ORIE 4741 Fall 2019
Predicting Presence of Cardiovascular Disease in Patients

Adam Guo(hg426), Karan Maheswari(km857), Krishna Raju(kk992)

December 10th, 2019

1 Intro

In this project, we are concerned with what factors attribute to a person having heart disease. Heart disease is a chronic disease that takes a great amount of resources to treat, and if handled improperly could take away someone's life easily. By figuring out which factors have the most probabilities of leading to heart diseases, we could monitor those factors for patients, and thus prevent heart disease. For this project, we would seek to build a predictive model using data from the 'Cardiovascular Disease dataset' from Kaggle, which contains heart disease information of around 70,000 patients.

1.1 Goal

Our goal for the project is two-fold. First, we want to come up with a machine learning model to determine which factors (features) in the dataset contribute the most to the patient having cardiovascular disease. Second, we want to be able to predict the risk of a new patient having cardiovascular disease given his/her data, and maybe come up with a confidence interval for the prediction. To evaluate our model performance, we want to look at the test error for the model, and also compare the important factors we got from the model to some clinical proven results.

1.2 Data

We are using the '*Cardiovascular Disease Dataset*' from Kaggle. Our data set has 12 features and 70,000 entries. The features in the dataset are of 3 types -

Objective - Factual Information
Examination - Results of medical examination
Subjective - Information given by patient

It is important to distinguish between the three types, so that we can decide how much 'reliance' we can put on each feature. For example, subjective data like alcohol intake might not always be accurate because people might falsely report (knowingly or unknowingly) their medical conditions.

Below we list all our features with their type and some other useful properties:

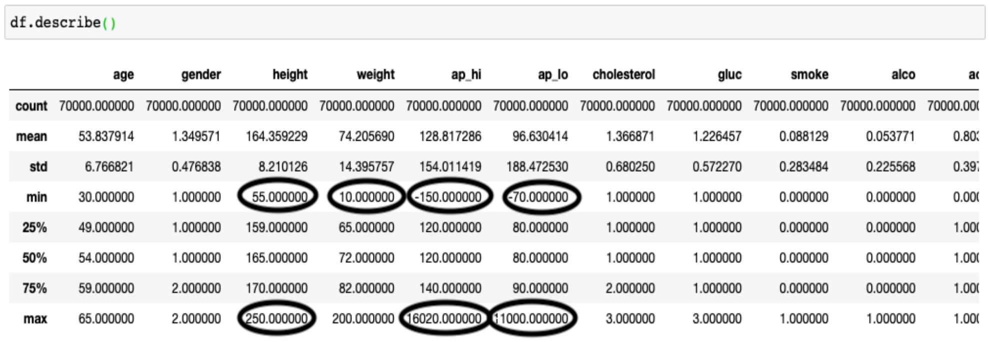
1. Age | Objective Feature | age | int (days)

2. Height | Objective Feature | height | int (cm) |
3. Weight | Objective Feature | weight | float (kg) |
4. Gender | Objective Feature | gender | categorical code |
5. Systolic blood pressure | Examination Feature | ap_hi | int |
6. Diastolic blood pressure | Examination Feature | ap_lo | int |
7. Cholesterol | Examination Feature | cholesterol | 1: normal, 2: above normal, 3: well above normal |
8. Glucose | Examination Feature | gluc | 1: normal, 2: above normal, 3: well above normal |
9. Smoking | Subjective Feature | smoke | binary |
10. Alcohol intake | Subjective Feature | alco | binary |
11. Physical activity | Subjective Feature | active | binary |
12. Presence or absence of cardiovascular disease | Target Variable | cardio | binary |

2. Procedure

2.1 Data Exploration

There are 70,000 examples and 11 features which describe the target variable.
There isn't any missing data but there are some outliers. We observed this in exploratory data analysis.



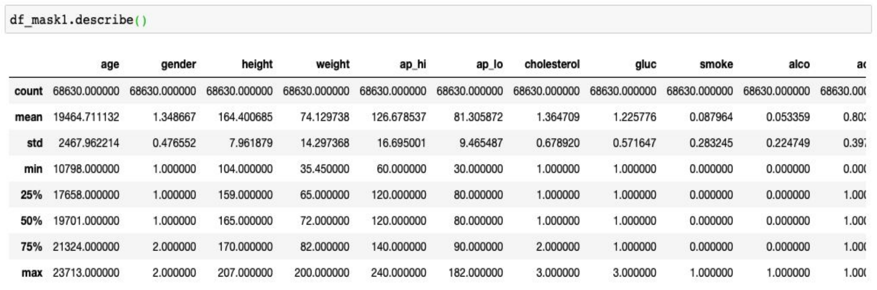
```
df.describe()
```

	age	gender	height	weight	ap_hi	ap_lo	cholesterol	gluc	smoke	alco	ac
count	70000.000000	70000.000000	70000.000000	70000.000000	70000.000000	70000.000000	70000.000000	70000.000000	70000.000000	70000.000000	70000.00
mean	53.837914	1.349571	164.359229	74.205690	128.817286	96.630414	1.366871	1.226457	0.088129	0.053771	0.80
std	6.766821	0.476838	8.210126	14.395757	154.011419	188.472530	0.680250	0.572270	0.283484	0.225568	0.39
min	30.000000	1.000000	55.000000	10.000000	-150.000000	-70.000000	1.000000	1.000000	0.000000	0.000000	0.00
25%	49.000000	1.000000	159.000000	65.000000	120.000000	80.000000	1.000000	1.000000	0.000000	0.000000	1.00
50%	54.000000	1.000000	165.000000	72.000000	120.000000	80.000000	1.000000	1.000000	0.000000	0.000000	1.00
75%	59.000000	2.000000	170.000000	82.000000	140.000000	90.000000	2.000000	1.000000	0.000000	0.000000	1.00
max	65.000000	2.000000	250.000000	200.000000	16020.000000	11000.000000	3.000000	3.000000	1.000000	1.000000	1.00

- Here we can see that weight feature (kgs) has a minimum value of 10. For a 30-65 year old person having a weight of 10 kgs is not possible.
- Similarly we see in height a person with a height of 55 cm. This indicates presence of corrupt values
- Another set of features which have corrupt values are ap_hi (Systolic Blood Pressure) and ap_lo (Diastolic Blood Pressure). After carefully studying the data we saw that there are garbage values of negative and very high blood pressure.

There are two possible solutions to this problem: either we can drop the samples with corrupt value or we can replace the value with median/mean values.

Since our dataset is large enough and our application is health critical we chose to drop the examples with corrupt entries.

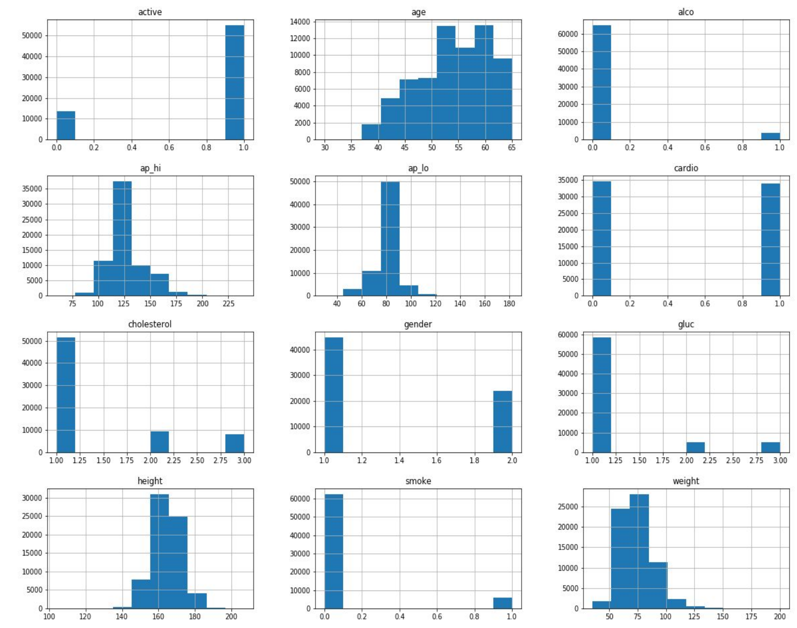


```
df_mask1.describe()
```

	age	gender	height	weight	ap_hi	ap_lo	cholesterol	gluc	smoke	alco	ac
count	68630.000000	68630.000000	68630.000000	68630.000000	68630.000000	68630.000000	68630.000000	68630.000000	68630.000000	68630.000000	68630.00
mean	19464.711132	1.348667	164.400685	74.129738	126.678537	81.305872	1.364709	1.225776	0.087964	0.053359	0.80
std	2467.962214	0.476552	7.961879	14.297368	16.695001	9.465487	0.678920	0.571647	0.283245	0.224749	0.39
min	10798.000000	1.000000	104.000000	35.450000	60.000000	30.000000	1.000000	1.000000	0.000000	0.000000	0.00
25%	17658.000000	1.000000	159.000000	65.000000	120.000000	80.000000	1.000000	1.000000	0.000000	0.000000	1.00
50%	19701.000000	1.000000	165.000000	72.000000	120.000000	80.000000	1.000000	1.000000	0.000000	0.000000	1.00
75%	21324.000000	2.000000	170.000000	82.000000	140.000000	90.000000	2.000000	1.000000	0.000000	0.000000	1.00
max	23713.000000	2.000000	207.000000	200.000000	240.000000	182.000000	3.000000	3.000000	1.000000	1.000000	1.00

After looking at the filtered data we realized age is in number of days. Age in number of days can not model the relationship between heart disease and age properly. So we transformed it into years.

Then we drew the histograms of all the features, which are useful for us to understand each feature and its distribution.



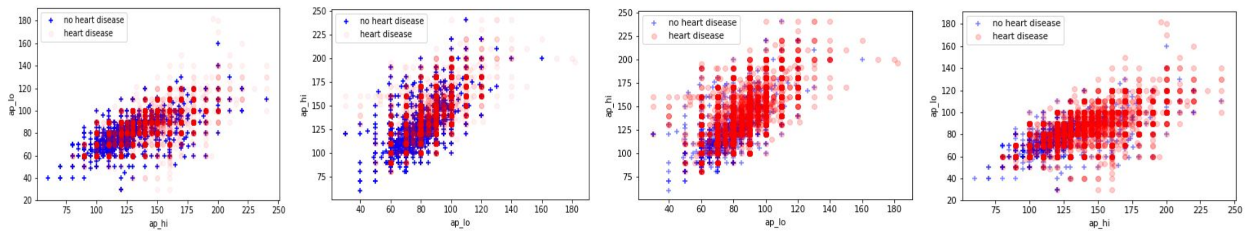
A few observations about the dataset from the histogram

Looking at the target we see that dataset is balanced (equal number of examples for both ‘heart disease’ and ‘no heart disease’ and not skewed.

Next we go on to realise that the number of men and women isn’t uniform and thus before making any conclusion about correlation of heart disease with gender we should take this fact in account.

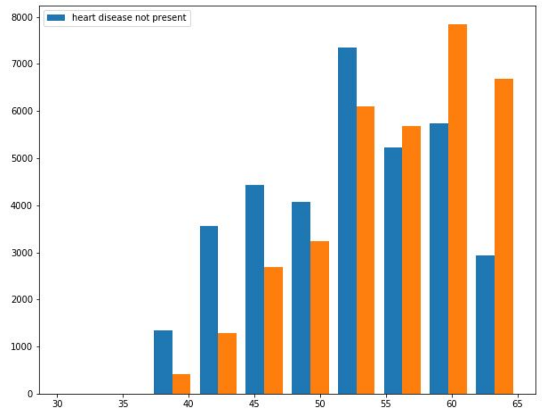
After this as commonly known that BP is correlated to heart disease, we wanted to model the relationship.

After drawing some scatter plots



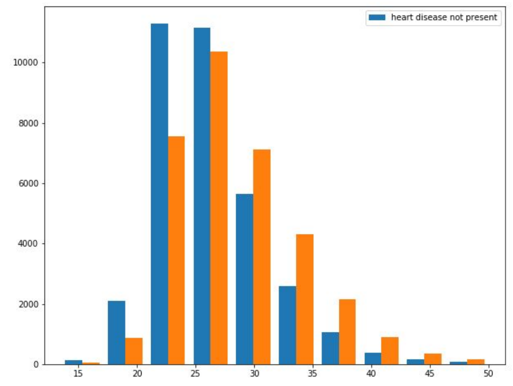
We can see a trend here that high blood pressure is somewhat correlated to the presence of heart disease.

Also we drew a histogram for age with presence of heart disease:



We can clearly see a trend here, “as age is increasing people with heart disease increases”
Also it can be seen that at lower age relatively fewer people have heart disease, and at higher age relatively few people do not have heart disease.

We couldn’t find any relation between height and presence of disease or weight and presence of disease so we did a polynomial transformation of height and weight which is Body Mass Index (height/weight^2). Here again we can very well see a trend in the figure below. At lower BMI fewer people have heart diseases and as obesity increases, the number of heart diseases also increases.



2.2 Feature Engineering

To better prepare our features to be used for our model, we did simple feature engineering for the project.

3 Model Development

3.1 Model Selection

We decided to split the data 85-15, meaning 85% of the entire dataset will go to training dataset, while 15% of the entire dataset will go to testing. We decided to not include an evaluation data for this project.

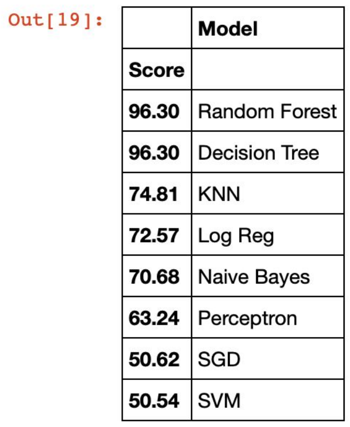
For this model selection portion, we are using Python's Sklearn library, which contains a lot of useful modules for a majority of the machine learning models.

Here are a list of models that we attempted:

- Linear model that uses Stochastic Gradient Descent
- Random Forest Classifier
- Logistic Regression
- K-Nearest Neighbor Classifier
- Gaussian Naive Bayes Classifier
- Perceptron Classifier
- Support Vector Machine
- Decision Tree Model

Training the models:

After training the models, we need to evaluate their training performance and decide on which metrics to be used to judge that. We decided to use the classification accuracy (prediction == actual) to compare each model that we trained, because our output label is a 2-class classification label, and the accuracy should be sufficient to do a simple comparison. The mean accuracy of each model is shown below:



Out[19]:

	Model
Score	
96.30	Random Forest
96.30	Decision Tree
74.81	KNN
72.57	Log Reg
70.68	Naive Bayes
63.24	Perceptron
50.62	SGD
50.54	SVM

We notice here that Random Forest and a Decision Tree Model actually achieve the best training performance. Both methods have similar structure - nodes for representing class label and branch for representing prediction outcome. A decision tree model as opposed to a random forest model will be nice because the decision tree is a white-box model, meaning that it is easily explainable, and one can clearly see where the decision is being split.

A random forest model, on the other hand, is just the ensemble method of the decision tree model. It builds multiple decision trees given random subsets of the entire dataset, and eventually average the results. Averaging here means each tree at the end will vote for the classification label, and the label that received most votes by simple majority will be the prediction. Compared to a decision tree model, random forest model doesn't overfit as often, because by subsetting the dataset and averaging the results, one effectively lower the variance in the final prediction. And therefore, to avoid potentially overfitting the dataset and to better be able to generalize the model to real world data, **we decide to use a random forest model**. We accept the inherent trade-off of using a black-box model, i.e, our model might not be interpretable but we could expect better results than a simpler white box model like decision trees. Our choice is also incentivised by the fact that, the basic corelation between our features and the risk of heart disease is already known. Therefore, it would not be of much value for us to build a white-box model which would simply reaffirm already known predictors of heart disease. Instead, we attempt to build a model which can outperform many of these simple models and do a better job at predicting heart disease.

3.2 Cross Validation & Overfitting

Given the high training accuracy that we are seeing from above, we were a little skeptical of the model's performance on real data. To give us a sense of how the model will perform on real data, we decided to use a 10-fold cross validation on the training dataset using the random forest model, to check for potential overfitting. Here is the result we get from the 10-fold cross validation:

```
Scores of model 2 with X_train 2:  [0.69704264 0.69222834 0.69893398 0.70323246 0.70392022 0.6966466
 0.69406707 0.70106639 0.71448228 0.71517028]
Mean of scores2:  70.16790259900283
Std Deviation of scores2:  0.007452267115960629
```

As we can see from the output, the mean accuracy of the 10 models is only around 70.17%, which is much lower than the training accuracy we saw earlier (>90%). Therefore, there is clearly some overfitting that we are doing earlier with the training of the random forest. Here we will describe how we battled overfitting.

Normally, the methods we discussed in class to reduce overfitting were to either

1. Get more data to fit the model
2. Reduce the amount of features
3. Use a less complex model

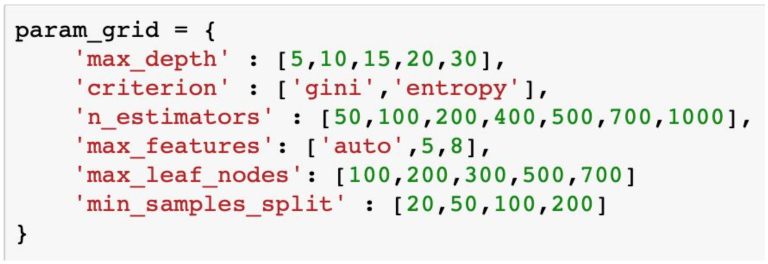
We couldn't possibly get more data to fit our model, so we chose two methods to reduce the overfitting. The first was to use less features in our model - we decided to drop additionally 'Alco', 'Smoke', and 'Active' from our model. The second was actually a little unconventional - because random forest is an ensemble method that meant to reduce variance, we decided to do some hyper-parameter tuning (eg. to train more trees in the forest) to reduce overfitting.

Hyperparameters, different from model parameters, are configurations external to the model and cannot be estimated or trained from the data. They are usually manually set by the person running the model, and given the predictive modeling problem, they can be fine-tuned in order to better the performance of the model. For our random forest model, we decided to perform a grid search, in order to find the optimal hyperparameter that results in the best prediction.

Here are some of the hyperparameters we can specify for our random forest model:

- max_depth: the maximum depths of the trees in the forest
- criterion: the scoring function used to construct the trees (gini or entropy)
- n_estimators: number of trees to train in the forest
- max_features: the maximum amount of features in each tree
- max_leaf_nodes: the maximum amount of leaf nodes in each tree
- min_smaples_split: the minimum amount of samples required to split an node

Here are some possible values (grids) of the hyperparameters that we hope to tune:



```
param_grid = {
    'max_depth' : [5,10,15,20,30],
    'criterion' : ['gini','entropy'],
    'n_estimators' : [50,100,200,400,500,700,1000],
    'max_features': ['auto',5,8],
    'max_leaf_nodes': [100,200,300,500,700]
    'min_samples_split' : [20,50,100,200]
}
```

What the grid searching is doing is that it iteratively trains the model based on unique combinations of those configurations, then it notes down the accuracy and variance of each model. At the end it returns the set of configuration that gave us the best results before. During grid search, in fact a cross-validation process is used to make sure the performance difference is resulted from the configurations, but not the trained weights(threshold) of the model.

Here is the result returned from grid search:

```
GridSearchCV(cv='warn', error_score='raise-deprecating',
       estimator=RandomForestClassifier(bootstrap=True, class_weight=None, criterion='entropy',
            max_depth=20, max_features='auto', max_leaf_nodes=200,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, n_estimators=700, n_jobs=-1,
            oob_score=False, random_state=1, verbose=0, warm_start=False),
       fit_params=None, iid='warn', n_jobs=-1,
       param_grid={'min_samples_split': [20, 50, 100, 200]},
       pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
       scoring=None, verbose=0)
```

We can notice that the best n_estimator, the number of trees in the forest was returned to be 700, much larger than what we used to train the model before(100). Also, notice that the max_depth, maximum depth of the tree is set to 20, further restricting the complexity of the decision trees in the forest. We will use this set of hyperparameters to retrain our model and get some results.

4 Testing

4.1 Retrain & Testing Model

Given the hyperparameters set returned from the grid search, we retrained our random forest model, which resulted in a training accuracy of 74.47%. This lowered accuracy is expected because we don't want our model to fit every bit of details of the training data too well, but instead want it to be able to have good performance for data we haven't seen before.
We later ran the model on the test dataset. Here are the accuracy numbers that we achieved:

```
Train Accuracy:  74.47
Test Accuracy:  73.56523433693852
```

The test accuracy is 73.565%, higher than from what we saw before (68%) and close to that of training. This is good because we have relatively high training and testing accuracy, and because they are close together, it could mean that our model would perform well on real world data. However, as discussed before, this also resulted in lowered training accuracy, and potentially more bias in the model, and we understand that the **bias-variance-tradeoff** is in effect here.

4.2 Model Performance Comparison

Given our increased bias, we wanted to see how other previous efforts have fared in tackling the same classification tasks, especially using other machine learning models.

We found one instance of a team solving the same task of cardiovascular disease prediction using a Neural Network solution. The best test accuracy they achieved after tuning and optimization was 72%. Here is a link to their project:
https://www.kaggle.com/camiloqq/nn-to-predict-cardiovascular-diseases

We also noticed from online that the highest accuracy to have achieved on this dataset was around 72% or 73%, and our solution right there was the gold standard. Therefore, we decided to stick with our models, features, and the hyperparameters that we chose.

4.3 Further Results Investigation

Because our prediction task involves a binary classification in the medical field, we also want to look at statistical measures specific for this field, such as sensitivity and specificity. Here are the definition of both terms:

- Sensitivity: also the recall or true positive rate, is the proportion of true positives that are correctly identified. In our case, it will be the percent of heart disease patients who are actually predicted as such.
- Specificity: also the true negative rate, is the proportion of true negatives that are correctly identified. In our case, it will be the percent of healthy people who are correctly predicted as such.

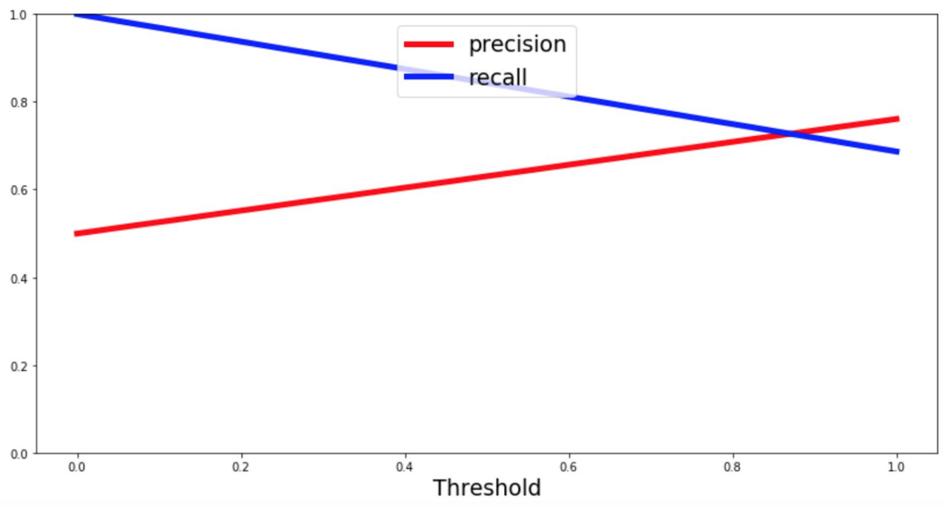
Both of those values are important, because in most settings, there are threshold requirements for sensitivity and specificity for medical classification tests to achieve. In fact, classification tests with both values above 90% are considered to have high credibility, although there are often tradeoffs between the two values.

To look at the sensitivity and specificity for our model, we calculated the true positives and true negatives, and for references we also included the false negatives and false positives. We then calculated the specificity and sensitivity and printed the results:

```
Classification Report :               precision    recall  f1-score   support

           0       0.71      0.78      0.75      5136
           1       0.76      0.69      0.72      5127

   micro avg       0.74      0.74      0.74     10263
   macro avg       0.74      0.74      0.73     10263
weighted avg       0.74      0.74      0.74     10263

True Positive : 3520
True Negative : 4030
False Positive : 1106
False Negative : 1607
Specificity : 0.7846573208722741
Sensitivity : 0.6865613419153501
```

As we can see from the output above, the sensitivity of the model is 68.66%, and the specificity of the model is 78.47%. The sensitivity value is actually a little disappointing - the low value indicates that we are not doing a good job avoiding false negatives in our prediction. And in this context, it means that our model is not doing a good job predicting that heart disease patients actually have heart disease, and so that the disease is overlooked. This could potentially result in the patient not getting immediate attention to his/her health and getting the right treatment.

One way to combat this low sensitivity value is set a different threshold for classifying a patient as having heart disease. If we make the threshold lower, then we are more likely to predict someone having heart disease, regardless of the person actually having it or not. Here we showed a graph describing how the sensitivity and specificity change with the moving threshold values (recall is sensitivity):



We see that there is clearly a tradeoff for both values, and setting the threshold lower could potentially benefit our model by increasing the sensitivity but decreasing the specificity.

5. Weapons of Math Destruction

Ou