Machine Learning Engineer Nanodegree

Capstone Project

B Indrajith Shetty
August 1st, 2018

Predicting **"Medical Appointment No Shows"** using Machine Learning

I. Definition

Project Overview

Domain

Every day clinics and hospitals across the world have to deal with people missing their appointments. In Brazil, it is estimated that nearly 30% of the people that make a medical appointment end up missing their appointment. This causes a lot of time (of medical experts) and other resources to be wasted. Apart from that, it also leads to unnecessarily long appointment wait times for other patients. This is a significant problem for patients as well as medical professionals.

To combat the issue described above, historically and currently, clinics and hospitals slightly overbook, assuming that some of the appointments will be No Shows. If we had a way of knowing possible No Shows before the appointment date, it would be very beneficial for everyone involved. In this project I try to predict if an appoint will be a No Show or not. This prediction will be very useful to decide when to overbook appointments.

Project Origin

The project idea and dataset have been acquired from a Kaggle competition, in the following link:

- https://www.kaggle.com/joniarroba/noshowappointments/home

Datasets

For the Predictive and exploratory analysis, I am using the dataset provided in the project's Kaggle link. This dataset is in a single csv file. It is collected from various medical appointments in Brazil. This record contains information of 110527 appointments. Each record has 14 characteristics detailing various aspects of the appointment.
Data Set description:

- Shape: 110527 x 14
- Columns:
    - PatientId: Identification of a patient

- AppointmentId: Identification of each appointment
- Gender: Male or Female
- ScheduleDay: The day someone called or registered the appointment
- AppointmentDay: The day of the actual appointment
- Age: age in numbers
- Neighbourhood: Area where the hospital is located
- Scholarship:
  - This says If the patient has medical scholarship or not.
  - Details: https://en.wikipedia.org/wiki/Bolsa_Fam%C3%ADlia
- Hipertension:
  - Patient has Hypertension or not
- Diabetes:
  - Patient has diabetes or not
- Alcoholism:
  - Alcoholism observed in patient (True or False)
- Handcap:
  - Patient Handicapped (True or False)
- SMS_received: 1 or more SMS sent to the patient about the appointment.
- No-show: if the patient showed up or not. This is the feature being predicted.

Problem Statement

The problem being solved here is a binary classification problem. I need to classify each appointment as either 'No-show' (True) or 'show'(False).

My solution to this problem is to use Machine Learning algorithms (for binary classification). Solution steps:

- I will create a binary classifier.
- Clean and preprocess the dataset to be suitable for the model.
- Split the data into training, validation and testing data sets and use the train set to train the model.
- Once trained, I will test the model on validation data set using various performance metrics like accuracy, F1-score and ROC characteristics.
- Particularly, I am interested in increasing the 'recall_score' and AUC.

Metrics

I will be using 4 evaluation metrics:

- Accuracy
    - Accuracy = Number of correct predictions/Total predictions
    - (TP+TN)/(TP+TN+FP+FN)
- F1-score: Since I want a measurement which considers recall as well as precision, I will use F1-score.
    - This is the harmonic mean of 'recall' and 'precision'.
        - 2*precision*recall / (precision + recall)
        - F1-score is a good balance between the recall and precision scores. Since it is the harmonic mean, it leans more towards the lower score among 'precision' and 'recall'.
    - Recall = TP/(TP+FN)
        - No-shows detected among all the No-shows
    - Precision = TP/(TP+FP)
        - True No-shows among all predicted No-shows
- Recall_score:Since I want to detect most of the 'No-show's, this metric will be useful
    - Recall= TP/(TP+FN)
- AUC (ROC): This measure is used here due to the possible uneven distribution of Predicted labels for this data set and the binary nature of the predictions.
    - This is the area under the ROC curve. More on this can be studies at the link below.
    - https://en.wikipedia.org/wiki/Receiver_operating_characteristic
- Note:
    - TP= True positive
        - No-shows among the appointments predicted as No-shows.
    - TN=True negative

        - Appointments that are NOT predicted as No-shows, and showed up
    - FP= False Positive
        - Appointments predicted as No-shows, but showed up
    - FN= False Negative
        - Appointments that are NOT predicted as No-shows, but didn't show up

The metrics mentioned above can be used to assess the performance of any of the models that I will be considering for this problem.

II. Analysis

Data Exploration and Visualization

The dataset being used for this analysis is in csv format.

Data Set description:

- Shape: 110527 x 14
- Columns:
    - PatientId: Identification of a patient
    - AppointmentId: Identification of each appointment
    - Gender: Male or Female
    - ScheduleDay: The day someone called or registered the appointment
    - AppointmentDay: The day of the actual appointment
    - Age: age in numbers
    - Neighbourhood: Area where the hospital is located
    - Scholarship:
        - This says If the patient has medical scholarship or not.
        - Details: https://en.wikipedia.org/wiki/Bolsa_Fam%C3%ADlia
    - Hipertension:
        - Patient has Hypertension or not
    - Diabetes:
        - Patient has diabetes or not
    - Alcoholism:
        - Alcoholism observed in patient (True or False)
    - Handcap:
        - Patient Handicapped (True or False)
    - SMS_received: 1 or more SMS sent to the patient about the appointment.
    - No-show: if the patient showed up or not. This is the feature being predicted.

Basic description of the dataset

	PatientId	AppointmentID	Age	Scholarship	Hypertension	Diabetes	Alcoholism	Handicap	SMS_received
count	1.105270e+05	1.105270e+05	110527.000000	110527.000000	110527.000000	110527.000000	110527.000000	110527.000000	110527.000000
mean	1.474963e+14	5.675305e+06	37.088874	0.098266	0.197246	0.071865	0.030400	0.022248	0.321026
std	2.560949e+14	7.129575e+04	23.110205	0.297675	0.397921	0.258265	0.171686	0.161543	0.466873
min	3.921784e+04	5.030230e+06	-1.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	4.172614e+12	5.640286e+06	18.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
50%	3.173184e+13	5.680573e+06	37.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
75%	9.439172e+13	5.725524e+06	55.000000	0.000000	0.000000	0.000000	0.000000	0.000000	1.000000
max	9.999816e+14	5.790484e+06	115.000000	1.000000	1.000000	1.000000	1.000000	4.000000	1.000000

Figure 1 Statistical Description of the data

Unique values of each of the columns

```
Age: [-1, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 2
9, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 5
8, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 8
7, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100, 102, 115]
Gender: ['F' 'M']
No-show: ['No' 'Yes']
Diabetes: [0 1]
Alcoholism: [0 1]
Hypertension: [1 0]
Handicap: [0 1 2 3 4]
Scholarship: [0 1]
SMS_received: [0 1]
```

Figure 2 Values in the columns

Adding a new column (‘wait_time’).

Looking at the available columns, one other feature seems to be implicit. This feature will be explicitly added for better understanding of each of the records. This new column(‘wait_time’) represents the time between the time of making the appointment and the actual appointment.

Checking for outliers in `wait_time`

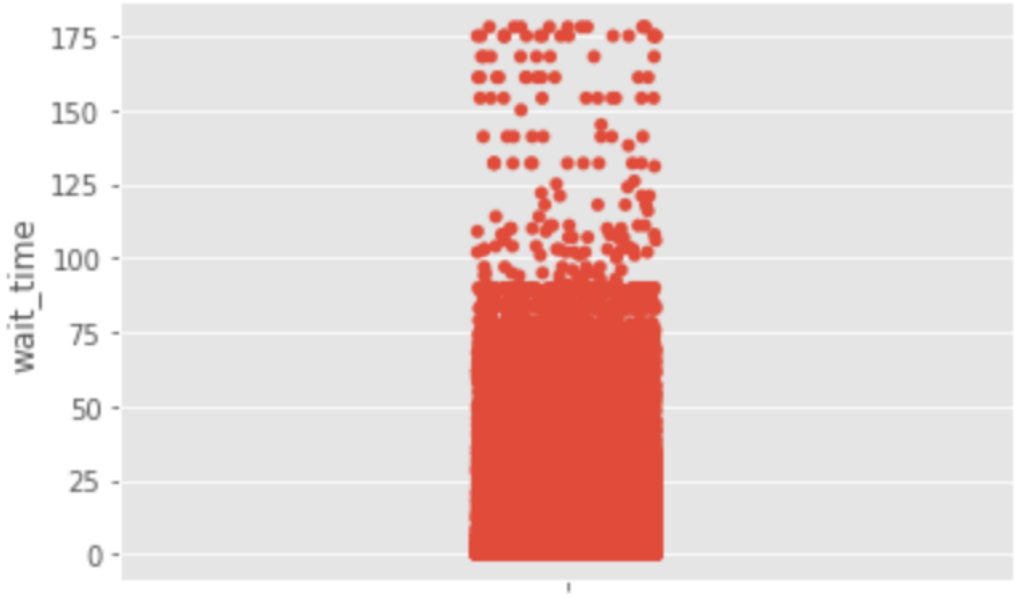

Figure 3 distribution plot for ‘wait_time' outlier detection

From the plot above, we see that there are fewer data points above ‘wait_time’ value of ‘100’. But that is not enough to classify them as outliers.

Outliers in `Age`

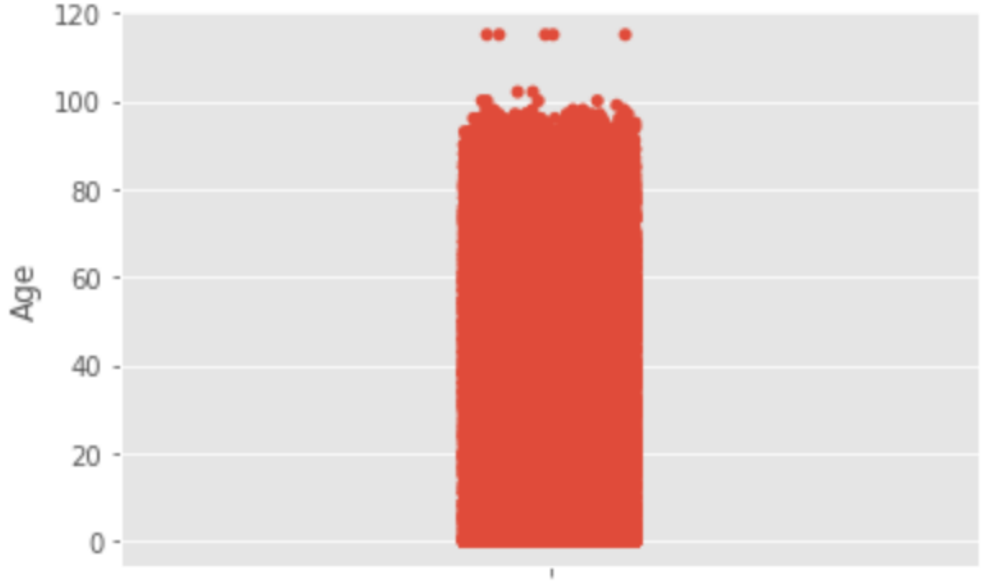

Figure 4 Distribution plot for 'Age' outlier detection

We can see that records with `Age`>100 are very rare. I will be considering them as outliers and remove them.

Appointment distributions based on Gender

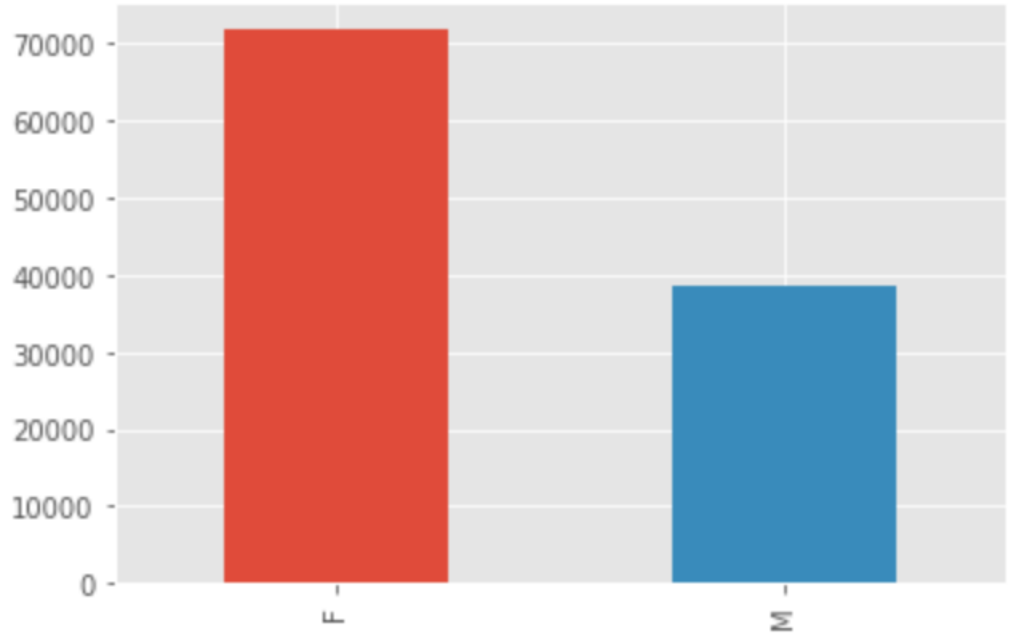

Figure 5 Appointments for different Genders

We see that women make almost twice the number of appointments compared to men.

Who plans ahead? (M or F)

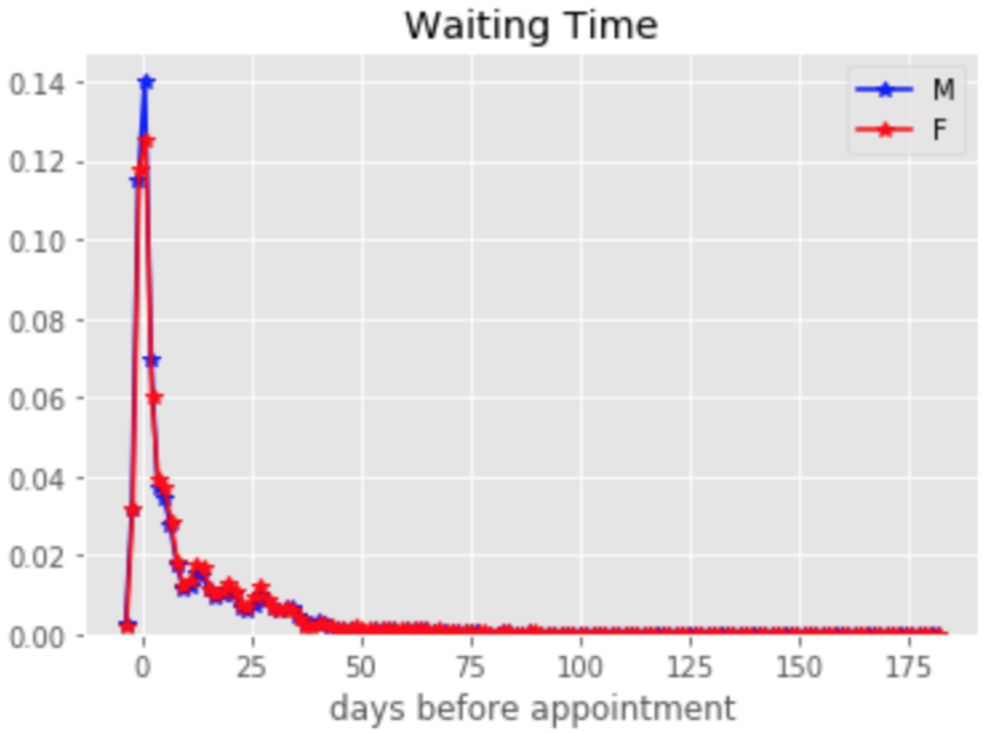

Figure 6 Plan ahead

Tendency to plan ahead seems to be similar across genders.

Class imbalance (label: No-show)

For classification problems, distribution of different target classes is very important. Below plot shows the distribution for the target variable.

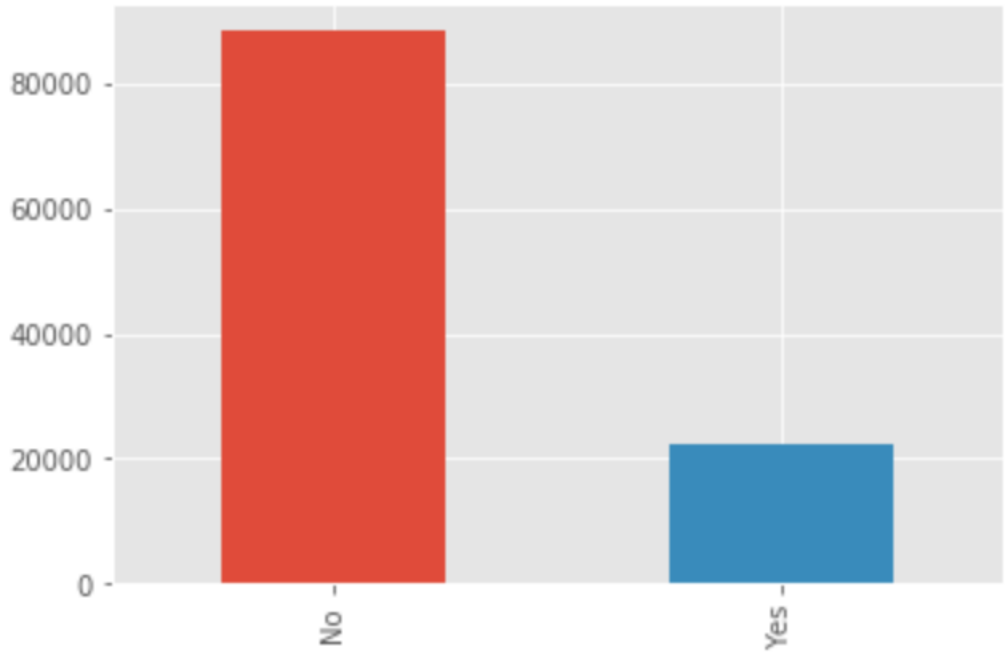

Figure 7 Class Imbalance (No-show)

Note: The class is imbalanced. Only about 30% of the appointments are No-shows. Due to this imbalance, `accuracy_score` won't be a good metric. I will be using other metrics like `recall_score` and `AUC`.

Correlation

It is beneficial to analyze the correlation of independent variables with each other. This helps us identify redundant features and remove them. In this dataset, after seeing the correlation matrix, due to lack of significant correlation, I am not going to remove the features without further investigation.

	Gender	Age	Scholarship	Hypertension	Diabetes	Alcoholism	Handicap	SMS_received	wait_time
Gender	1.000000	0.106452	0.114296	0.055722	0.032556	-0.106166	-0.022813	0.046302	0.027205
Age	0.106452	1.000000	-0.092463	0.504586	0.292391	0.095810	0.078032	0.012633	0.032698
Scholarship	0.114296	-0.092463	1.000000	-0.019730	-0.024894	0.035022	-0.008587	0.001192	-0.030078
Hypertension	0.055722	0.504586	-0.019730	1.000000	0.433085	0.087970	0.080083	-0.006270	-0.018753
Diabetes	0.032556	0.292391	-0.024894	0.433085	1.000000	0.018473	0.057530	-0.014552	-0.028084
Alcoholism	-0.106166	0.095810	0.035022	0.087970	0.018473	1.000000	0.004647	-0.026149	-0.037823
Handicap	-0.022813	0.078032	-0.008587	0.080083	0.057530	0.004647	1.000000	-0.024162	-0.019535
SMS_received	0.046302	0.012633	0.001192	-0.006270	-0.014552	-0.026149	-0.024162	1.000000	0.388191
wait_time	0.027205	0.032698	-0.030078	-0.018753	-0.028084	-0.037823	-0.019535	0.388191	1.000000

Figure 8 Correlation matrix of the independent variables

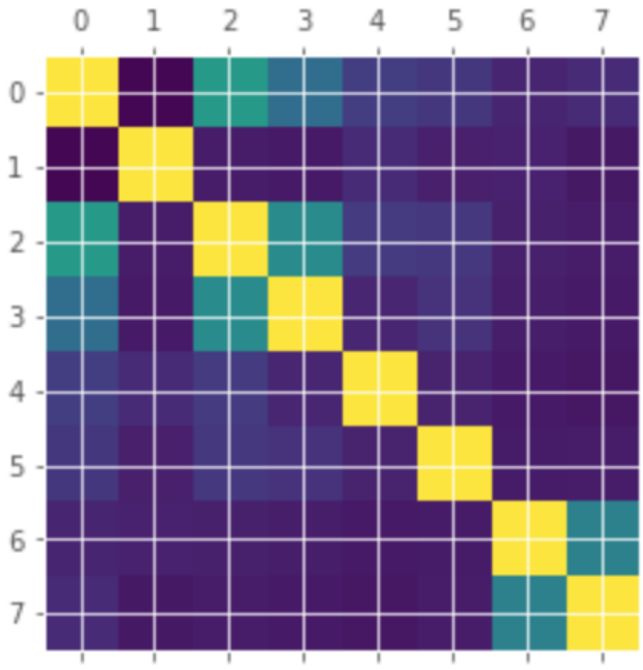

Figure 9 Visual representation of the correlation matrix shown in Figure 7. Lighter the color of the cell, higher the correlation.

Important features

A decision tree classifier is trained on the given data and is used to extract the most important features.

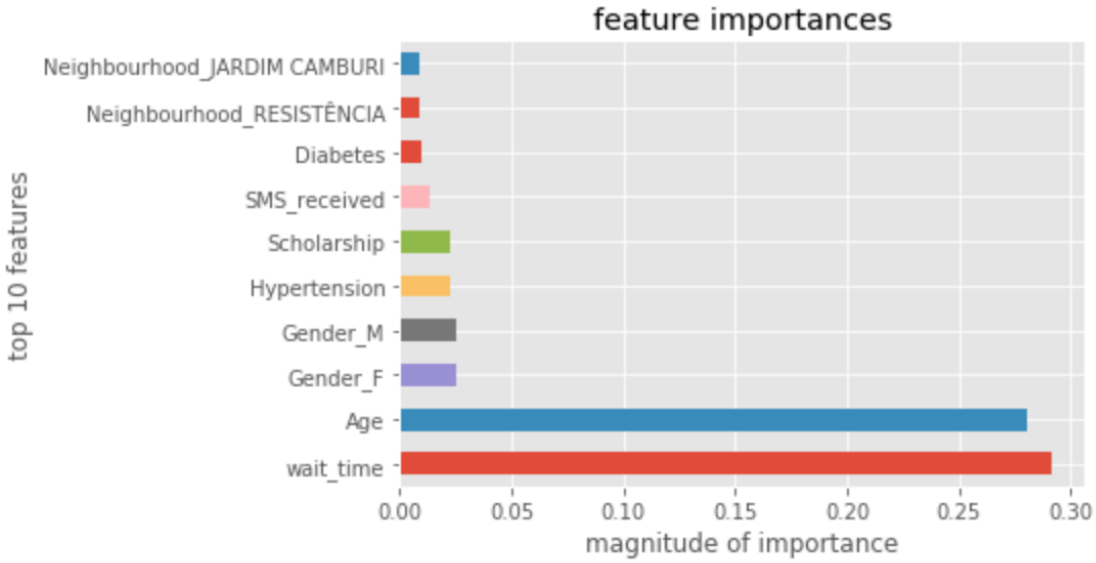

Figure 10 Feature importance provided by the trained Decision tree classifier.

Algorithms and Techniques

In this project, I will be using several classification algorithms with ensemble techniques.

AdaBoost

This is one of the ensemble techniques used for classification. This is a good technique to take multiple weak estimators and build a better stronger estimator. This is flexible. It can be used with many different algorithms. Here I will be using it with decision trees and SVM.
Basics of the algorithm:

- Fit a simple decision tree to the data.
- Evaluate the model against the input
- For the incorrectly classified data points, increase their weight.
- Fit another model on the weighted data and reevaluate. Repeat this step several times.
- In the end the model will have the output of several weak learners which it can ensemble appropriately to give the final classification.

Some of the estimators used:

- Decision trees
    - One of the simplest algorithms. My benchmark is a simple decision tree. I would like to see if the Decision tree can be used with ensemble techniques to get better performance.
    - Also, this is scale invariant.
    - Parameters to be searched:
        - `min_samples_split`: [2,5]

            - This is the minimum number of items per node for the tree to split the node further
        - ‘max_depth’: [None,4,10]
            - This is the maximum depth of the tree. Lower the depth, lower the overfitting.
- Logistic Regression
    - This is another simple algorithm that is suitable for binary classification.
    - Parameters to be searched:
        - `C`: [0.01,0.1,1,10,100]
            - It is the inverse of the regularization strength. Higher `C` implies lower regularization.
- Default
    - `base_estimator`: DecisionTreeClassifier
    - `n_estimattors`:50
- Parameters to be searched.
    - `base_estimator`: Decision Tree Classifier, Logistic Regression
    - `n_estimattors`:25,50,80,160

Grid Search

I will be using grid search for tuning the parameters mentioned above. Grid search is a good technique where I can create a grid of different values for different hypermeters and compare the performance.
Since I am trying to increase ROC area under the curve, I will be using ‘roc_auc’ scoring method for grid search.

Input

I will extract the input, preprocess the data and then feed them to the algorithms discussed above. More detail on the preprocessing is discussed later.

Benchmark

I have trained a Decision tree classifier with default hyperparameters which acts as the benchmark. I have also obtained its performance metrics. It is easy to implement and can be used without much preprocessing of the data. Performance of the Benchmark model is given below:

- F1_score: 0.32
- Accuracy_score: 0.70
- ROC_auc_score (area under ROC): 0.56
    - Needs significantly improvement

III. Methodology

Data Preprocessing

Before feeding the data to AdaBoost classifier, I need to transform some of the input features.

Spelling Correction

Just for ease of understanding, I have corrected the spelling of some features:

- 'Hipertension' to 'Hypertension'
- 'Handcap' to 'Handicap'

Changing the Data Types

- 'AppointmentDay' and 'ScheduleDay'
    - Values in these fields represent 'datetime' quantities. But they are provided as 'string'. So, I have converted them to date time.

Error and NaN in data

In the visualizatinos above we saw that we didn't have any NaN records. But we had invalid data for 'Age' features. i.e., Some records had negative values for age. I have removed those records containing invalid data.

New features

Looking at the given features, we can think of another implicit features which represents the wait time for every appointment, after booking the appointment. This feature is named as 'wait_time'. And this feature is calculated as the number of days between 'scheduleday' and 'AppointmentDay'.

Outlier removal

Based on the visualizations described in the previous section, records with age more than 100 have been removed.

Value mappings

'No-show' features values have been mapped to numeric values:

- 'No'->0
- 'Yes'->1

Unnecessary columns have been removed

- 'AppointmentDay'
    - Since 'wait_time' is derived from this feature, we don't need it anymore.
- 'ScheduledDay'
    - Since 'wait_time' is derived from this feature, we don't need it anymore.
- 'PatientId'

    - This field is just an index of patients. It is irrelevant to the chances of not showing up.
- 'AppointmentID'
    - This field is just an index of appointments. It is irrelevant to the chances of not showing up.

Split into features and labels

To feed the data into the classifier, I have extracted the independent variables as 'features' and the dependent variable as 'labels'.

Categorical data

Categorical features have been converted into new feature. For every categorical value, new feature is added which has binary value (1/0).

Scaling

Different input features have different value range. It is a good idea to use standard scaling methods to normalize all the input features so that all the features will have similar value ranges. This makes sure that no feature gets special importance just due to its magnitude.

Implementation

All t of the preprocessing, algorithms and feature transformations have been done in a Jupyter Notebook (Python). This Notebook is provided with this report.

Algorithms have been implemented using the '**scikit-learn'** library. Data loading and preprocessing is done using '**pandas'** library.

The implementation steps are described below:

Data loading

'Pandas' is used to load the csv file as a DataFrame (in memory representation as table of records).

Visualizations

- 'matplotlib' and 'seaborn' libraries have been used to visualize the data.
- Bar plots have been used to visualize the distribution of the class(labels) as well as values of other features.
- DataFrame's methods have been used to get the statistical description of the dataset.
- Seaborn's stripplot has been used to visually detect outliers. This plot shows the density of different values for a given feature.
- 'kdeplot' is used to visualize the trend in different characteristics.
- Correlation matrix is calculated and visualized which shows the correlation of different independent variables with respect to each other. This information is intended to detect redundant features. In case of this dataset, significant correlation has not been found.

Feature Engineering

- A decision Tree classifier is used to extract the 10 most important features.
    - These are visualized using bar plot.
- Categorical features have been converted into new features where for every categorical value, a new feature is added which indicates of a particular entry(record) has that particular categorical value or not.
- Some unnecessary features have been added. And some useful features have been derived from the given features.
    - 'AppointmentDay'
        - Since 'wait_time' is derived from this feature, we don't need it anymore.
    - 'ScheduledDay'
        - Since 'wait_time' is derived from this feature, we don't need it anymore.
    - 'PatientId'
        - This field is just an index of patients. It is irrelevant to the chances of not showing up.
    - 'AppointmentID'
        - This field is just an index of appointments. It is irrelevant to the chances of not showing up.
    - 'wait_time'
        - And this feature is calculated as the number of days between 'scheduleday' and 'AppointmentDay'.
- **PCA (Principal component analysis)**
    - PCA has been performed where we extract principal components that explain most of the variance in the data.
    - But after performing PCA, I realized that to get significant variance explained in my principal components, I needed to select very large number of components which was almost equal to the number of features. So, I have decided that PCA is not going to be much useful in this case.

Preprocessing

- The dataset is cleaned by removed invalid records.
- Outliers detected in the previous section have been removed.
- Categorical features have been handled as described above.
- scikit-learn's StandardScaler is used to scale the dataset.
    - Standardization of a dataset is a common requirement for many machine learning estimators: they might behave badly if the individual feature do not more or less look like standard normally distributed data
    - Standardized features by removing the mean and scaling to unit variance.

Model Selection and Training

Now that I have the data preprocessed and ready, I will feeding it to different Adaboost classifiers. In this case I am using Adaboost with two different base estimators.

Decision tree classifier

- In this case, decision trees with default configuration are used as base estimators for the Adaboost classifier.
- The implementation in scikit-learn is used for this purpose.
- Metric scores:

```
f1_score: 0.317193675889
accuracy: 0.749909518639
roc_auc : 0.576925731571
recall  : 0.284658587053
```

Logistic Regression

- In this case, Logistic regression classifier is used as the base estimator for the Adaboost classifier.
- Metric scores(with held out test data):

```
f1_score: 0.0310478654592
accuracy: 0.796706776441
roc_auc : 0.503912490476
recall  : 0.0163043478261
```

Refinement

The initial results have been described in the previous section. The previous models have been refined by tuning the hyperparameters. For this, I have used GridSearch method which takes a set of values for hyperparameters and creates and validates models with all those combinations. I have used 2 separate grid searches for 2 separate estimators.

Grid search 1

- In this step, an AdaBoost classifier is trained using Decision Tree classifiers.
- Parameter grid

```
param_grid = {
              'base_estimator__min_samples_split':[2,5],
              'base_estimator__max_depth':[None,4,10],
              "n_estimators": [80],
              'base_estimator__class_weight':['balanced']
}
```

- In this grid search I am using Decision Tree classifier as the base estimator. The various parameters have been explained in the previous section.
- The grid search in this case is set to look for the model with the best 'auc_roc' score (area under the curve).
- After running the grid search, I got the following parameters as the best parameters.

```
{'base_estimator__class_weight': 'balanced',
 'base_estimator__max_depth': 4,
 'base_estimator__min_samples_split': 5,
 'n_estimators': 80}
```

- Metrics:

- The following metrics scores are obtained for the selected best model (on held out test data).

```
f1_score: 0.434566994214
accuracy: 0.628604174207
roc_auc : 0.661977846194
recall  : 0.717404487568
```

  - We can see that there is a significant increase in the 'auc_roc' score.

Grid Search 2

- In this step, an AdaBoost classifier is trained using Logistic Regression Classifier.
- Parameter grid

```
param_grid = {'base_estimator__C':[0.01,0.1,1,10,100],
                                "n_estimators": [80,160],
               'base_estimator__class_weight':['balanced'],
              }
```

- In this grid search I am using Logistic regression classifier as the base estimator. The various parameters have been explained in the previous section.
- The grid search in this case is set to look for the model with the best 'auc_roc' score (area under the curve).
- After running the grid search, I got the following parameters as the best model parameters.

```
{'base_estimator__C': 0.1,
 'base_estimator__class_weight': 'balanced',
 'n_estimators': 80}
```

- Metrics:
  - The following metrics scores are obtained for the selected best model (on held out test data).

```
f1_score: 0.399475700746
accuracy: 0.640728676559
roc_auc : 0.620222101077
recall  : 0.585574933491
```

    - We can see that there is a slight increase in the 'roc_auc' score.

IV. Results

Model Evaluation and Validation

Based on the model validation and evaluation techniques performed earlier, I have selected the following model as the best model:

- Classifier: Adaboost
  - Base estimator : DecisionTreeClassifier
    - Criterion : gini
    - Max depth : 4
    - Minimum split : 5
  - Number of estimators: 80

```
AdaBoostClassifier(algorithm='SAMME.R',
          base_estimator=DecisionTreeClassifier(class_weight='balanced', criterion='gini', max_depth=4,
            max_features=None, max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=5,
            min_weight_fraction_leaf=0.0, presort=False, random_state=None,
            splitter='best'),
          learning_rate=1.0, n_estimators=80, random_state=None)
```

- The reason for selecting this model is its performance in 'roc_auc' score. Even though the **performance** is significantly **less than I had expected**, this is still better than the benchmark. Due to the nature of this problem, as explained earlier, I want a model that has high 'roc_auc' score.
- Metric score on training data:

```
f1_score: 0.477557027226
accuracy: 0.651254475654
roc_auc : 0.70256279527
recall  : 0.788679245283
```

- Metrics for validation data

```
f1_score: 0.439610037913
accuracy: 0.625527807938
roc_auc : 0.660569750207
recall  : 0.719775347325
```

- The metric scores for the best model selected are (on test data):

```
f1_score: 0.434566994214
accuracy: 0.628604174207
roc_auc : 0.661977846194
recall  : 0.717404487568
```

  - As we can see this model has a good 'recall_score' along with the highest 'roc_auc' score.
- Robustness:
  - The model has less variance.

- Since the models scores are evaluated on the held-out test data set, match its performance on train and validation sets, we can safely assume that the result obtained is robust.

Justification

Benchmark model performance

The benchmark model is a Decision tree classifier with the default implementation in 'scikit-learn' library.

Benchmark model:

```
DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=None,
            max_features=None, max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, presort=False, random_state=None,
            splitter='best')
```

Metric scores for the benchmark model:

```
f1_score: 0.321219512195
accuracy: 0.748190372783
roc_auc : 0.579296052676
recall  : 0.297627118644
```

As we can see, the benchmark model has very low 'recall' and 'roc_auc' scores. Both are important in this project.

```
AdaBoostClassifier(algorithm='SAMME.R',
          base_estimator=DecisionTreeClassifier(class_weight='balanced', criterion='gini', max_depth=4,
            max_features=None, max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=5,
            min_weight_fraction_leaf=0.0, presort=False, random_state=None,
            splitter='best'),
          learning_rate=1.0, n_estimators=80, random_state=None)
```

Final selected model

The model selected as the final (best) model is an Adaboost classifier using Decision trees as base models.

The parameters are as follows:

Metric scores for the best model:

- Validation data

```
f1_score: 0.442053878141
accuracy: 0.627699360598
roc_auc : 0.663032950007
recall  : 0.722731303577
```

- Test data

```
f1_score: 0.436007684567
accuracy: 0.628121606949
roc_auc : 0.663613900721
recall  : 0.722559126743
```

We can see that the final model has significantly higher scores ('recall' and 'roc_auc') on the test and validation sets. Even though this model has some variance, it is still generalizing well on the test data.
The problem required us to increase the recall significantly (29% to 72%), since finding out 'No-shows' was important. But this problem also can't tolerate very high false positives. So, 'roc_auc' should also be considered.
'roc_auc' has improved from 57% to 66%.
The high 'recall' and 'roc_auc' scores are an indication that we have found a good solution to the problem.

V. Conclusion

Free-Form Visualization

In this section, I want to emphasize on one of the very important visualizations. Usually when we have very high number of features, it is beneficial to be able to see which of the features dictate the classification. In other words, it is useful to know the most influential features. Following visualization plots the 10 most important features.

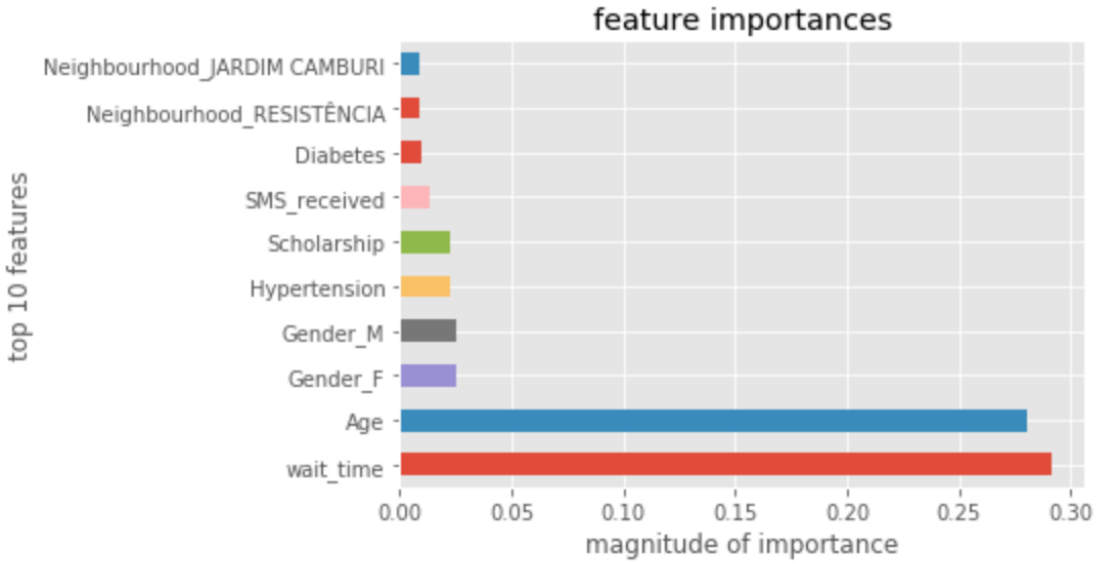

Figure 11Feature importances

The figure above shows the most important features. This **DOES NOT** mean that these features are enough for classification. In fact, use only these features will decrease the performance significantly.

It is interesting to note that `wait_time`, the feature that was added by us is the most influential feature.

Reflection

Summary

This project consisted of the following processes:

- Extracted and cleaned the data
- Preprocessed the data to be suitable for the models chosen.
- Explored the data with relevant visualizations.
- Removed unnecessary features
- Added new features.

- Removed invalid records from the dataset.
- Scaled the dataset to make sure that the data ranges in each of the columns have similar magnitude.
- Split the dataset into train, validation and test datasets.
- Built the benchmark model and tested its performance.
- Ran PCA analysis on the dataset to get the principal components. Based on the variance explained by the principal components, I decided that the PCA analysis was not beneficial in this case.
- Model selection and validation:
    - Used Grid search to find a model with the highest 'roc_auc' score.
    - Once the best model is found, it is evaluated on the test dataset.

The most interesting part was feature engineering.

- The PCA analysis did not give a better set of features.
- The newly added feature turned out to be the most influential feature.

The final model's performance is close to the expected performance.

Improvement

The solution provided can be slightly customized by modifying the weight(importance) given to different classes ('No-show','Show') during the classifier training step. In this case, I could have done that with changing the 'class_weight' parameter of the decision tree classifier that I used. In this case I used 'class_weight='balanced''. This means that both the classes will be balanced during training. Increasing the weight for 'No-show' would have increased the recall. But that would have causes more false positives. Depending upon the custom problem being solved, this method can be used to get a solution more suited to the problem.

Another improvement would be to use deep neural networks. I trried creating simple neural network and increased the depth. With increase in neurons, the performance kept increasing. But due to lack of better understanding of the effect of parameters, I decided to abandon that method. With more understanding of the neural networks, it could be possible to use them for better performance.

References

- https://www.kaggle.com/joniarroba/noshowappointments/home
- https://github.com/udacity/machine-learning/blob/master/projects/capstone/capstone_report_template.md
- http://scikit-learn.org/