Steps taken to prepare data:

1. Subtract 1 from both the affection status and gender columns to ensure the data was following one hot encoding.
2. Split data into two dataframes, one holding only those rows where the affection status was 0, the other holding the rows where the affection status was 1
3. **Sampled with replacement from the dataframe where the affection status was 0 (not affected), as in the original data there exists a ⅔, ⅓ split between patients who are affected and those who are not**
4. Combined the two dataframes back together
5. **Filtered out all rows where the protein values from BL, V4, V6, V8 were below 5, as those entries are likely the result of fat finger errors**
6. Normalized age, BL, V4, V6, V8 values across the whole dataset, by calculating the mean and std for each column and applying the rule (x_i - mu / sigma)
7. Filtered out the PD Duration, V10 and V12 columns, as:
    a. The regression analysis suggests there to be very little signal in the PD Duration field, as well as the fact that no patient whose affection status is 0 has a PD Duration
    b. The V10 field is very unbalanced even after the resampling (resampling just creates duplicates of existing rows, if those rows don't have filled out values for V10, resampling can't fully eliminate the imbalance problem)
    c. The V12 field just doesn't have enough rows filled out for both affected and not affected patients
8. **The result is that we are using the following features after normalization and resampling:**
    a. **Gender, Age, BL, V4, V6, V8**

Some notes on the stats reported

Per metric I report 4 different versions of how one could calculate the metric:

1. all - Per label metric value.

2. micro - Performance across the whole dataset.
   Will be the same across all metrics when all labels are used.

3. macro - The average of the per label value of the metric… average of the “all” values <-usually what packages return

4. weighted - The weighted, by frequency of class, average of the per label value of the metric.

Some Notes:

1. “Support” represents the number and distribution of labels passed into the model for testing

2. I ran 10-fold cross validation and took the mean and stdev of the returned scores

3. I haven’t tweaked parameters for the models, just reporting what params were being used

4. Graphs are included at the end of the presentation

Random Forest Performance

Params: 30 trees used, no max depth, using gini coefficient to decide how to split trees

Cross Fold Mean: **0.818**, Cross Fold Stdev: **0.055**

For a particular split of data, we have the following metrics:

accuracy: 0.748

precision:
- all: ['0.681', '0.833'] <- notAffected, affected
- micro: 0.748
- **macro: 0.757**
- weighted: 0.764

recall:
- all: ['0.839', '0.672']
- micro: 0.748
- **macro: 0.755**
- weighted: 0.748

f1_measure:
- all: ['0.752', '0.744']
- micro: 0.748
- **macro: 0.748**
- weighted: 0.748

support:
- **y_true_dist: ['0.455', '0.545']**
- y_true_count: 123.000
- **y_predicted_dist: ['0.561', '0.439']**
- y_predicted_count: 123.000

Overall we see good results for the random forest classifier, the average cross fold accuracy is 0.818 with a very small stdev. We see that the model is a little biased towards labeling patients as being notAffected, evidenced by the high recall and lower precision for those data points that have the not affected label. On the flip side when the model does label a patient as being affected, we see a high degree of precision. The test split chosen was one where there was roughly equal numbers of both classes of patients.

Adaboost Performance

Params: 100 trees used, learning rate of 1.0

Cross Fold Mean: **0.687**, Cross Fold Stdev: **0.040**

For a particular split of data, we have the following metrics:

accuracy: 0.634

precision:
- all: ['0.590', '0.677'] <-notAffected, affected
- micro: 0.634
- **macro: 0.634**
- weighted: 0.638

recall:
- all: ['0.643', '0.627']
- micro: 0.634
- **macro: 0.635**
- weighted: 0.634

f1_measure:
- all: ['0.615', '0.651']
- micro: 0.634
- **macro: 0.633**
- weighted: 0.635

support:
- **y_true_dist: ['0.455', '0.545']**
- y_true_count: 123.000
- **y_predicted_dist: ['0.496', '0.504']**
- y_predicted_count: 123.000

The performance of the adaboost classifier acts more as a support of the fact that piecewise (think tree) boosting classification is picking up on some amount of signal. While the performance isn't quite as good as the Random Forest performance, it is clearly better than random, as evidenced by the mid 60 performance across all metrics. Performance could probably be improved if parameters were tuned better, but at least from this performance we know on a balanced dataset, there is hope in classifying patients.

SVM (RBF Kernel) Performance

Params: RBF Kernel, C=1

Cross Fold Mean: **0.579,** Cross Fold Stdev: **0.076**

For a particular split of data, we have the following metrics:

accuracy: 0.602

precision:
- all: ['0.544', '0.705'] <- notAffected, affected
- micro: 0.602
- **macro: 0.624**
- weighted: 0.632

recall:
- all: ['0.768', '0.463']
- micro: 0.602
- **macro: 0.615**
- weighted: 0.602

f1_measure:
- all: ['0.637', '0.559']
- micro: 0.602
- **macro: 0.598**
- weighted: 0.594

support:
- **y_true_dist: ['0.455', '0.545']**
- y_true_count: 123.000
- **y_predicted_dist: ['0.642', '0.358']**
- y_predicted_count: 123.000

An SVM classifier with a rbf kernel performed the best out of the global models, however its performance was split between the two classes. The model mostly marked data points as being notAffected (64%), and as a result had a very high recall for notAffected patients but much lower precision. This split is best seen in the confusion matrix for the classifier (slide 11). This trend that we are seeing, is probably caused by the upsampling of the notAffected data to create a balanced dataset, I would guess that this might be alleviated by using the SMOTE method to better balance the data.

SVM (Linear Kernel) Performance

Params: Linear Kernel, C=1

Cross Fold Mean: **0.541**, Cross Fold Stdev: **0.078**

For a particular split of data, we have the following metrics:

accuracy: 0.593

precision:
- all: ['0.548', '0.639'] <- notAffected, affected
- micro: 0.593
- **macro: 0.594**
- weighted: 0.598

recall:
- all: ['0.607', '0.582']
- micro: 0.593
- **macro: 0.595**
- weighted: 0.593

f1_measure:
- all: ['0.576', '0.609']
- micro: 0.593
- **macro: 0.593**
- weighted: 0.594

support:
- **y_true_dist: ['0.455', '0.545']**
- y_true_count: 123.000
- **y_predicted_dist: ['0.504', '0.496']**
- y_predicted_count: 123.000

An SVM classifier with a linear kernel performed with a little more balance between the two classes and its performance suggests that there is definitely some worth in trying to fit a global model to the data The classifier supports the trend that we have seen in all the classifiers so far that there is a slight leniency towards marking a datapoint as notAffected, evidenced by the fact that the % of points labeled as notAffected is consistently higher than the true % of points that are notAffected in all four classifiers so far.

Logistic Regression Performance

Params: penalty = L2 (though performance for L1 was very similar), solver = liblinear

Cross Fold Mean: **0.510,** Cross Fold Stdev: **0.093**

For a particular split of data, we have the following metrics:

accuracy: 0.545

precision:
- all: ['0.500', '0.608'] <- notAffected, affected
- micro: 0.545
- **macro: 0.554**
- weighted: 0.559

recall:
- all: ['0.643', '0.463']
- micro: 0.545
- **macro: 0.553**
- weighted: 0.545

f1_measure:
- all: ['0.563', '0.525']
- micro: 0.545
- **macro: 0.544**
- weighted: 0.542

support:
- **y_true_dist: ['0.455', '0.545']**
- y_true_count: 123.000
- **y_predicted_dist: ['0.585', '0.415']**
- y_predicted_count: 123.000

The worst performer of all the classifiers I ran. Did barely better than random, but kept up the trend of being biased towards labeling points as notAffected.

Conclusion

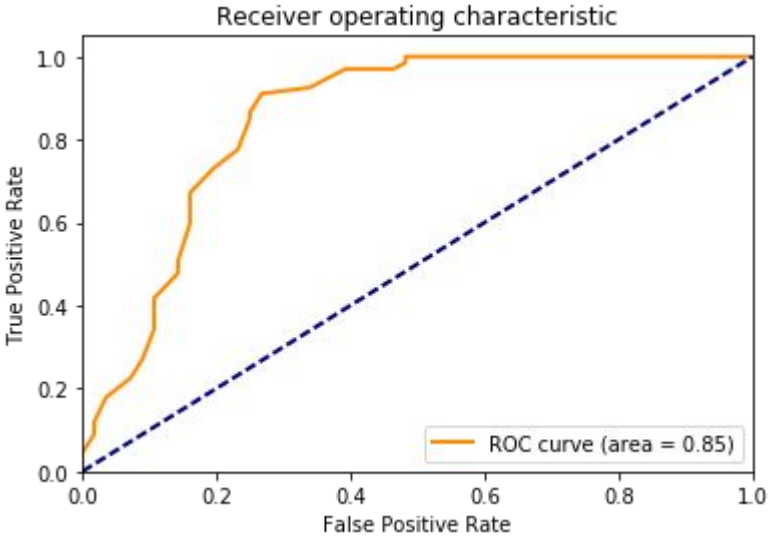
It's clear from all these classifiers that trying some of the synthetic data creation methods for upsampling will be a worthwhile exercise. Regardless I feel good that we have exhausted a good number of approaches to this problem, and that after balancing the data, removing fat finger errors and using age, gender and 4 of protein level measurements as features we were able to achieve good results with the bagging classifiers. Before starting the writing process, here are a couple steps I would like to try out, before I feel I have turned over every stone in this data dive:

1. Try out the methods described here for oversampling to see if the bias towards labeling points as notAffected was a result of the naive implementation of oversampling (Punted on for now)
2. Try running the global models on only male data and only female data (look below)
3. Try running the bagging models on gender and age alone / look at feature importance to ensure that models are indeed using protein level differences to classify patients (Done by next Monday PM)

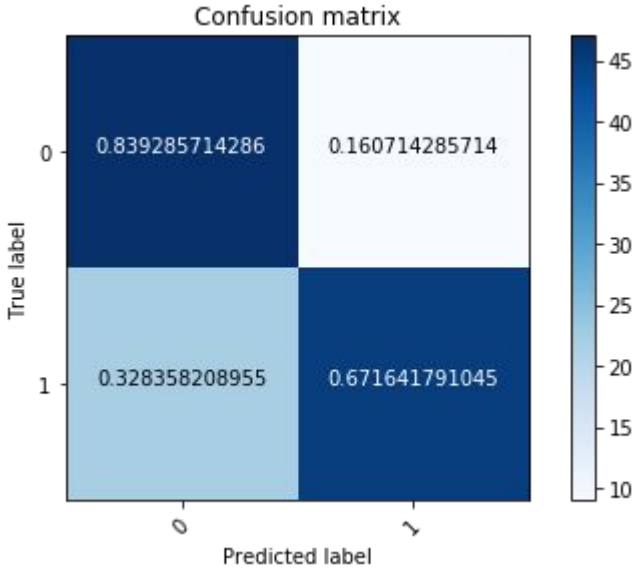
I'm still a little unclear as to how I would start writing this paper, in terms of understanding where this data came from and what is the significance of achieving good classification results on this data. I can definitely talk about how I prepared the data, what my initial dives looked like (bar plots, time series visualization, regression analysis) and then finally all the classification models I used. If you are free next week for a skype call that would be much appreciated.

Graphs
(for the same split of test data used in the previous slides)

Random Forest

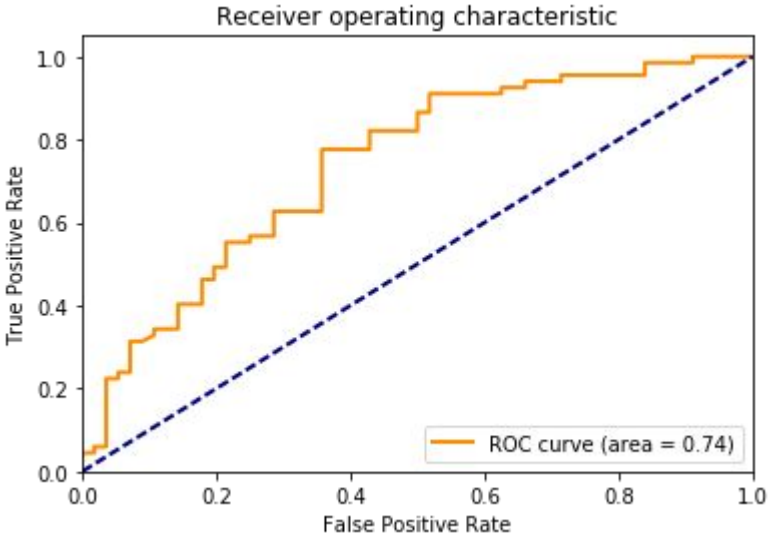


Normalized confusion matrix
[[0.84 0.16]
[0.33 0.67]]

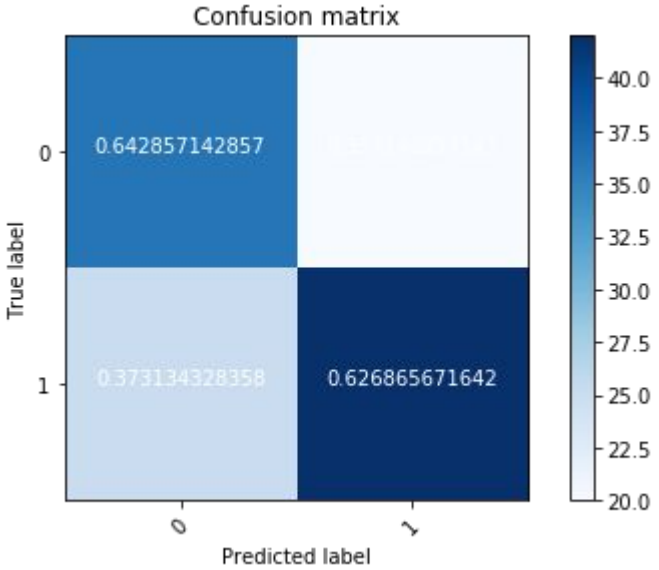


Graphs
(for the same split of test data used in the previous slides)

Adaboost

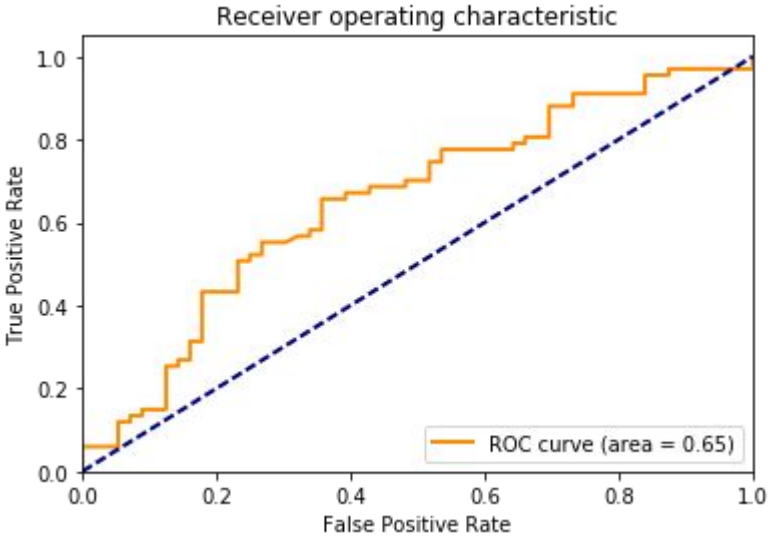


Normalized confusion matrix
[[0.64 0.36]
[0.37 0.63]]

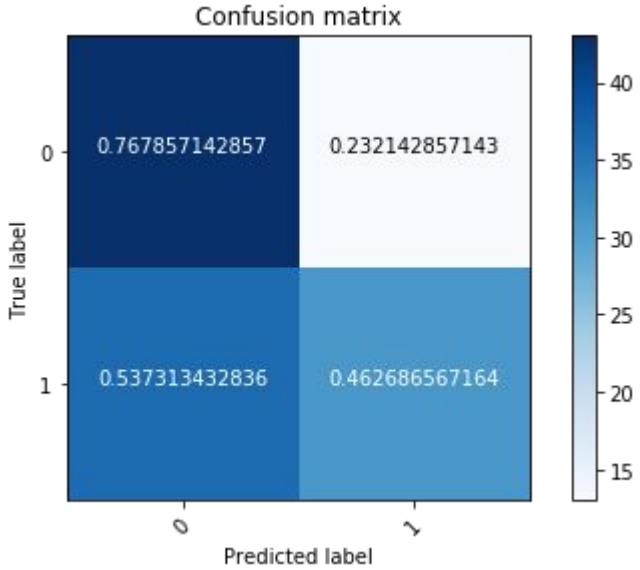


Graphs
(for the same split of test data used in the previous slides)

SVM (RBF Kernel)

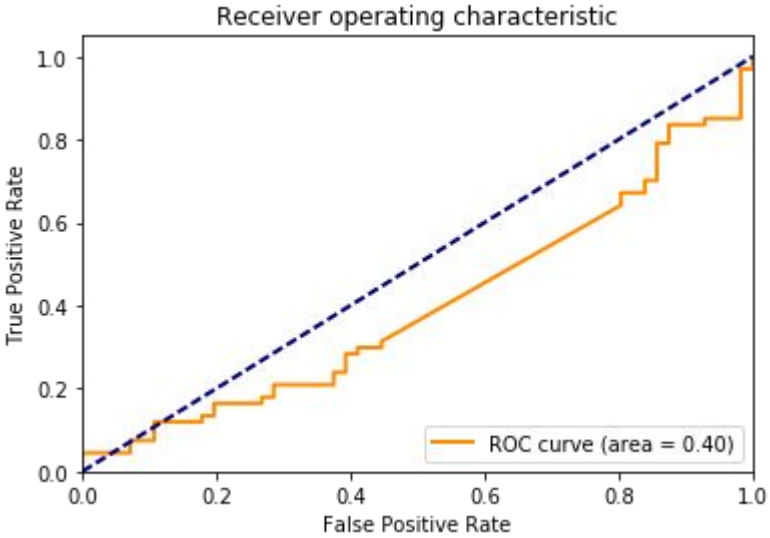


Normalized confusion matrix
[[0.77 0.23]
[0.54 0.46]]

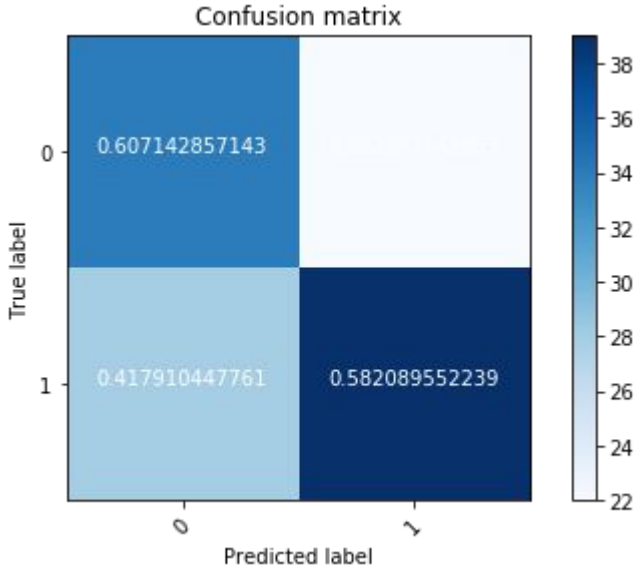


Graphs
(for the same split of test data used in the previous slides)

SVM (Linear Kernel)

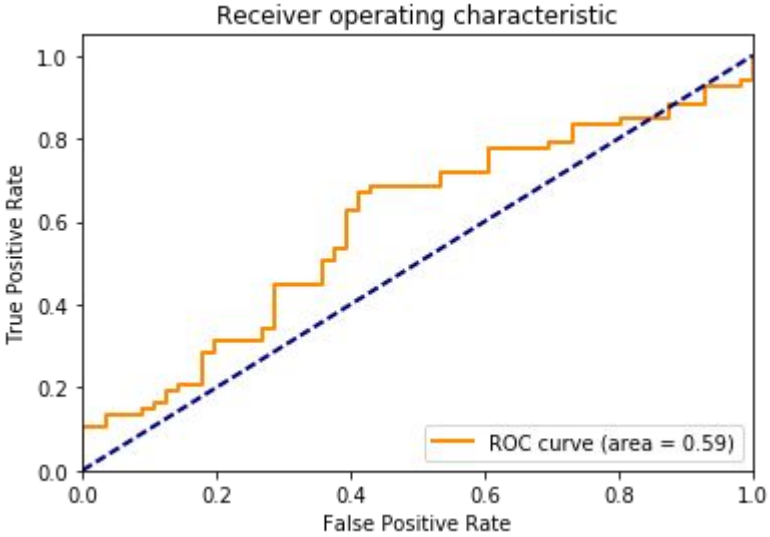


Normalized confusion matrix
[[0.61 0.39]
[0.42 0.58]]



Graphs
(for the same split of test data used in the previous slides)

Logistic Regression



Normalized confusion matrix
[[0.64 0.36]
[0.54 0.46]]