, grouped by ShowedUp status

The first surprising thing was that only in about a third of appointments is a reminder sent to the patient. Furthermore, in the normalized plot, we can see that of the appointments where a notification is sent out, those have a *higher* proportion of no-show rates.

3 Methods

Given that our data is not iid and that our target variable is imbalanced, we need a splitting strategy that is able to both group by PatientID and stratify the target variable. For this reason, I used sklearn's StratifiedGroupKFold splitter. The splitting strategy involves two rounds of splitting: The first round splits the dataset into a "trainval" and test set. The test set is held out to evaluate the model's performance on previously unseen data points. Meanwhile, the second round of splitting involves using the "trainval" set for multiple rounds of cross validation in GridSearchCV.

Like our splitting strategy, our preprocessing strategy also involves two rounds of preprocessing. The first round involves data cleaning and feature engineering that can be done prior to the first round of splitting. Four new features, "ApptDayOfWeek", "SchedDayOfWeek", "ApptMonth", and "SchedMonth" were created from the "AppointmentDay" and "ScheduledDay" features, which contained date values. Subsequently, a "first preprocessor" pipeline was constructed that ordinal encoded these columns in the order that months occur in a calendar year and days follow in a week. In addition, the "Neighborhood" feature was one-hot encoded, creating 81 new columns in the dataset, each representing a unique neighborhood in the dataset. After this, the first round of splitting occurs which produces the "trainval" and test sets.

Afterwards, a secondary pipeline is constructed which involves secondary preprocessing, a final StandardScaler function which is applied to all the feature values, and the machine learning model itself. The secondary preprocessing step involves a MinMax scaler which is applied to the

Age feature since the age values of a patient tend to be fairly contained. On the other hand, our previous EDA showed that DateDiff has lots of outliers, so it follows to apply standard scaling. A subsequent StandardScaler function is applied to all features, including the ordinal-encoded and one-hot-encoded features. After preprocessing, we are left with 94 features.

For our model-building, we tested five different machine learning algorithms: Logistic Regression, KNeighbors Classifier, Random Forest Classifier, Support Vector Classifier, and XGBoost Classifier. The test parameters values for each algorithm are provided in Table 1:

Logistic Regression	solver: saga max_iter: 1000000 class_weight: [None, 'balanced'] penalty: [‘l1’, ‘l2’] C: np.logspace(-2,2,13)
KNeighbors Classifier	n_neighbors: [1, 3, 5, 10, 20, 30] weights: [‘uniform’, ‘distance’, None]
Random Forest Classifier	class_weight: [None, 'balanced'] max_depth: [1, 3, 5, 10, 30, 100] max_features: ['sqrt', 'log2', 0.3, 0.5]
Support Vector Classifier (SVC) (sampled 0.20)	class_weight: [None, 'balanced’] C: [1e-3, 1e-2, 1e-1, 1e0, 1e1, 1e2, 1e3] gamma: [1e-4, 1e-2, 1e0, 1e2]
XGBoost Classifier	n_estimators : 10000 early_stopping_rounds: 50 subsample: 0.9 colsample_bytree: 0.9 eval_metric: 'aucpr' scale_pos_weight': [weight_ratio, None] learning_rate: 0.1 max_depth: [1, 3, 5, 7, 10] reg_alpha: [0, 0.01, 0.1, 1, 10] reg_lambda: [0, 0.01, 0.1, 1, 10]

Table 1. Tested parameter values for different machine learning algorithms

The second pipeline and the following parameter grid are then passed into GridSearchCV along with several other parameters, including ‘f1’ for the ‘scoring parameter and StratifiedGroupKFold for the cross validator. The scoring parameter is the strategy used by GridSearchCV to evaluate the performance of the model during cross validation on the test set. I specifically use the F1 score for this parameter since F1 score is often used to account imbalanced target variables given that its calculation does not involve True Negatives, which would otherwise inflate its score.

The whole process, from splitting to model building, was iterated over five random states to measure the variability in the F1 score caused by the randomness in splitting and non-deterministic model algorithms. In each iteration, the best performing model of each type of algorithm and its test scores are recorded.

4 Results

The results of each machine-learning algorithm's best model, averaged over 5 random states, are presented below:

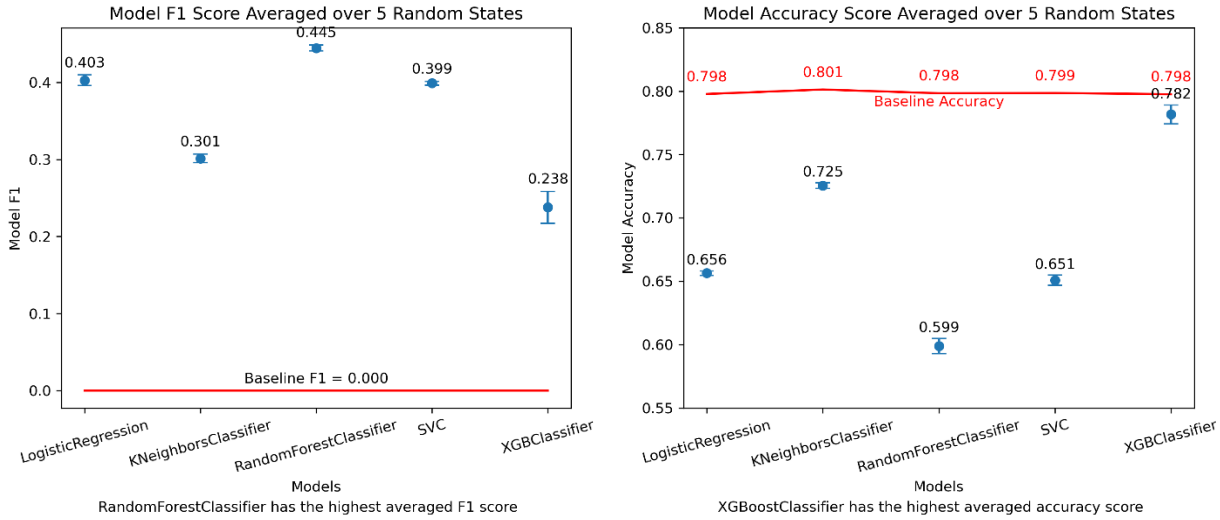

Figure 6. F1 scores and accuracies of ML models averaged over 5 states, compared against the baseline

Our RandomForestClassifier has the highest average F1 score at 45%, and our XGBClassifier has the lowest at 24%. Standard deviations were low across the board, likely due to our KFold Cross Validation, except for XGBClassifier which has a noticeably larger standard deviation at 0.02 compared to other models at around 0.003. With a standard deviation of around 0.004, our RandomForestClassifier's averaged F1 score is 120 standard deviations above the baseline. Though our models are significantly better compared to the baseline F1 of 0, a maximum F1 score of 45% is not particularly impressive.

When evaluating accuracy, the results are inverted. XGBoost has the highest accuracy at 78% and RandomForestClassifier has the lowest at 60%. All our models' accuracies performed worse than the baseline accuracy that hovered around 80% between random states.

Focusing on the RandomForestClassifier, we can break down its predictions using a confusion matrix:

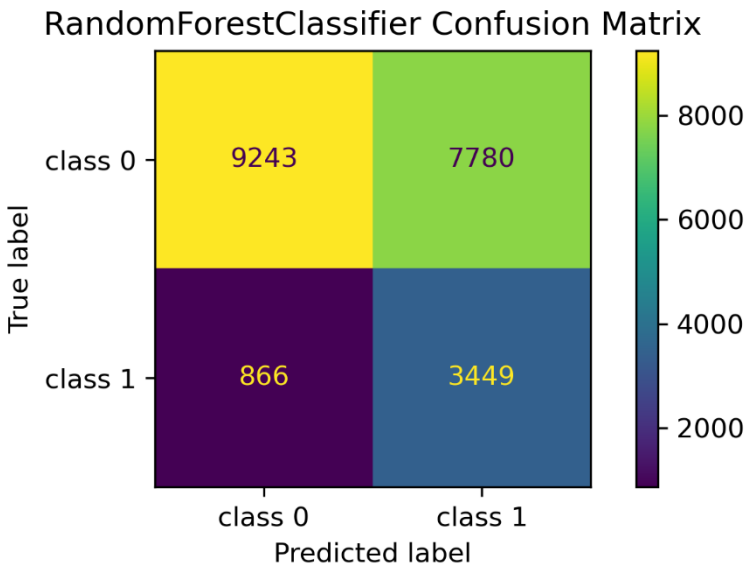

Figure 7. Confusion Matrix of Random Forest Classifier in Random State 0

The Random Forest model appears to be making an even number of class 0 and class 1 predictions. It is not purely predicting the majority label (class 0), but its performance remains relatively poor, especially given the large number of False Positive predictions.

We have several methods at our disposal to measure which features contribute the most to the model's predictions. Global feature importance metrics available to Random Forest Classifier models include Permutation Feature Importance, Gini Importance, and Shap values.

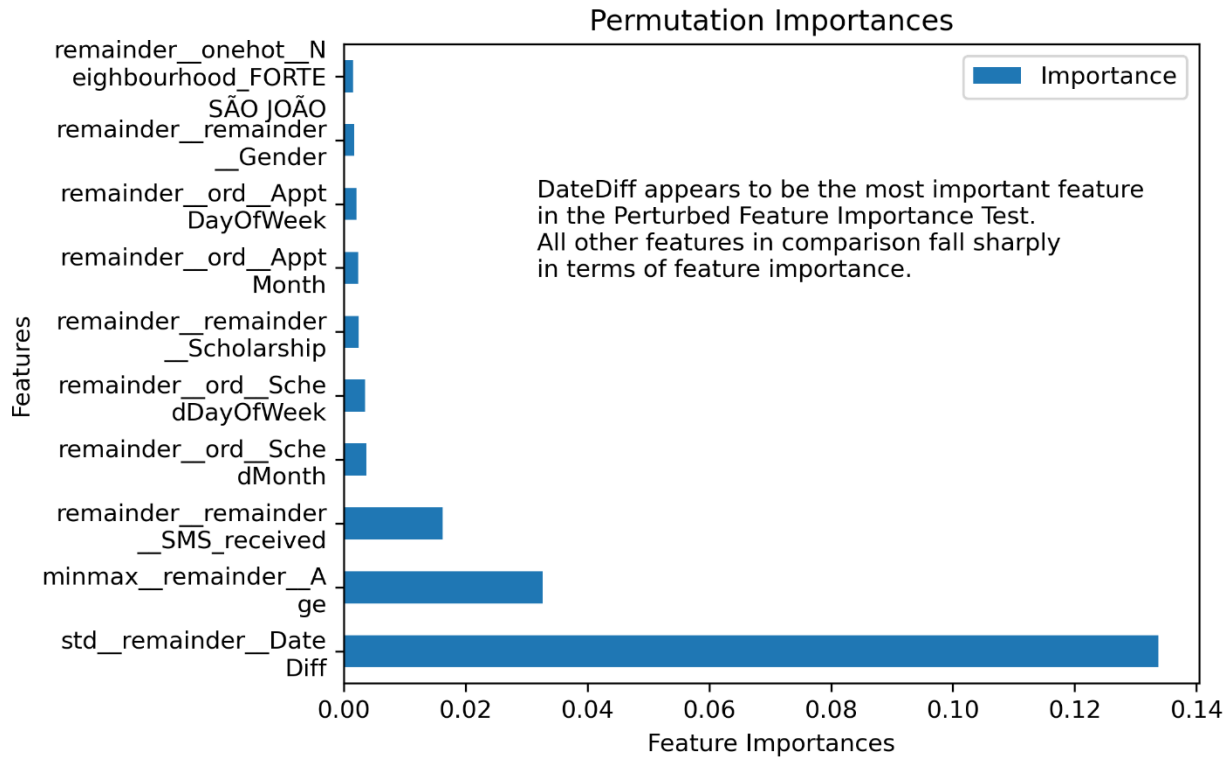

Figure 8. Top 10 feature importances by Permutation Importance

The permutation feature importances method estimates the importance of each feature by measuring the drop in predictive power when shuffling (i.e. “perturbing”) that feature’s values.

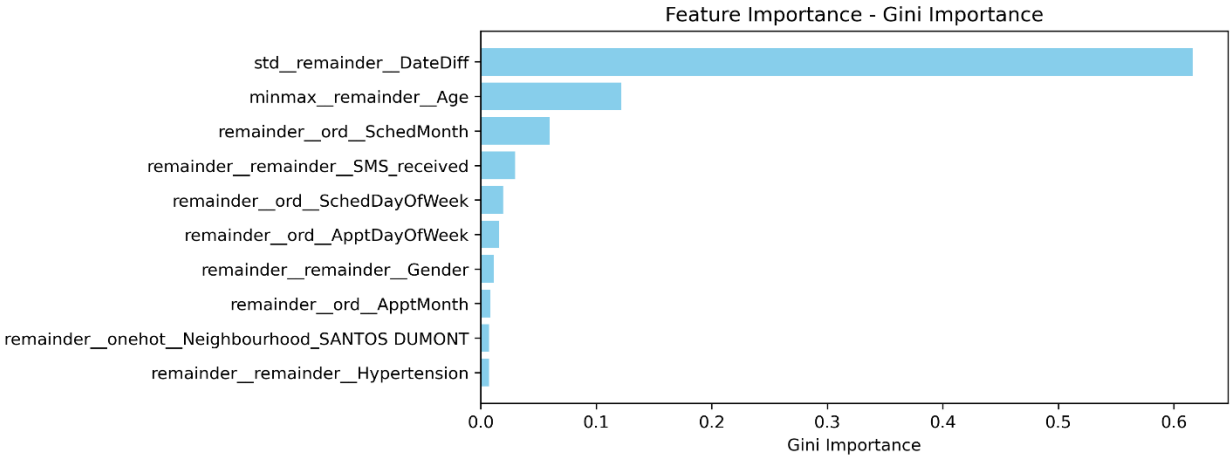

Figure 9. Top 10 feature importances by Gini Importance

Gini importance is a metric specific to tree-based methods. It estimates the importance of a feature computed as the decrease in Gini impurity brought by introducing a feature. Gini impurity is a measure of how well data points are split into homogenous nodes based on the parent feature or node.

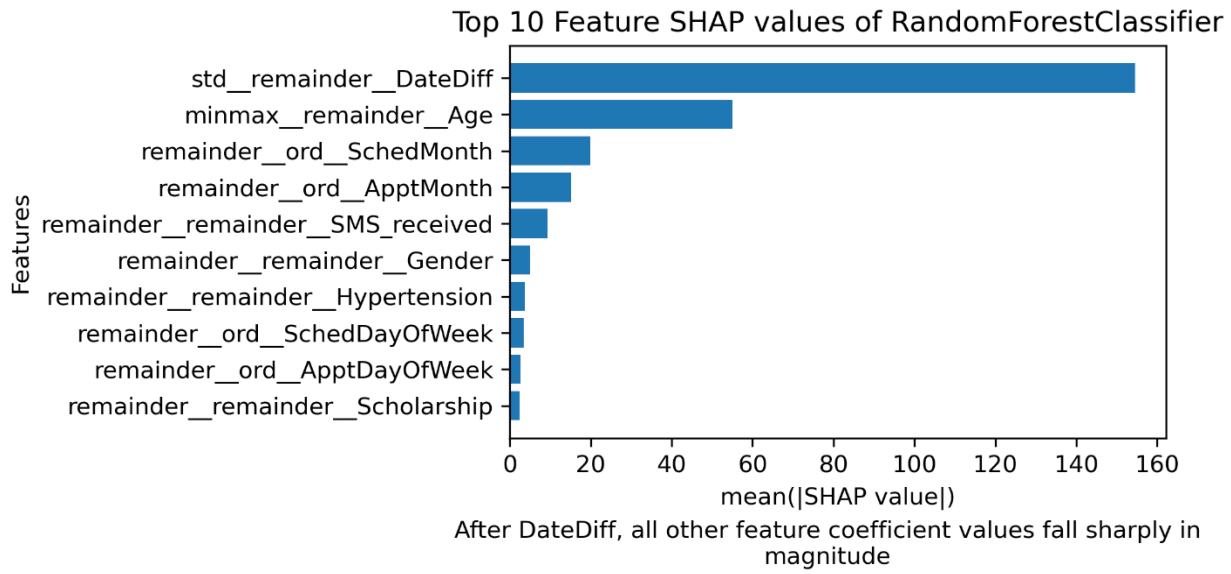

Figure 10. Top 10 feature importances by shap values

SHAP values are based on a "game theory" approach to assign an importance value to each feature in the model.

These metrics tend to show the same thing: DateDiff is clearly the most important feature and contributes the most to the model's predictions. The Age feature takes a distant second place. Other features fall sharply in terms of feature importance.

Though unshown, the least important features tended to be some combination of the one-hot encoded neighborhood categories.

Shap values can also be applied to measure local feature importance, i.e. measuring importance on individual data points.

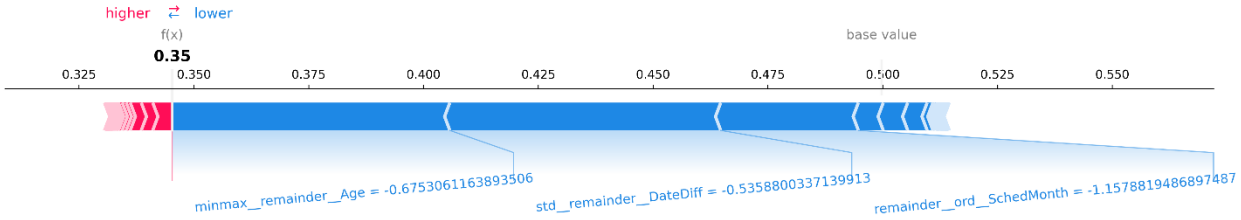

Figure 11. SHAP force plot for data point 1

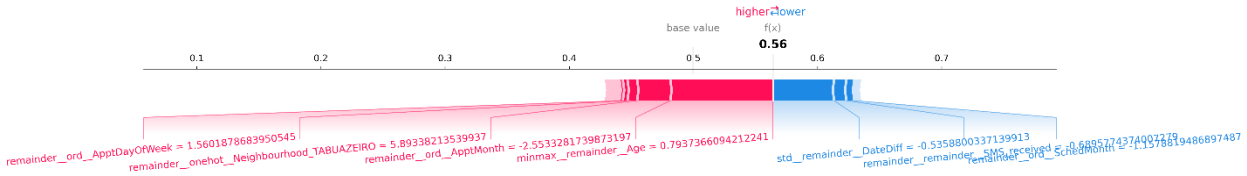

Figure 12. SHAP force plot for data point 42

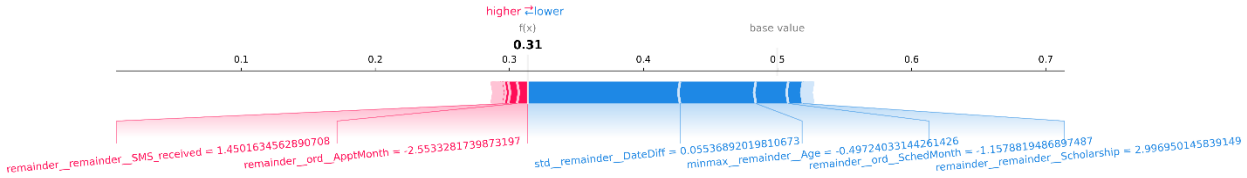

Figure 13. SHAP force plot for data point 96

These SHAP force plots predict the probability of an individual data point belonging to class 0, i.e. the patient shows up to the appointment. With a probability threshold of 0.5, the model classifies Figure 10 as a no-show, Figure 11 as a yes-show, and Figure 12 as a no-show.

5 Outlook

The best performing model was our Random Forest Classifier, if evaluated solely for F1 scores. Otherwise, it would be the XGB Classifier if evaluated for accuracy. Our Random Forest Classifier had the following parameter values:

class_weight	“balanced”
max_depth	10
max_features	0.5

For improvement I would try increasing the number of k-folds during cross-validation. Currently we use K=5 folds for the first split and K=4 folds for the cross validation. Increasing this would increase computational cost but may increase the accuracy of the results since the model is trained on more data.

Another avenue to explore is applying GPU acceleration to our XGBoost model. Doing so would decrease the amount of time needed to train each model. We can try more parameter values and potentially train a much better model.

Finally, I would further experiment with datetime column feature engineering. DateDiff was the most important feature, so researching new dimensions of the datetime columns may result in new, more relevant features.

6 References

[1] https://www.kaggle.com/datasets/samuelotiattakorah/healthcare-no-show-appointment-data

[2] Github repository - https://github.com/dwang0120/Hospital-No-Shows