PREDICTING MEDICAL APPOINTMENT NO-SHOWS

Machine Learning Engineer Nanodegree Capstone

Dalton J Schutte

Table of Contents

DEFINITION

PROJECT OVERVIEW

The medical system in the United States is notorious for its high costs compared to that of other nations, in 2015 coming in at over USD 3.2 trillion (~USD 10,000 per person). A part of this cost is attributable to no-show appointments, when a patient fails to attend their scheduled medical appointment. One study estimated the costs of no-show appointments at USD 150 billion[1], about 4.6% of total costs.

This project will attempt to produce a machine learning model that can predict the likelihood of a patient not showing up for their appointment. Using a publicly available dataset of medical appointment no-shows, a machine learning model will be chosen through a rigorous series of experiments and trained using optimal hyperparameters to predict the likelihood of a patient not showing up for their appointment. This model could be deployed in web or mobile apps or integrated into the clinic's EMR to enable clinics to decide if additional reminders or follow-up are necessary with certain patients to reduce the likelihood of a patient missing their appointment. Ideally, this additional outreach would help clinics reduce the costs stemming from no-shows.

PROBLEM STATEMENT

The goal is to produce a machine learning model that delivers likelihood scores (ranging from 0 to 1). To accomplish this, the following steps are necessary:

1. Download and process the Medical Appointment No-Show data[2]
2. Conduct a series of experiments to find the best model for this dataset
    a. Evaluate nine out-of-the box learners from the scikit-learn package
    b. Perform a random search of the hyperparameter space for the top three performing models
    c. Evaluate the top three using the best found hyperparameter configuration
    d. Construct a Self-Regularizing Network architecture and find optimal hyperparameters
    e. Evaluate the SNN using the hyperparameters
    f. Select best performing of the three sklearn models and the SNN models
3. Train the chosen model using the found configuration

METRIC

While accuracy is a common metric for training binary classifiers, and was initially the proposed metric, the decision was made to instead use the area under the receiver operator characteristic curve (ROC AUC). The ROC AUC gives a better sense for the balance between a model's completeness (ability to catch positive cases) and discriminability (ability to differentiate between a positive case and a negative case when assigning a "positive" prediction). A model that strikes a good balance between completeness and discriminability is desired since a model that has poor completeness won't identify sufficiently many patients as potential no-shows and thus not address the fundamental problem and a model that has poor discriminability results in clinic staff reaching out to patients who may be low risk for missing an appointment and thus wasting their time.

The ROC AUC is calculated by first plotting the true positive rate and false positive rate of the classifier at various thresholds. So, first all predictions that are 0.0 are assigned to class 0 and all at or above 0.0 are assigned to 1, then

[1] https://www.scisolutions.com/uploads/news/Missed-Appts-Cost-HMT-Article-042617.pdf

[2] https://www.kaggle.com/joniarroba/noshowappointments

all predictions that are 0.01 or below are assigned to 0 and all at or above 0.01 are assigned to 1, end this process is repeated by incrementally increasing the threshold until the threshold is at 1. Then the area under this curve is calculated to produce the ROC AUC.

Analysis

Exploratory Data Analysis

The Medical No-Show dataset is comprised of 110,527 medical appointments scheduled in Brazil. The dataset contains 14 features:

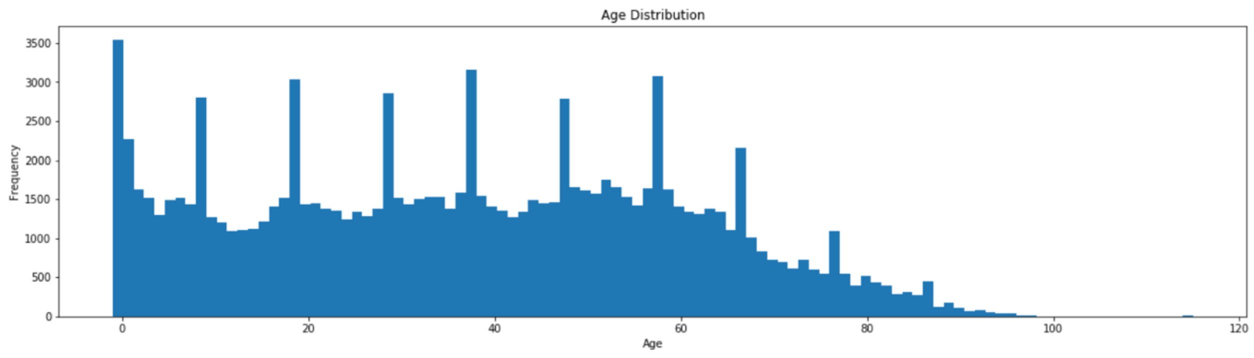
- **PatientId:** Unique ID number assigned to each patient
- **AppointmentID**: Unique ID number assigned to each appointment
- **Gender:** (M/F) indicates the patient's gender
- **ScheduledDay:** date the scheduling was completed
- **AppointmentDay:** date of appointment
- **Age:** age of the patient
- **Neighbourhood:** neighborhood the appointment is in
- **Scholarship:** Boolean, indicating participation in a social welfare program
- **Hypertension:** Boolean, indicating if patient has hypertension
- **Diabetes:** Boolean, indicating if the patient has diabetes
- **Alcoholism:** Boolean, indicating if the patient has a history of alcoholism
- **Handicap:** indicating the degree of handicap for a patient
- **SMS_Received:** Boolean, indicating if the patient received a text reminder for the appointment
- **NoShow:** (Y/N) indicating if the patient failed to attend their scheduled appointment

There were no missing values in any of the rows. There were several entries where the age was given as 0, and it was assumed that these were newborns under a year of age. Overall, the dataset was remarkably clean and complete.

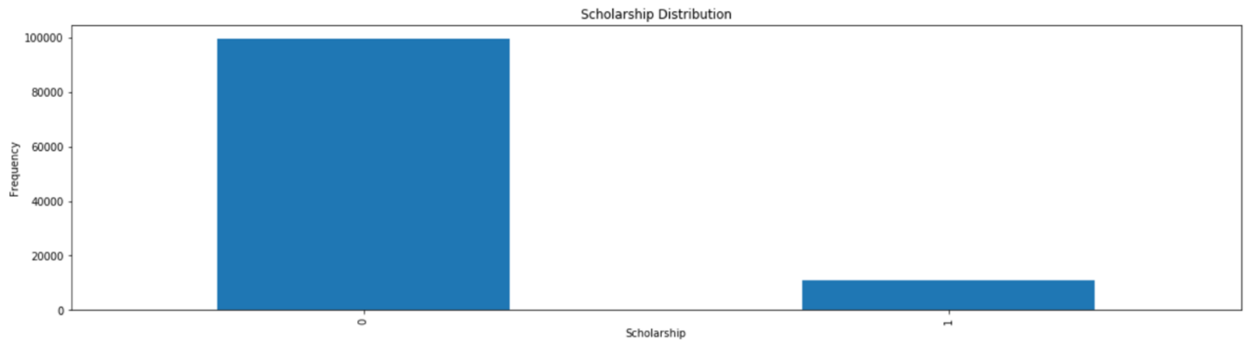
Data Visualization

Distributions were constructed for each of the variables and a correlation plot was made using Cramer's V for categorical variables. ID fields and date fields were not visualized. Features that were visualized but not used for training the model, discussed further below, are not included below but can be found in Appendix A. Excluded features were Gender and Neighborhood.

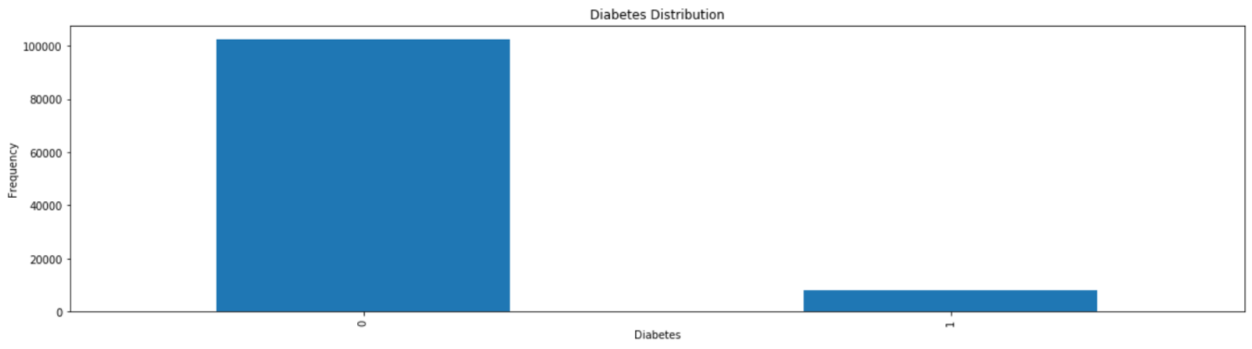
The plot below shows the age distribution. Under the age of 60, the distribution is fairly uniform and gradually tapers off as age increases. The mean age is 37 with a standard deviation of 23 years.



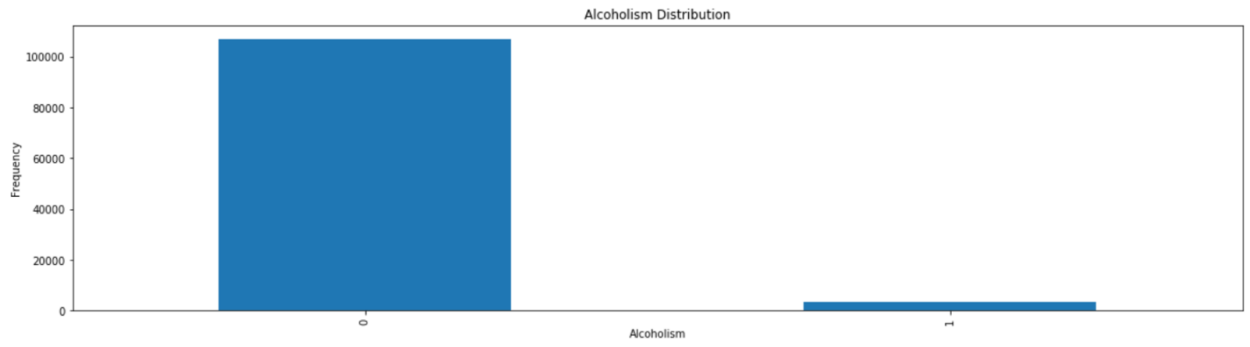
The plot below shows the distribution of patients receiving social welfare. The majority of patients (90.17%) are not receiving benefits.



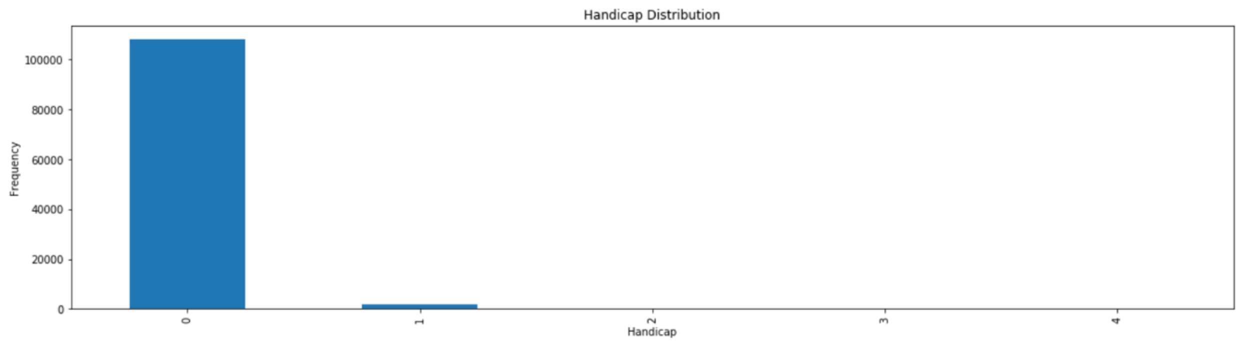
The plot below shows the distribution of patients with diagnosed diabetes. The majority of patients (92.81%) do not have a diabetes diagnosis.



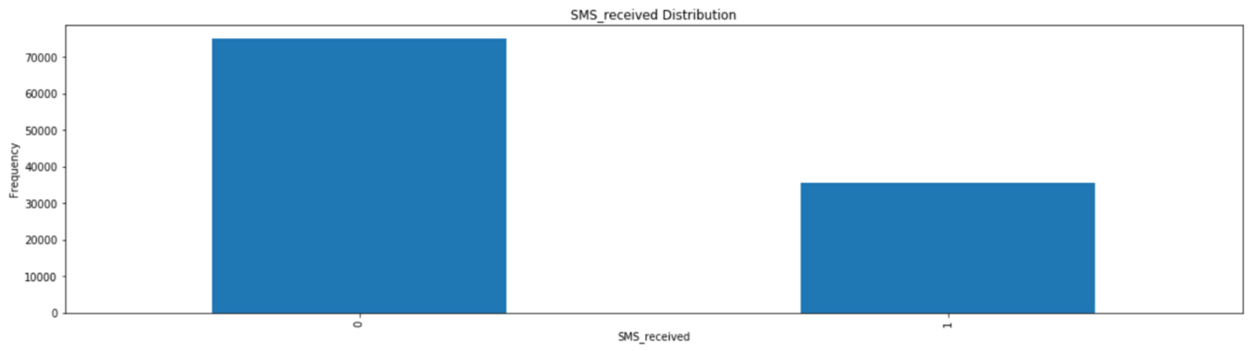
The plot below shows the distribution of patients with a reported history of alcoholism. The majority of patients (96.96%) did not report a history of alcoholism.



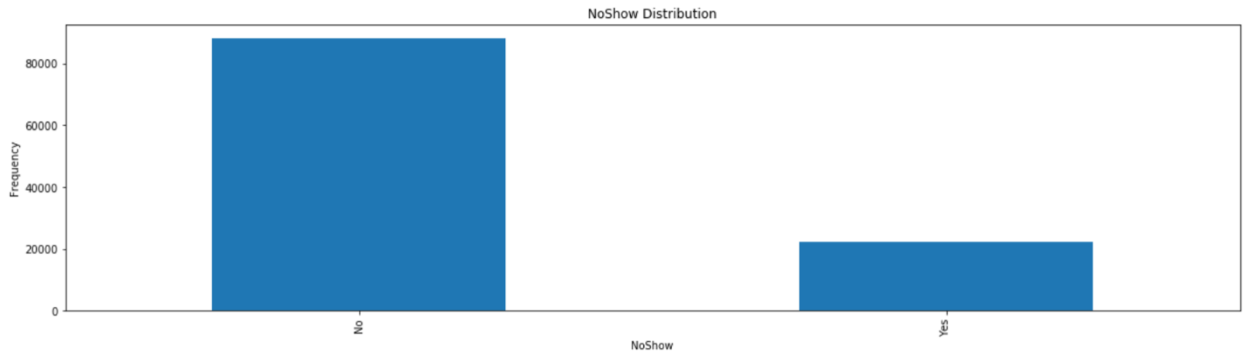
The plot below shows the distribution of diagnosed handicaps amongst patients. The majority of patients (97.97%) do not have any degree of diagnosed handicap.

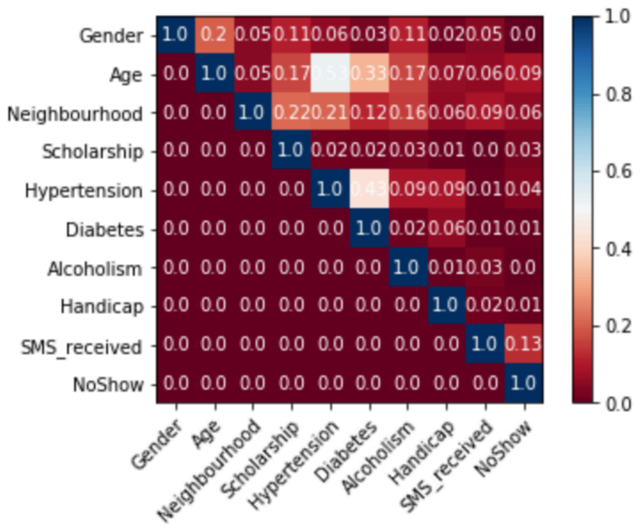


The plot below shows the distribution of patients that received a text reminder of their appointment. The majority of patients (67.90%) did not receive a text reminder.



The plot below shows the distribution of patients who did not show up for their appointment. The majority of patients (79.81%) made their scheduled appointment. This demonstrates that the dataset is imbalanced in favor of patients who made their appointment and this imbalance will be addressed prior to modeling.





The figure to the left is a correlation matrix between all of the features. Due to the presence of categorical variables, it was necessary to compute the degrees of association using Cramer's V. It can be seen that none of the features are particularly strongly associated with any other. The highest degree of associations are between age and hypertension (0.53), which one would expect as hypertension is often a disease that presents later in life as arteries lose some of their elasticity, and hypertension and diabetes (0.43), which is also not surprising since diabetes can cause atherosclerosis which can lead to hypertension. It should also be noted that there are no strong associations between any features and the outcome variable, NoShow. Due to the lack of any strong associations, no features were excluded on the basis of strong associations.

Data Exclusions

Due to concerns about implicit biases being learned from gender, the feature was excluded from the dataset. Neighborhood was also excluded due to the neighborhoods all being in Brazil. Using neighborhoods in Brazil as a predictive feature limits the usefulness of the model outside of those regions. Excluding it allows the model to be used in the United States, for example, without impairing the model.

Algorithms and Techniques

The final classifier was chosen from one of several model types. From scikit-learn, the initial models were: Stochastic Gradient Descent classifier,[3] Elastic Net classifier[4], Logistic Regression classifier[5], Gaussian naïve Bayes classifier[6], Random Forest classifier[7], Gradient Boosting classifier[8], Linear Support Vector Classifier[9], k-Nearest Neighbors

3 https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html?highlight=sgdclassifier#sklearn.linear_model.SGDClassifier

4 https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.ElasticNet.html?highlight=elasticnet#sklearn.linear_model.ElasticNet

5 https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html?highlight=logistic%20regression#sklearn.linear_model.LogisticRegression

6 https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.GaussianNB.html?highlight=gaussiannb#sklearn.naive_bayes.GaussianNB

7 https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html?highlight=random%20forest#sklearn.ensemble.RandomForestClassifier

8 https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.GradientBoostingClassifier.html?highlight=gradient%20boosting#sklearn.ensemble.GradientBoostingClassifier

9 https://scikit-learn.org/stable/modules/generated/sklearn.svm.LinearSVC.html?highlight=linearsvc#sklearn.svm.LinearSVC

classifier[10], and Nearest Centroid classifier[11]. All of the models were evaluated using the default settings specified in the scikit-learn documentation. Additionally, a Self-Normalizing Network [12]was considered since it demonstrated good results on tabular data in the original paper. The SNN was trained using Stochastic Gradient Descent with Nesterov momentum and a One-Cycle Learning Rate scheduler[13].

Stochastic Gradient Descent Classifier

The Stochastic Gradient Descent classifier works fitting regularized linear models with stochastic gradient descent (SGD). As with the SGD optimizer for neural networks, learning occurs by computing the loss after each mini-batch and used to update the model after each mini-batch. The base model from scikit-learn uses a linear SVM, hinge loss, and L2 penalty. It learns by fitting a model over mini-batches to minimize the loss. Predictions are made with a model using the parameters that were learned during optimization. This model was included for testing since it is using a similar optimization method to the SNN, but with a simpler estimator, and if the SNN fails to do well, it would be worthwhile to see if the simpler model performs well.

Elastic Net

Elastic Net is simply linear regression with L1 and L2 regularizers used together. This has a regularizer of the form:

$$\sum(\alpha|\beta| + (1-\alpha)\beta^2)$$

The second term encourages highly correlated features to be averaged and the first term encourages a sparse solution in the coefficients of the averages. This helps avoid the individual pitfalls of L1 and L2 regularizations. As such, the Elastic Net makes predictions the same way a linear classifier does and learns in a similar manner. However, the objective function is slightly modified to use the above regularization. This was included since it is a variant of linear regression that typically results in improved performance.

Logistic Regression

Logistic regression is one of the classic models and in the binary case is simply a single linear function that is fit, typically, using maximum likelihood. Predictions are made simply by applying the fitted linear function to the input. Being such a well-understood and simple model, it was included in these experiments.

Gaussian Naïve-Bayes

Naïve Bayes is named because it, naïvely, assumes that the features are all independent. The Gaussian variant takes this further and assumes that the continuous features are normally distributed. The Bayesian model is fit to the data, and a classifier is obtained by applying the maximum *a posteriori* decision rule. This rule classifies input based on what the most probable result is given the data distribution the model has seen during training. This was included since, despite its unrealistic assumptions, it often produces results as good as or better than more complex models.

Random Forest

The Random Forest classifier fits a series of decision trees to various subsets of the training data and makes predictions, in the classification case, by assigning a class based on the mode of the predictions of all of the fitted decision trees in the forest. This technique, while being more computationally expensive than a single decision tree, has the advantage of being robust to overfitting since many decision tree classifiers are being considered rather than

[10] https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html?highlight=kneighbors#sklearn.neighbors.KNeighborsClassifier

[11] https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.NearestCentroid.html?highlight=nearestcentroid#sklearn.neighbors.NearestCentroid

[12] https://arxiv.org/abs/1706.02515

[13] https://arxiv.org/pdf/1803.09820.pdf

one. It is typically the case that an ensemble of machine learning algorithms produces a better result than a single algorithm and that observation is capitalized on in the case of Random Forests. This model type was included since ensembles typically have good performance and Random Forests in particular allow for extraction of feature importance which gives a degree of explainability often absent from other models.

Gradient Boosting

Gradient Boosting uses a series of weak learners, decision trees in this case, fitted in a greedy and stage-wise approach to model the data by using the steepest descent along the gradient of the loss function. The general algorithm consists of initializing a constant model, a single node tree in this case, computing the residuals for the model with respect to the negative gradient of the loss function, finding a constant that minimizes the sum of the losses, using that constant to define an additional model, then adding it on to the series of constant models. The end result is an additive model comprised of simple, constant models. It makes predictions by feeding the features through the additive series to produce a class prediction. Boosting algorithms, gradient boosting in particular, have seen huge success in Kaggle competitions and elsewhere which was a primary driver behind their inclusion in these experiments.

Linear Support Vector Classifier

In general, Support Vector Machines (SVM) use a kernel function, such as a radial basis function, polynomial, or other function, and attempt to find a form of the kernel function such that the hyperplane defined by the function separates the classes by the largest possible margin. In the case of a linear SVM, the algorithm attempts to find a linear function that separates the classes of the training data by the largest possible margin. To make predictions, the model assigns a class based on which "side" of the function the datapoint falls on. For example, if the input is falls "above" the function and the class associated with the half-space above the function is "No Show" then the model predicts the input is "No Show'. SVMs have enjoyed success and are a standard, and effective, algorithm and so their inclusion in this study was natural.

k-Nearest Neighbors

The k-Nearest Neighbors algorithm is an intuitive, non-parametric model that, at a high-level, works by assigning a class to a datapoint based on the "vote" of the k closest datapoints. For example, in a 7-NN algorithm, the input datapoint is plotted, the 7 closest points are gathered, and their classes counted, the class with the highest count becomes the class of the newly plotted datapoint. However, there can be issues when the classes are imbalanced since the mere presence of a larger ratio of one class increases the likelihood of any random point being assigned to that class so care must be taken in these cases by either weighting the voting power of different classes or balancing the dataset. This model was included for its simplicity and the potential issue of class imbalance was not a concern since techniques were used to balance the dataset.

Nearest Centroid

The Nearest Centroid algorithm is somewhat similar in concept to the k-NN algorithm. The training set is plotted and the centroid for each class, the mean location in space of all datapoints belonging to that class, is calculated. Predictions are made by plotting the input data and assigning the datapoint to the class of the nearest centroid. This can be thought of as 1-NN where all of the datapoints for each class are replaced by a single point for each class. This was included also for its simplicity, but also because using a single centroid as the predictor rather than k nearest datapoints prevents any potential issues that might not be resolved by the class balancing techniques.

Self-Normalizing Network

A self-normalizing network is a type of neural network model that “self-normalize” using a special activation function that naturally converges the neural activations at each layer to zero mean and unit variance. These are proven to be robust to noise and perturbations. The structure of a SNN allows for deeper networks to be trained using stronger regularization methods to produce robust, abstract representations of data. The SNN, as with many neural models, is trained using some variant of gradient descent. In this case, stochastic gradient with momentum was used since it has been shown to produce models that generalize better than those trained using adaptive gradient methods. Predictions are made by feeding the input into the neural network and applying a sigmoid function to the output and assigning the class based on the desired threshold. In this case, 0.5 was used as the minimum threshold to assign an input to class 1 (No Show). This model was included since it demonstrated state of the art performance on tabular datasets in the original paper and provided a very structured architecture with fewer hyperparameters than a vanilla feed-forward network.

Class Imbalance

Due to the imbalance in the outcome feature, the imbalanced-learn[14] package was used to balance the dataset. The SMOTEENN class was used, which first uses SMOTE to generate more instances of the minority class then uses Edited Nearest Neighbors to remove samples that do not “agree” enough with their neighborhood to produce a balanced dataset with synthetic samples that are more closely located to its class cluster.

Hyperparameter Search

The top three scikit-learn models were configured by searching over the hyperparameter space using the scikit-learn RandomizedSearchCV[15] algorithm to find the best performing configuration with respect to ROC AUC. The optimal hyperparameter configuration for the SNN was found by using the hyperopt library[16] to search over the hyperparameter space outlined in the paper for the models trained on the HTRU2 dataset. The hyperopt library uses the Tree of Parzen Estimators[17] technique to find an optimal hyperparameter configuration.

Benchmark

Since there is no published benchmark for this particular problem, a benchmark was established by using the top performing model from the first set of evaluations prior to hyperparameter optimization. This ended up being the Random Forest classifier which achieved a mean ROC AUC of 0.803 over seven runs.

Methodology

Data Preparation and Engineering

The data was imported and cleaned so that all the original features except for gender, neighborhood, patient ID, and appointment ID remained. No other preparation or cleaning was necessary since the dataset was complete and very clean.

Data engineering included feature creation and scaling. Using the AppointmentDay feature, the day of the week was extracted and stored as appt_day as a categorical variable. AppointmentDay and ScheduledDay were used to calculate the number of days wait time between when the patient scheduled the appointment and when the

14 https://imbalanced-learn.readthedocs.io/en/stable/

15 https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.RandomizedSearchCV.html?highlight=randomizedsearch#sklearn.model_selection.RandomizedSearchCV

16 https://github.com/hyperopt/hyperopt

17 https://papers.nips.cc/paper/4443-algorithms-for-hyper-parameter-optimization.pdf

appointment was set to occur and was stored as days_wait. Then AppointmentDay and ScheduledDay were discarded. The PatientId was used to search for duplicates and counted to produce prev_visits which represents the number of previous appointments the patients had scheduled. PatientId was also used to count the number of times a patient failed to show up for an appointment and was stored as prev_noshows, PatientId was then discarded as was AppointmentID since IDs are often randomly generated sequences of numbers that do not contain useful information outside of identification. A ratio of prev_noshows to prev_visits was calculated and stored as pct_noshows to represent the frequency with which a patient failed to show up to their past appointments.

Once the feature creation was completed, all features were scaled using the scikit-learn StandardScaler[18] class. This goes through each feature, sets the mean to zero, and scales the variance to one. The process of scaling has been shown to help improve model learning.

Model Experiments

Scikit-Learn Models

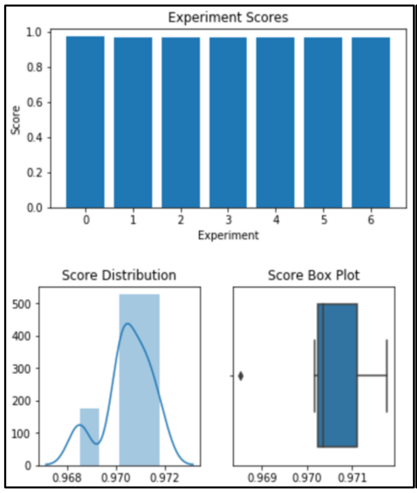
Each of the nine scikit-learn models were subjected to seven experiments on 20% of the unbalanced data to enable quick testing. Each experiment included balancing the data using SMOTEENN, splitting into training and testing sets (70/30), instantiating, fitting, and scoring the model. This process was repeated seven times, each time with a unique random seed with the same set of seven random seeds being used for each of the models. The top three models were chosen based on their average ROC AUC scores from the seven experiments. These ended up being the random forest, gradient boosting, and k-NN models. The results for the eight models can be found in Appendix B.

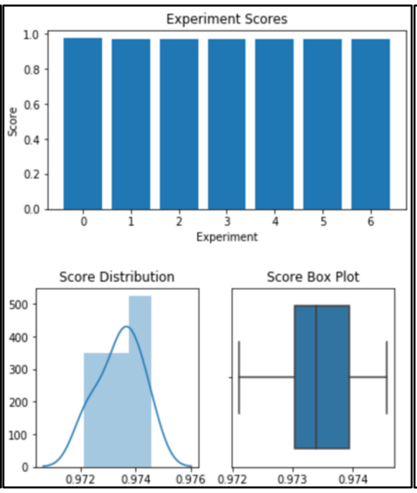
For each of the top three models, four to six parameters were chosen, and parameter spaces were defined for each model. Each of the model's hyperparameter spaces were searched using the scikit-learn RandomizedSearchCV which conducts a random search of the defined hyperparameter space. The number of configurations for each model was set as 15 configurations per parameter included for search. The table below summarizes the best-found parameters for each model after the random search was completed.

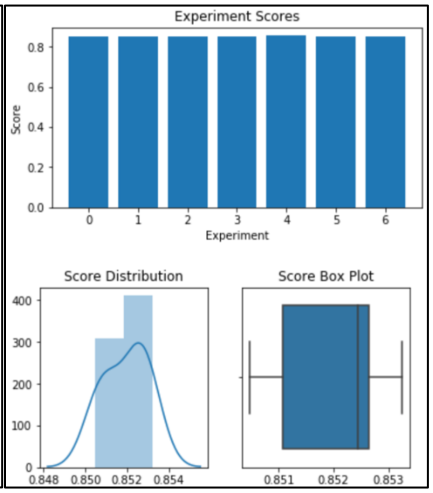
Model	Parameter Name	Value
Random Forest	n_estimators	83
	min_samples_split	3
	min_samples_leaf	2
	max_features	'sqrt'
	max_depth	9
	criterion	'entropy'
Gradient Boosting	subsample	0.8889
	n_estimators	114
	min_samples_split	2
	min_samples_leaf	6
	max_depth	2
	learning_rate	0.2334
k-Nearest Neighbors	weights	'uniform'
	n_neighbors	9
	leaf_size	21
	algorithm	'ball_tree'

[18] https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html

Each of the models were then trained using the same procedure that was used for the first round of experiments, but the parameters were set to the hyperparameters above. The results are shown below.







Random Forest Gradient Boosting k-Nearest Neighbors

The Gradient Boosting model achieved the highest mean ROC AUC of 0.973, then the Random Forest model with a mean ROC AUC of 0.971, then the k-NN with a mean ROC AUC of 0.852. All models obtained better scores than the benchmark.

SNN Model

The SNN model was subjected to the hyperparameter search before being trained fully. This was done to assess the viability of a deep learning model for this particular task. Since a SNN model is not available in scikit-learn, one had to be made using PyTorch which made the pipeline for automated testing that was used for the scikit-learn models not applicable to the SNN model, which was another reason for it being sent straight for hyperparameter optimization rather than being included with the other nine models for a first round evaluation. The SNN model uses embedding layers for each of the categorical variables, with embedding dimension equal to half the number of categories rounded up. Each continuous variable is fed into the model normally. Dropout for the embedding layers was set to 0.05 and for the remaining layers to 0.03. Batch normalization was used to aid in regularization. At inference time, a sigmoid function will be applied to the output of the network to produce a likelihood value between 0 and 1. A one cycle learning rate scheduler was used to facilitate convergence and model training was stopped if no decrease in the validation loss compared to the prior best validation loss occurred after three consecutive epochs.

The best found hyperparameters for the SNN model are outlined in the table below.

Model	Parameter Name	Value
Self-Normalizing Network	lr (learning rate)	0.00274
	momentum	0.656
	network_shape	'conic
	num_layers	16
	first_layer_width	256

Due to the intense time requirements of training the SNN, it was decided to train it a single time to compare against the top three scikit-learn models rather than averaging the performance over seven experiments. The SNN obtained a ROC AUC of 0.903 after five epochs and had stopped after five epochs since there had not been a decrease in the validation loss for three consecutive epochs. The model failed to outperform the Gradient Boosting model.

Implementation

The main implementation phases are the experiment and model selection phases. The main processes can be described as:

1. Select nine scikit-learn models
2. Perform seven quick train-test experiments for each model on a subset of the data
3. Select the top three models based on mean ROC AUC
4. Perform random search of each model's hyperparameter space
5. Perform seven train-test experiments for each of the three configured models on the full dataset
6. Select the top model based on the mean ROC AUC
7. Perform hyperparameter optimization for the SNN model
8. Train the SNN model using the optimized hyperparameters
9. Select the top model between the trained SNN and trained scikit-learn model

Refinement

Refinement was naturally integrated into the process described above. Searching the hyperparameter space, via random search or optimization, produced refined models compared to the models used in the initial round of experiments. For example, the Gradient Boosting model improved from a mean ROC AUC of 0.784 in the initial experiments to a mean ROC AUC of 0.973 in the final round of experiments. The Random Forest and k-Nearest Neighbor models saw similar performance increases of 0.803 to 0.971 and 0.742 to 0.852, respectively.

Conclusion

Model Evaluation and Validation

The final Gradient Boosting model has the parameters described in the table below.

Parameter	Value
subsample	0.8889
n_estimators	114
min_samples_split	2
min_samples_leaf	6
max_depth	2
learning_rate	0.2334

A subsample value less than 1 results in a version called Stochastic Gradient Boosting which trains each base learner using a portion of the available training data which, similarly to SGD for neural networks, reduces the variance of the updates.

The number of estimators, or boosting stages, that are used when fitting the model was set to 114 estimators. Increasing the number of estimators increases the ability of the model to fit the data (i.e. increase the representational capacity) which can produce a more accurate model.

The minimum samples per split sets the minimum required number of samples for an internal node to split and was set to 2. The higher the number, the more samples are required to produce a split. The trade-off being that more samples required to split the node results in a model less prone to overfitting but also less able to capture minute features of the data.

The minimum samples per leaf sets the minimum required number of samples for a node to be considered a leaf node and was set to 6. This also has the effect that even if there are sufficient samples for a split, if there are not enough samples for the leaf then the split will not occur. For example, if there are 10 samples at a node it will split, but if the split is 5 to each potential leaf then the split will not occur since the minimum samples per leaf is not met. This, similarly, has the tradeoff of decreasing the likelihood of overfitting as the minimum increases but also losing the ability to capture fine details.

The max depth sets the maximum number of layers that are allowed in each estimator and is set to 2. This means that each estimator tree can have at most two layers of nodes.

The learning rate scales the contribution of each estimator and is set to 0.2334. A learning rate less than 1 effectively engages a regularization technique called "shrinkage" and improves the generalizability of the model. However, this trades-off by increasing the computation time especially as the number of estimators increases.

By using the mean performance over multiple experiments, the generalizability of the model is demonstrated, and a low standard deviation suggests some degree of robustness. Thus, the Gradient Boosting's mean ROC AUC of 0.973 and standard deviation of 0.001 suggests that the model generalizes well and is fairly robust.

Suggestions for Improvement

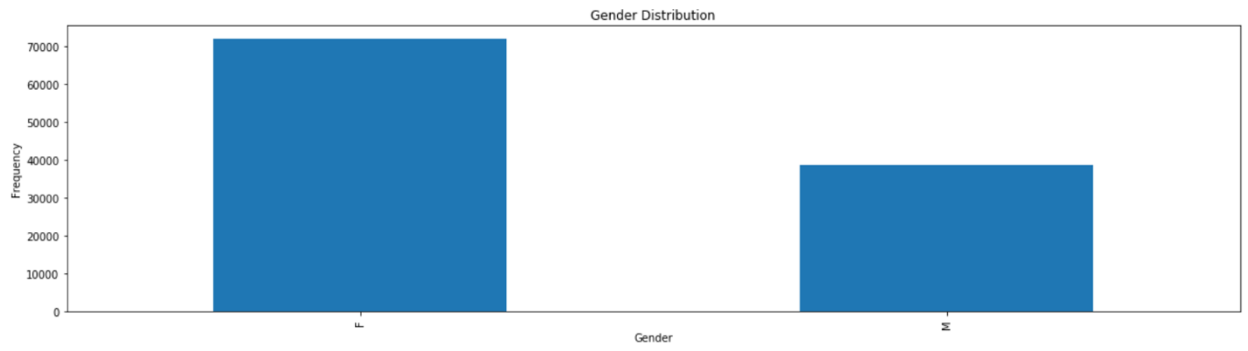
To further improve on this approach, more models could be considered for the initial experiments and more selected for hyperparameter optimization. The hyperparameter spaces could be expanded by adding more parameters and/or increasing the number of potential values within the parameters. Additionally, Bayesian Optimization methods could be applied to the hyperparameter search to decrease the number of configurations that need to be tried before a good configuration is found.

If more compute power is available, more experiments could be conducted, and models could be trained for longer and more quickly.

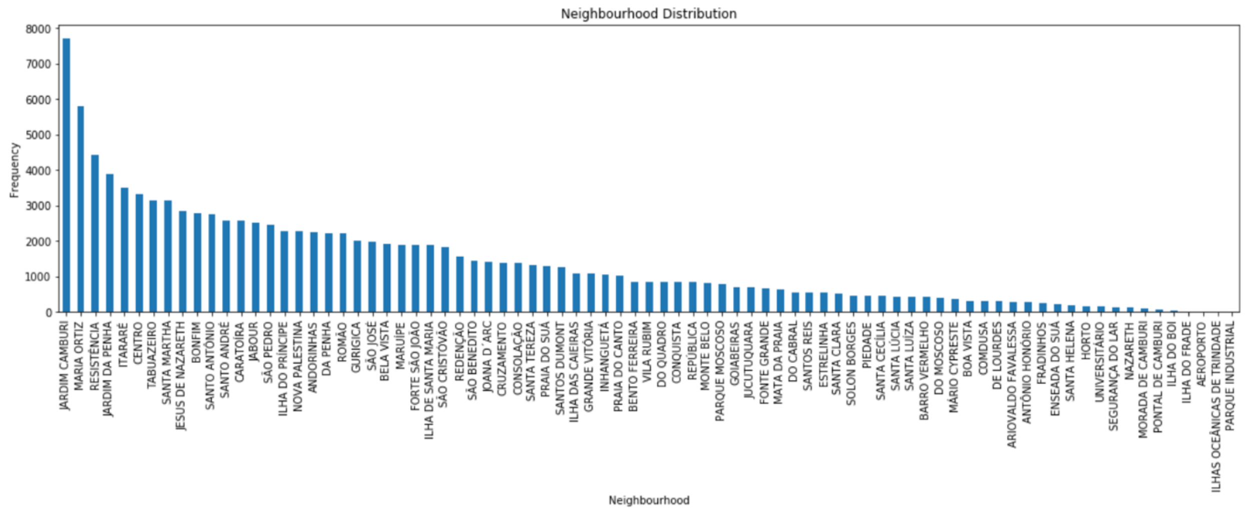
APPENDICES

APPENDIX A – EXCLUDED FEATURE DISTRIBUTION

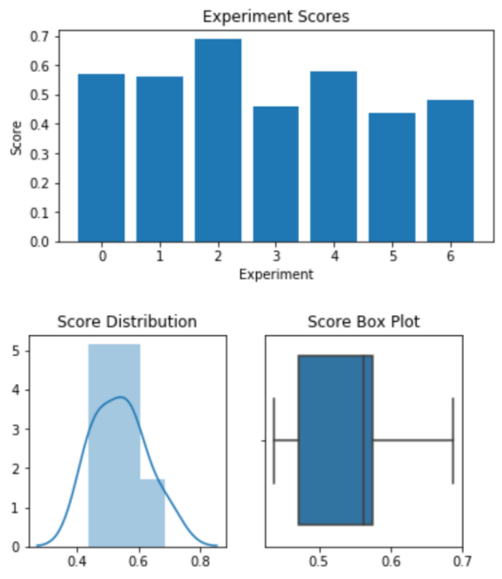
Gender Distribution



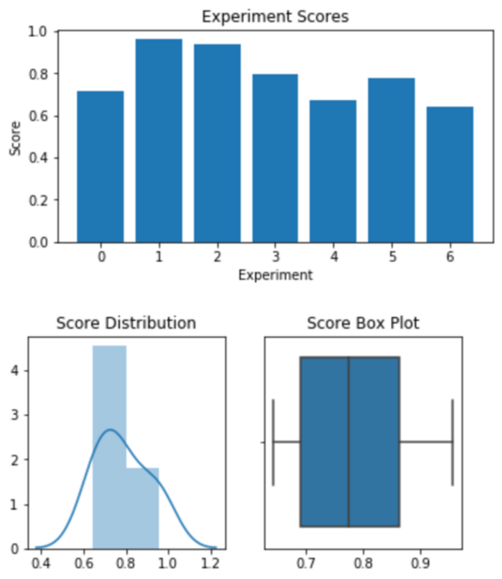
Neighborhood Distribution



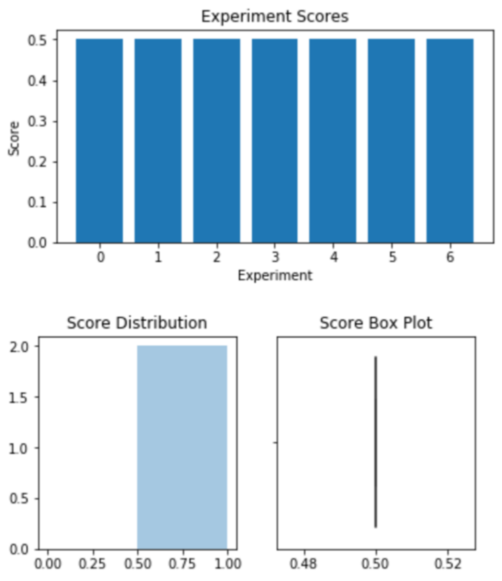
APPENDIX B – FIRST ROUND MODEL EXPERIMENT RESULTS



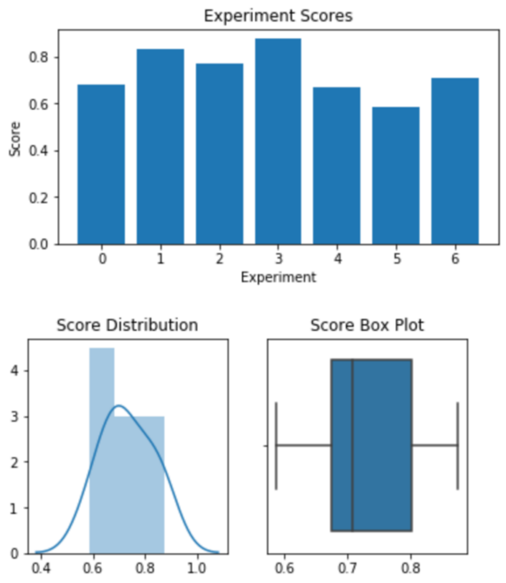
Nearest Centroid



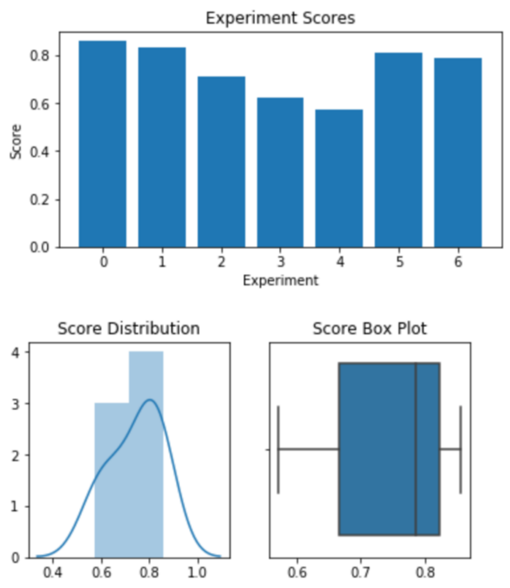
Gradient Boosting



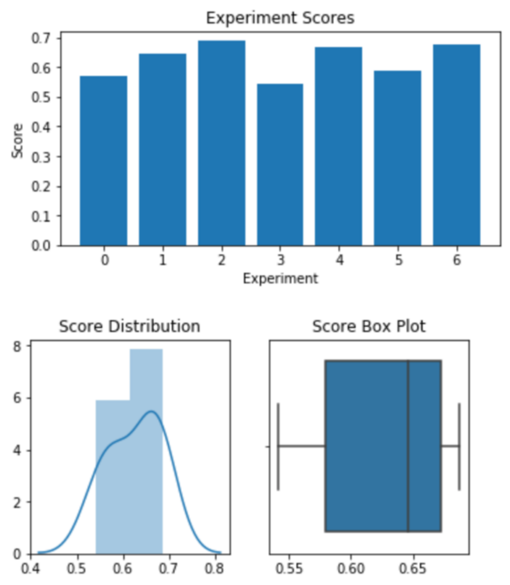
Elastic Net



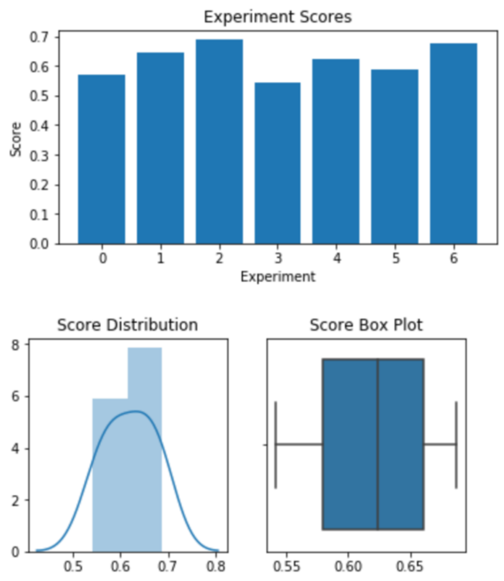
Gaussian naïve Bayes



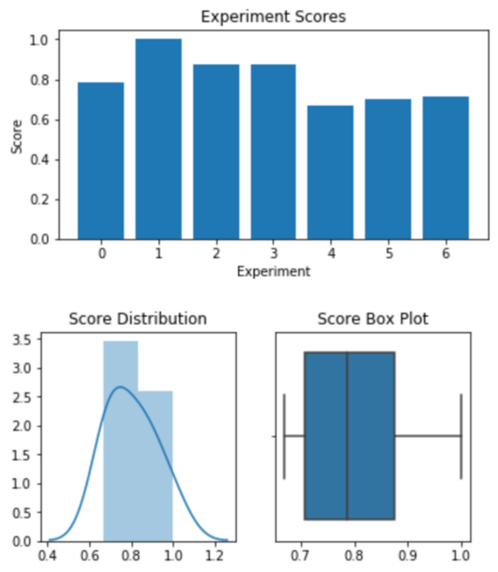
k-Nearest Neighbors



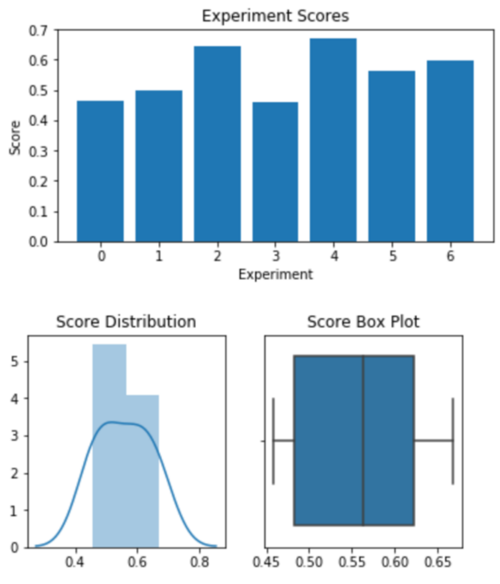
Logistic Regression



Linear SVC



Random Forest



Stochastic Gradient Descent