CHLA No Show Predictor

By: Tommy Ropp

**Business Understanding:**

This project intends to predict whether or not patients will show up to an appointment at the Children’s Hospital Los Angeles (CHLA). CHLA is the #1 ranked children’s hospital in California and the Pacific region according to U.S. News. As of 2023 they admitted 14,600 patients annually and employed more than 6,000 people. Serving so many patients CHLA inevitably ran into the issue of patients not showing up for appointments, wasting vital hospital resources and people’s time. In response, CHLA began double booking appointments to ensure there was a patient in each available time slot. However, this had an adverse effect as when patients did show up for their appointments, they often had longer wait times.

To solve this problem CHLA questioned whether it was possible to predict patient no-shows to more effectively double book appointments. If this were possible CHLA could better allocate their resources and plan for no shows in advance. By seeing which patients had a higher likelihood of not showing up CHLA could also implement intervening measures to encourage those high risk patients to either reschedule or arrive on time thus improving no show rates overall.

**Exploratory Analysis:**

Github link: <https://github.com/troppster225/CHLA_predictor>

Deployed model: <https://chlapredictor.streamlit.app/>

A screenshot of a computer

AI-generated content may be incorrect.

Above is a first look at our data. To begin with we had 24 columns and 21,060 rows of cleaned data. However, in order to make our model more user friendly and get it ready for our model we had to pare down our columns and perform preprocessing. First we need to assess the cleanliness of the data to see if any missing value handling is necessary.

A screenshot of a computer

AI-generated content may be incorrect.

A graph of a schedule

AI-generated content may be incorrect.A blue bar graph with white text

AI-generated content may be incorrect.We can see from the table above that there are no missing values in the original dataset. Now we check the value counts of some of our features to get a better sense of their distribution.

A blue bar graph with white text

AI-generated content may be incorrect.

Here we see the majority of our appointments are follow ups with the status checked out. However, we can see that within our appointment status column there is a feature that specifies if that appointment was a no show, leading us to remove the feature. In terms of clinic distribution, we also wanted to get a sense for where the most no shows were coming from.

A blue pie chart with white text

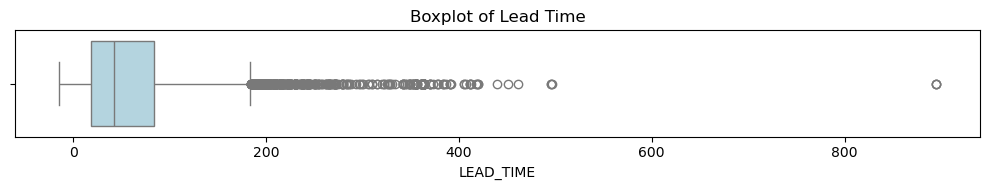
AI-generated content may be incorrect.

We can see that Valencia Care Center has the highest share of no shows compared to total visits despite not being the most visited clinic. It’s possible that clinic location is an important factor in whether or not an appointment is a no show. Next we’ll generate a report for all of our categorical variables.

A screenshot of a computer

AI-generated content may be incorrect.

We can see relatively low cardinality for our categorical variables besides APPT\_DATE, BOOK\_DATE, and ZIPCODE, all of which we remove. This means we don’t have to worry about combining values or creating overly complex trees that could result in overfitting. However, this is still an issue we’ll check on later. Let’s look at the distribution of some of our numerical values next.



Lead time seems to have a wide distribution with many outliers, including an extreme outlier pas the 800-day mark. To not skew our results we capped the lead time at 400 days. Solely based off intuition one could surmise that lead time will be an important factor in the final no show prediction.

A graph with dots and numbers

AI-generated content may be incorrect.

Our number of appointments feature seems to also be right skewed with a fair number of outliers. With a cardinality of 28 there doesn’t seem to be a huge range in the number of appointments that patients have had previously.

A screenshot of a graph

AI-generated content may be incorrect.

We can also see that there are some high correlations between the APPT\_NUM feature and other features in the different TOTAL\_NUMBER\_OF group of columns. Due to this, we removed the APPT\_NUM as this is described within those other columns.

Now that we understand our data’s individual features better, it’s time to start looking at our dataset as a whole and the interaction between different variables. First, we want to know the balance of our data.

A black background with white text

AI-generated content may be incorrect.

We can see that there is a clear imbalance in the data. Our data is no-where near the 70/30 split we desire for good training data. Some under sampling or oversampling will be required, something covered in a later section. In order to see the interaction between our variables, categorical features were encoded using one hot encoding due to their non ordinal nature.

A graph of a number of words

AI-generated content may be incorrect.

Above is a messy correlation heat map of all of our features. However, some important findings were still derived from this figure leading to the dropping of many columns. First as we see there is a 100% correlation between APPT\_STATUS\_NOSHOW and IS\_NOSHOW (our target variable). Due to this, APPT\_STATUS\_NOSHOW and all other APPT\_STATUS columns were dropped. Next, we noticed high correlations within the different race and ethnicity features. To ensure there was no undesired interaction between these columns and the prediction of our target variable they were dropped. By dropping these columns we were left with the following heatmap:

A chart with numbers and symbols

AI-generated content may be incorrect.

The interactions between independent variables aren’t as strong as before and we’ve paired them down to a manageable level. Following this we performed under sampling of our majority class, pairing the IS\_NOSHOW Y’s down to 3000 samples. In order to get closer to a 60/40 split, SMOTE was used to oversample our minority class, leaving us with the following distributions for our training and test sets:

A black screen with white text

AI-generated content may be incorrect.

**Model Performance Comparison:**

A screenshot of a graph

AI-generated content may be incorrect.

Three different tree-based models were used to predict patient no shows with their metrics listed in the above table. The reason tree-based models were selected over other classifying models in the first place was because of the nature of the problem. As we can see from our correlation table, none of the independent variables have a strong and clear linear relationship with our target variable, making it a fair assumption to think that these relationships are complex and highly dependent on the values of other independent features. A decision trees ability to branch from different values fits this problem logically as there are a variety of factors that lead to all of us needing to reschedule our appointments at some time. An ensemble of decision trees allows this tool to be more generalizable to the wide distribution of patients that CHLA serves.

From the table above, we can clearly see that RandomForest performed the best overall. GradientBoost did have a slight edge when it came to F1 score. However, RandomForest’s discriminatory power as shown by its AUC was by far the best of the bunch. RandomForest both catches fewer false positives and more actual positives than the other two models. For these reasons, RandomForest was the clear choice for our final model.

**Conclusion and Discussion:**

Ultimately our final model shows that it is relatively good at predicting no show appointments according to historical data. That means that at the moment the data was captured, we could predict no shows at an accuracy of 84% with an AUC of 84. However, with new unseen data and changing patterns in patients’ behavior, our model could start to drift. Once we deploy the model and put it into a production environment, constant monitoring and diligent MLOPs will be required to continue assessing its accuracy and ensure that it’s staying up to date with the ground truth.

One main observation I made with this model was how it tended to over classify Y’s. When inputting some test cases from the 2024 data, it was predicting some false positives that could lead to inaccurate double bookings. While this is still better than the methodology that the hospital was employing previously, these false positives will still be something to keep an eye on. Perhaps adjusting the probability threshold to more closely match reality could be one way to handle this issue.

When it comes to limitations of the model there are two main ones that come to mind. The first is a data limitation. Having more data like the type of appointment that patient was going in for (i.e. a physical, therapy, a surgery) could have a large effect on the no show probability. A previous group that worked on this model also had access to a Google Maps API that allowed for the calculation of commute distance from the patient’s zip code to the clinic. This information could have also potentially improved our model’s performance.

Besides a data limitation, I believe the other obvious limitation of the model is the randomness of human behavior. Of course, this is captured in the fact that a model will never be 100% accurate, but I do think this plays a bigger role in the reason that we are sometimes late to appointments or miss them altogether. Things come up and unavoidable, random events may impede in our ability to be on time constantly. This noise within the reasons behind no shows is completely unpredictable. So, it seems that just because a good deal of no shows are predictable, it doesn’t mean they will all be predictable.

**User Front End: A screenshot of a medical record

AI-generated content may be incorrect.**

**A screenshot of a computer

AI-generated content may be incorrect.**

**CHLA ML Model Development vs. Deployment:**

Part one of this project entailed the model development process. This involved data exploration, cleaning, feature engineering, and training multiple models. I also performed hyperparameter tuning and model comparison to get the most optimized model for our use case. All of these steps are different parts of the model development process which is iterative and continual. From this process we got a final serialized model that we could then take and put into a system that allows for users to interact with it through a front end. This process involved using streamlit to build an inferencing application with a backend architecture including input validation, data processing, and data validation. This process started offline with our app being hosted locally but eventually was deployed to the cloud through streamlit’s cloud database. In effect, we have operationalized our model through this part of the process, delivering a functional, user-ready product rather than just an accurate model.

One observation I have about this predictive application is its user friendliness. The streamlit interface makes the model easy to use for non-technical stakeholders and gives clear business insights that can be used to make the double-booking process for effective. However, a downfall of this model is its reliance on streamlit’s database and its susceptibility to drift. For future iterations, a better, more secure cloud solution would be better than streamlit’s free offering. To handle drift continual training and a good MLOps process should be implemented.

In terms of the limitations of this application I would point to both the customizability of streamlit and the features of the model itself discussed in the previous section. Streamlit comes with prebuilt functions that are plentiful for getting working versions of deployed models up and running fast. However, for larger scale, real-world environments where this application would be used, a more robust web framework potentially involving web development may be required to customize it to CHLA’s requirements. Introducing Kubernetes could also help with scalability by distributing server workload.

Overall, I learned how both the model development and model deployment processes work together to create a finalized production ready product. Building the best performing model both involved looking at using the model with the best metrics while also trying to maximize model interpretability and usability. The deployment process also involved some live data processing where the user’s input was used to filter a dataset and with the dataset then being encoded to match the training dataset. In all, part one helped me develop my technical ML skillset and part two broadened my perspective by showing how an ML model fits into a larger ecosystem of product development, monitoring, and user experience.