README.md

SVM vs Random Forest?

C. Augusto Suarez: University of New Brunswick, Dec 2019

This project is a Comparison of a support vector machine and a random forest classifier on the same dataset: the BNG_Heart_Statlog data from OpenML. This dataset contains one million entries of 13 features and a 14th "class" column: the Absence/0 or Presence/1 of heart disease in the subject. It is a python built project using jupyter notebooks and the necessary enviornment can be built from the heart_statlog.yml file included. I also included the dataset for 2 reasons:

- The dataset is small enough to upload to github
- The dataset is already openly available on OpenML; For private datasets, I strongly believe that *the enclosed data should be protected and kept private under **all** circumstances*.

The Process

Initial Visualizations

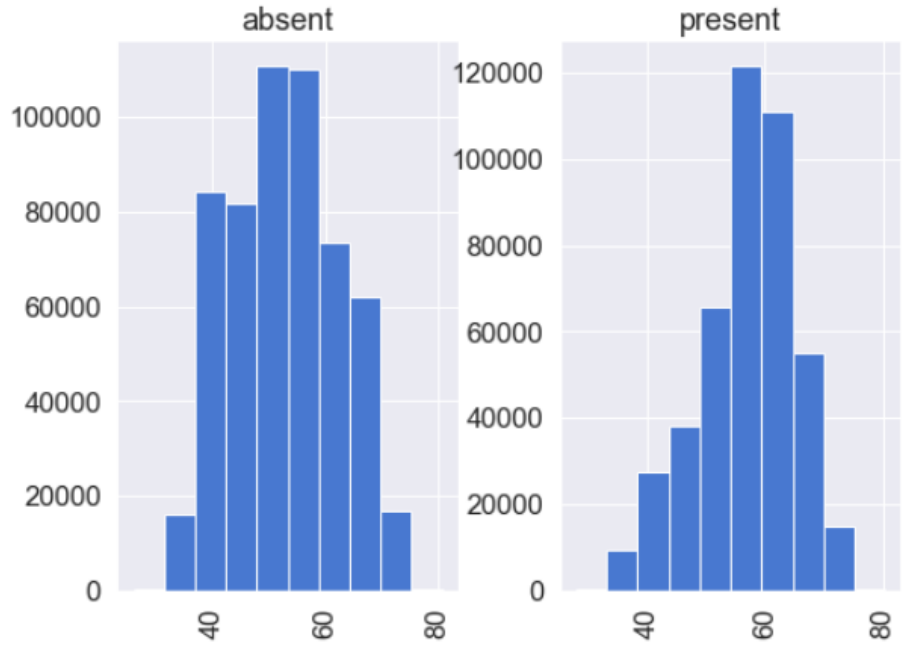
We take an initial gander at the dataset and observe 13 features as input, with a binary output: the presence or absence of heart disease. We can see that all of our input data is numerical but our output data is categorical of type string.

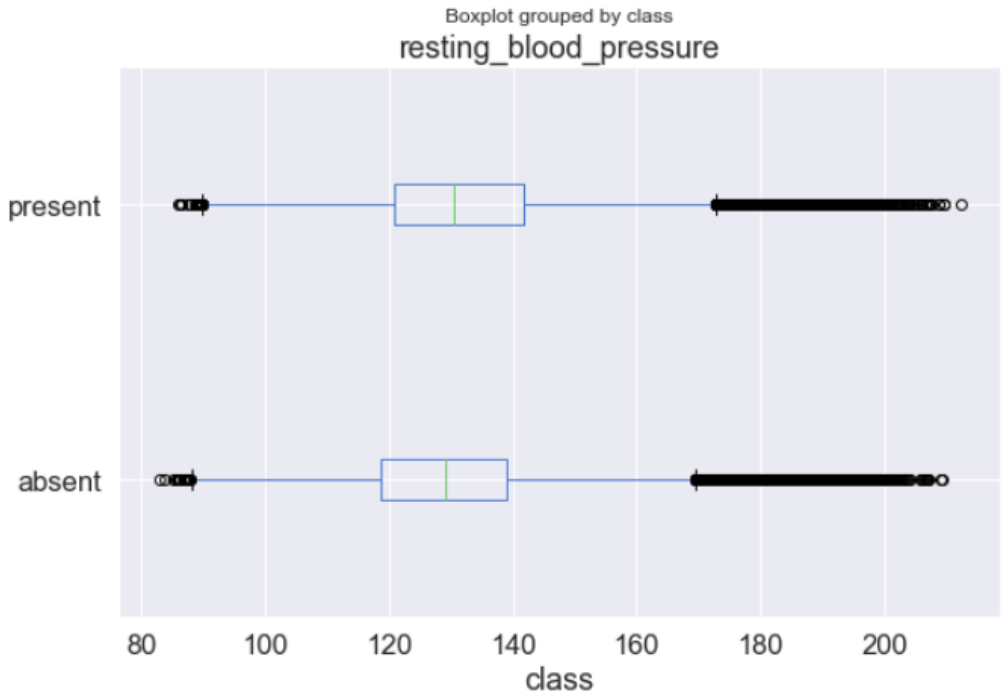
```
            age  sex     chest  resting_blood_pressure  serum_cholestoral  \
0     53.494725    1  1.150395              117.978412         242.009370
1     37.320375    0  1.887693              118.455670         218.156844
2     48.520214    1  3.000000              141.819366         173.382704
3     59.587959    0  4.000000              106.368725         222.732859
4     58.805677    1  3.000000              121.035286         257.257441
...         ...  ...       ...                     ...                ...
1995  52.749168    1  4.000000              144.225749         226.073033
1996  51.394149    1  2.142255              131.160264         248.009946
1997  47.780269    1  3.000000              116.555172         195.984541
1998  58.099015    1  4.000000              120.360043         247.371229
1999  59.325386    1  1.910519              138.064760         257.154476

      fasting_blood_sugar  resting_electrocardiographic_results  \
0                       0                                     0
1                       1                                     2
2                       0                                     2
3                       0                                     2
4                       0                                     0
...                   ...                                   ...
1995                    0                                     2
1996                    0                                     2
1997                    0                                     0
1998                    0                                     0
1999                    0                                     2

      maximum_heart_rate_achieved  exercise_induced_angina   oldpeak  slope  \
0                      133.361344                        0  3.089391      2
1                      148.458625                        0  0.000000      3
2                      141.198191                        0  1.071691      2
3                      141.659888                        1  0.866638      2
4                      145.333117                        0  1.212600      3
...                           ...                      ...       ...    ...
1995                   132.088962                        1  0.971895      3
1996                   118.974799                        0  1.355360      2
1997                   171.761064                        1  0.444107      2
1998                   145.284331                        1  3.034997      2
1999                   161.833016                        0  3.192773      3

      number_of_major_vessels  thal
0                           1     3
1                           0     3
2                           0     6
3                           0     7
4                           0     7
...                       ...   ...
1995                        1     7
1996                        0     7
1997                        0     7
1998                        1     7
1999                        1     7

[2000 rows x 13 columns]
0       present
1        absent
2        absent
3       present
4        absent
         ...
1995    present
1996    present
1997     absent
1998    present
1999    present
Name: class, Length: 2000, dtype: object
```

We can easily fix this problem by using a nominal converter to get numerical output instead. Seen below:

```
             age  sex     chest  resting_blood_pressure  serum_cholestoral  \
0      53.494725    1  1.150395              117.978412         242.009370
1      37.320375    0  1.887693              118.455670         218.156844
2      48.520214    1  3.000000              141.819366         173.382704
3      59.587959    0  4.000000              106.368725         222.732859
4      58.805677    1  3.000000              121.035286         257.257441
...          ...  ...       ...                     ...                ...
1995   52.749168    1  4.000000              144.225749         226.073033
1996   51.394149    1  2.142255              131.160264         248.009946
1997   47.780269    1  3.000000              116.555172         195.984541
1998   58.099015    1  4.000000              120.360043         247.371229
1999   59.325386    1  1.910519              138.064760         257.154476

      fasting_blood_sugar  resting_electrocardiographic_results  \
0                       0                                     0
1                       1                                     2
2                       0                                     2
3                       0                                     2
4                       0                                     0
...                   ...                                   ...
1995                    0                                     2
1996                    0                                     2
1997                    0                                     0
1998                    0                                     0
1999                    0                                     2

      maximum_heart_rate_achieved  exercise_induced_angina   oldpeak  slope  \
0                      133.361344                        0  3.089391      2
1                      148.458625                        0  0.000000      3
2                      141.198191                        0  1.071691      2
3                      141.659888                        1  0.866638      2
4                      145.333117                        0  1.212600      3
...                           ...                      ...       ...    ...
1995                   132.088962                        1  0.971895      3
1996                   118.974799                        0  1.355360      2
1997                   171.761064                        1  0.444107      2
1998                   145.284331                        1  3.034997      2
1999                   161.833016                        0  3.192773      3

      number_of_major_vessels  thal
0                           1     3
1                           0     3
2                           0     6
3                           0     7
4                           0     7
...                       ...   ...
1995                        1     7
1996                        0     7
1997                        0     7
1998                        1     7
1999                        1     7

[2000 rows x 13 columns]
0       1
1       0
2       0
3       1
4       0
       ..
1995    1
1996    1
1997    0
1998    1
1999    1
Name: class, Length: 2000, dtype: int64
```

After we've done our initial exploration, we continue by looking at visualizations of what I think are intuitive relationships. In this case, I looked at the distribution of ages among both classes. Here we can see that, although not a deciding factor, the age distributions of both classes are, in fact, slightly different. We can also see that the spreads of resting blood pressure are similar among both classes but not identical.





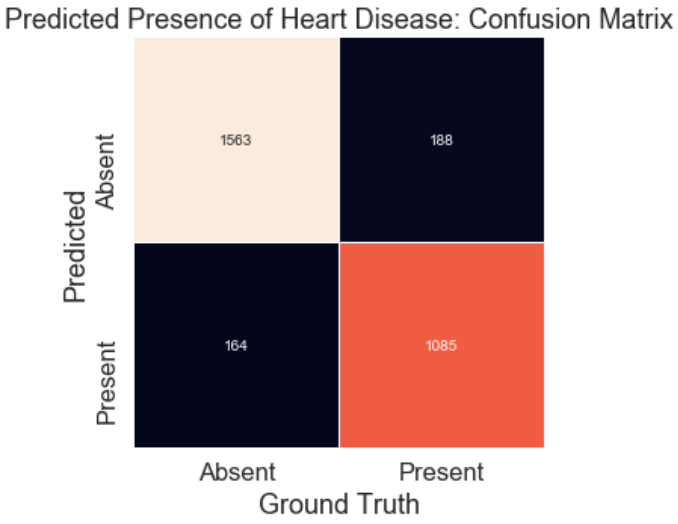
Model Selection

After this, I decided on the models that I wanted to compare for this particular dataset. Even though neural networks have had remarkable success classifying similar data, I wanted to try machine learning models that I had not implemented before so I chose to look at SVM's and Random Forests.

- SVM's because they have the ability to learn nonlinear relationships between features using kernel methods. We can also, for the same reason, test it earlier without having to preprocess our data excessively.
- Random Forests because if the relationships are not of high order the model will perform better, train faster and scale to a larger subset of the data with more accuracy.

Model Training

As mentioned previously, we could use the SVM 'kernel trick' to predict with almost no data preparation. And we can see below that after only a only a few tuned hyperparameters. The SVM already classifies the small subset with high accuracy. Below is the confusion matrix with 20000 samples and a radial basis function kernel.



SVM Tuning

In this section, we will describe a series of concise steps taken in an attempt to increase the model accuracy and scale it to the desired size of one million entries (the entire test set). I used the parameters in the best trainings to fine tune my svm for the following group of trainings. For this, I made use of the *GridSearchCV* library which greatly streamlined my workflow because of its capacity to automate several trainings at once; For simple machine learning tasks, I highly recommend it. In this case: the steps (almost all of them, at least) we followed for tuning were as follows:

- We increased the sample size from *20000 to 40000* and added the default *3rd degree polynomial* kernel to our kernel searchspace parameters. We used a *slack variable selection of 10, 50 and 100* as well as a selection of *1k, 5k and 10k for maximum number of iterations*. We started the tuning by using a *2 fold cross validation* in our grid search. Training time: 5 min, 46 s.

	mean_fit_time	std_fit_time	mean_score_time	params	split0_test_score	split1_test_score	mean_test_score	std_test_score	rank_test_score
17	8.663867	0.009016	4.314955	{'C': 100, 'kernel': 'rbf', 'max_iter': 10000}	0.877419	0.878758	0.878088	0.000669	1
11	9.075226	0.120178	4.794215	{'C': 50, 'kernel': 'rbf', 'max_iter': 10000}	0.877125	0.876934	0.877029	0.000095	2
16	8.275860	0.426856	4.425071	{'C': 100, 'kernel': 'rbf', 'max_iter': 5000}	0.876831	0.877228	0.877029	0.000199	2
10	8.848827	0.346261	4.730819	{'C': 50, 'kernel': 'rbf', 'max_iter': 5000}	0.876595	0.875934	0.876265	0.000331	4
2	7.792671	0.248858	3.432300	{'C': 10, 'kernel': 'poly', 'max_iter': 10000}	0.872890	0.872522	0.872706	0.000184	5

	mean_fit_time	std_fit_time	mean_score_time	params	split0_test_score	split1_test_score	mean_test_score	std_test_score	rank_test_score
12	1.061638	0.024414	0.756474	{'C': 100, 'kernel': 'poly', 'max_iter': 1000}	0.444386	0.627508	0.535941	0.091561	14
9	2.022568	0.056871	1.506990	{'C': 50, 'kernel': 'rbf', 'max_iter': 1000}	0.511970	0.442379	0.477176	0.034795	15
3	1.994137	0.059325	1.499014	{'C': 10, 'kernel': 'rbf', 'max_iter': 1000}	0.450621	0.442967	0.446794	0.003827	16
15	2.450962	0.081264	1.616160	{'C': 100, 'kernel': 'rbf', 'max_iter': 1000}	0.442033	0.449026	0.445529	0.003497	17
0	1.363350	0.018992	0.960454	{'C': 10, 'kernel': 'poly', 'max_iter': 1000}	0.442621	0.444732	0.443676	0.001056	18

- From these results we increased the resolution of the *slack variable to go from 10-100* in increments of 10, and changed the *max iterations to 5k, 7.5k, and 10k*. Training time: 23 min, 18 s.

	mean_fit_time	std_fit_time	mean_score_time	params	split0_test_score	split1_test_score	mean_test_score	std_test_score	rank_test_score
59	8.659144	0.008326	4.319487	{'C': 100, 'kernel': 'rbf', 'max_iter': 10000}	0.877419	0.878758	0.878088	0.000669	1
52	8.618943	0.025931	4.370328	{'C': 90, 'kernel': 'rbf', 'max_iter': 7500}	0.877184	0.878934	0.878059	0.000875	2
58	8.587030	0.019948	4.332909	{'C': 100, 'kernel': 'rbf', 'max_iter': 7500}	0.877007	0.878993	0.878000	0.000993	3
46	8.778112	0.043478	4.418700	{'C': 80, 'kernel': 'rbf', 'max_iter': 7500}	0.877066	0.878522	0.877794	0.000728	4
53	8.781028	0.090278	4.404700	{'C': 90, 'kernel': 'rbf', 'max_iter': 10000}	0.877066	0.878405	0.877735	0.000669	5

	mean_fit_time	std_fit_time	mean_score_time	params	split0_test_score	split1_test_score	mean_test_score	std_test_score	rank_test_score
55	5.708730	0.092750	2.540225	{'C': 100, 'kernel': 'poly', 'max_iter': 7500}	0.663784	0.846579	0.755176	0.091397	56
42	4.635845	0.025350	2.450051	{'C': 80, 'kernel': 'poly', 'max_iter': 5000}	0.810423	0.682040	0.746235	0.064191	57
30	4.838152	0.014973	2.557553	{'C': 60, 'kernel': 'poly', 'max_iter': 5000}	0.678254	0.793164	0.735706	0.057455	58
24	4.971840	0.031764	2.622808	{'C': 50, 'kernel': 'poly', 'max_iter': 5000}	0.608435	0.841991	0.725206	0.116778	59
54	4.354016	0.123331	2.364695	{'C': 100, 'kernel': 'poly', 'max_iter': 5000}	0.544380	0.747044	0.645706	0.101332	60

- Then we increased *2 fold validation to 3 fold validation* in an attempt to check for overfitting, *slack of only 90, iterations of 7.5k*. Training time: 2 min, 10 s.

mean_fit_time	std_fit_time	mean_score_time	params	split0_test_score	split1_test_score	split2_test_score	mean_test_score	std_test_score	rank_test_score
16.972378	1.551206	4.197940	{'C': 90, 'kernel': 'rbf', 'max_iter': 7500}	0.879389	0.874713	0.880604	0.878235	0.002539	1
9.276382	0.532255	2.372331	{'C': 90, 'kernel': 'poly', 'max_iter': 7500}	0.660402	0.556467	0.642429	0.619765	0.045357	2

mean_fit_time	std_fit_time	mean_score_time	params	split0_test_score	split1_test_score	split2_test_score	mean_test_score	std_test_score	rank_test_score
16.972378	1.551206	4.197940	{'C': 90, 'kernel': 'rbf', 'max_iter': 7500}	0.879389	0.874713	0.880604	0.878235	0.002539	1
9.276382	0.532255	2.372331	{'C': 90, 'kernel': 'poly', 'max_iter': 7500}	0.660402	0.556467	0.642429	0.619765	0.045357	2

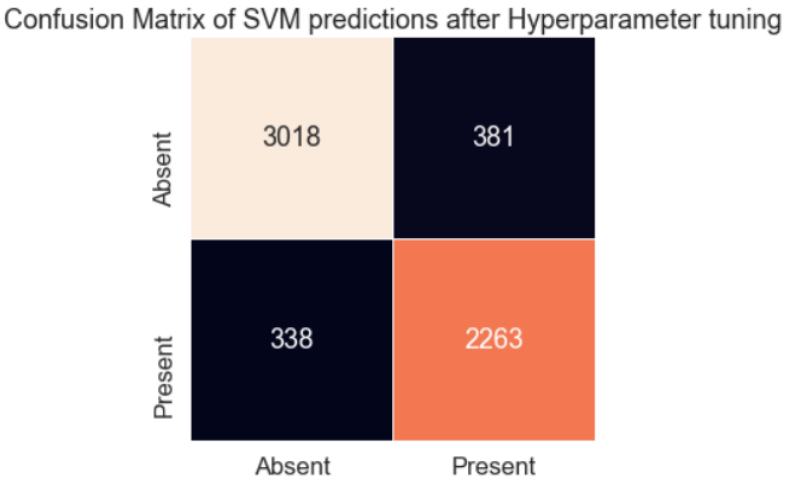
- Added 'linear' kernel to search space parameters and went up to *4 fold* validation. Training time: 2 min, 46 s.

	mean_fit_time	std_fit_time	mean_score_time	params	split0_test_score	split1_test_score	split2_test_score	split3_test_score	mean_test_score	std_test_s
2	17.370211	0.141349	3.166293	{'C': 90, 'kernel': 'rbf', 'max_iter': 7500}	0.878955	0.873059	0.880000	0.878809	0.877706	0.00
0	10.025362	0.160700	1.782017	{'C': 90, 'kernel': 'poly', 'max_iter': 7500}	0.858723	0.748353	0.845647	0.623956	0.769176	0.09
1	2.023611	0.033900	0.117935	{'C': 90, 'kernel': 'linear', 'max_iter': 7500}	0.406658	0.508588	0.469765	0.465231	0.462559	0.03

- Removed 'linear' and 'poly' kernels from search space parameters and added sigmoid also to check. Confusion matrix and classification report to "see if model performs as advertised". Training time: 4 min, 12 s.

d_fit_time	mean_score_time	params	split0_test_score	split1_test_score	split2_test_score	split3_test_score	mean_test_score	std_test_score	rank_test_score
0.248856	3.181271	{'C': 90, 'kernel': 'rbf', 'max_iter': 7500}	0.878955	0.873059	0.880000	0.878809	0.877706	0.002722	1
0.762241	7.090637	{'C': 90, 'kernel': 'sigmoid', 'max_iter': 7500}	0.547347	0.546000	0.546588	0.538887	0.544706	0.003393	2

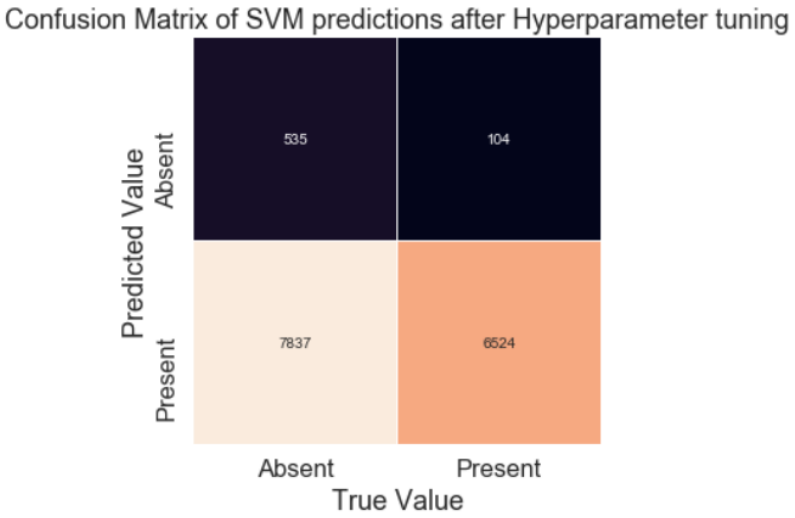
```
Accuracy: 0.8801666666666667
Precision: 0.8700499807766243
Recall: 0.8559001512859304
Sample size 40000, we want to validate that the svm actually performs 'as advertised' so we start making confusion matrices
```



- Here is where I wanted to see if the model would scale up. As such, I decided to *increase to sample size from 40000 samples to 100,000 samples*. and let it run over night. I was also using only 'rbf' for the kernel parameter and a slack variable selction of 80-100 in increments of 5. Training time: 33 min, 32 s.

	mean_fit_time	std_fit_time	mean_score_time	params	split0_test_score	split1_test_score	split2_test_score	split3_test_score	mean_test_score	std_test_s
1	78.264342	0.297184	17.072810	{'C': 85, 'kernel': 'rbf', 'max_iter': 10000}	0.649334	0.550845	0.722716	0.765165	0.672012	0.08
0	78.788051	0.786422	17.306097	{'C': 80, 'kernel': 'rbf', 'max_iter': 10000}	0.531175	0.588490	0.724222	0.653960	0.624459	0.07
2	78.392117	0.241711	17.027535	{'C': 90, 'kernel': 'rbf', 'max_iter': 10000}	0.598701	0.520399	0.775566	0.565203	0.614965	0.09
4	77.984320	0.186527	17.030373	{'C': 100, 'kernel': 'rbf', 'max_iter': 10000}	0.578232	0.547974	0.589957	0.722481	0.609659	0.06
3	78.300485	0.226358	17.018934	{'C': 95, 'kernel': 'rbf', 'max_iter': 10000}	0.567173	0.529622	0.526331	0.554661	0.544447	0.01

```
Accuracy: 0.4706
Precision: 0.45428591323723977
Recall: 0.9843089921544961
Sample size 100000, after first confusion matrix lets increase the sample size again and let it run overnight. This way we can
verify if accuracy holds with larger dataset
```



Here we reach a point where we need to do some more work before we can continue scaling

We can see that the increase in samples fed into the SVM training drastically took a toll on our accuracy metrics. This happens because as our training data increases, so do our number of support vectors, meaning that our classification boundaries have a harder time generalizing to unseen data. To attempt to fix this, I tried several approaches separately and in a pipeline.

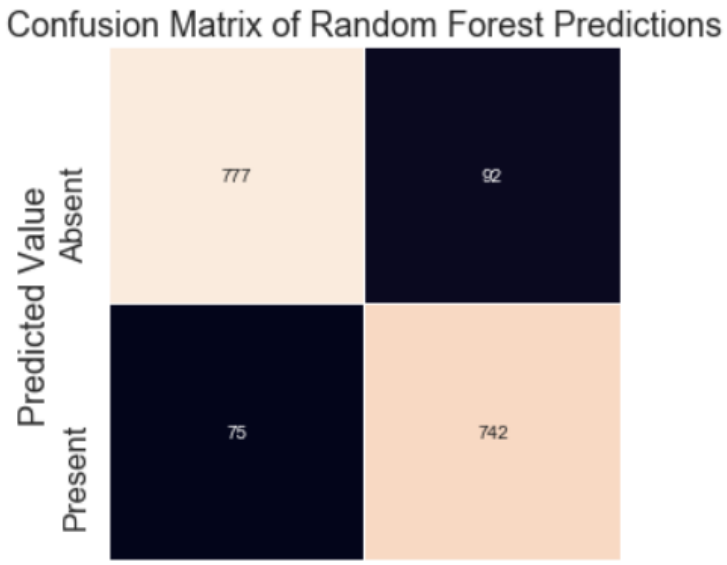
- The first thing I tried was a simple normalization: Every data point was normalized to mean 0 and standard deviation 1.
- When this failed to make a noticeable impact, I looked at principal component analysis and attempted to train on the new transformed, pca-fit data. I also tried tuning the number of iterations that the PCA did on the data.
- When PCA failed, I tried undersampling AND oversampling techniques to see if it was an class imbalance problem. For undersampling I used random undersampling, which takes a random subset of the *majority* class of equal size to the under-represented class to account for the imbalance. For oversampling, I used scikit-learn's implementation of **S**ynthetic **M**inority **O**versampling **TE**chiniques (SMOTE) and applied it to the *minority* class. This technique uses points in the minority class, calculates the difference vector to each of its K-nearest neighbors, then creates a new data point somewhere along that vector by using its product with a random number between 0 and 1.

No combination of these techniques produced more adequate results, so I turned my attention to my second machine learning algorithm: the **Random Forest Classifier**.

Random Forest Tuning

- After the initial tuning stages of the random forest, I arrived suprisingly quickly at an acceptable classifier for a *sample size of 10000* with smote upsampling and *a 4-fold cross validation*.

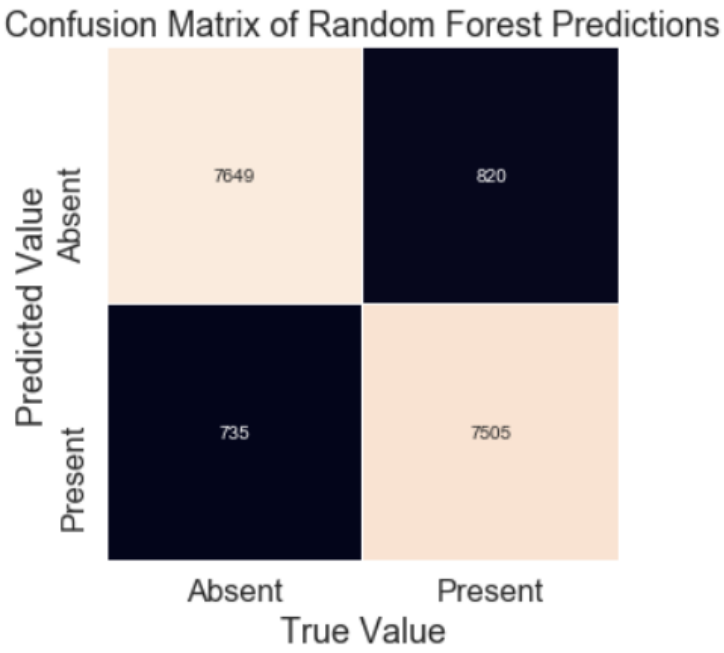
	precision	recall	f1-score	support
0	0.89	0.91	0.90	852
1	0.91	0.89	0.90	834
accuracy			0.90	1686
macro avg	0.90	0.90	0.90	1686
weighted avg	0.90	0.90	0.90	1686



- Maintaining the upsampling technique, I scaled the sample size from *10k to 100k samples* and wanted to see how our new classifier would perform at the same size that our SVM went awry. In our search space, we used *100, 500, and 1000 as our number of estimators*, we used *None (Default), 16, and 64 as our max leaf nodes* and *None (Default, 8 and 16) as our max depth parameter*. Essentially I wanted to see if there was a way that we could maximize our accuracy metrics while using effective parameters for a shorter training time.

mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_max_depth	param_max_leaf_nodes	param_n_estimators	params	split0_test_scor
90.410445	1.868103	3.922258	0.095859	16	None	500	{'max_depth': 16, 'max_leaf_nodes': None, 'n_e...	0.90811
182.647543	3.407902	7.956617	0.192155	16	None	1000	{'max_depth': 16, 'max_leaf_nodes': None, 'n_e...	0.90752
183.203560	0.936608	8.226387	0.036824	None	None	1000	{'max_depth': None, 'max_leaf_nodes': None, 'n...	0.90684
91.541070	0.691638	4.116603	0.019617	None	None	500	{'max_depth': None, 'max_leaf_nodes': None, 'n...	0.90710
17.854215	0.309265	0.777442	0.005165	16	None	100	{'max_depth': 16, 'max_leaf_nodes': None, 'n_e...	0.90684

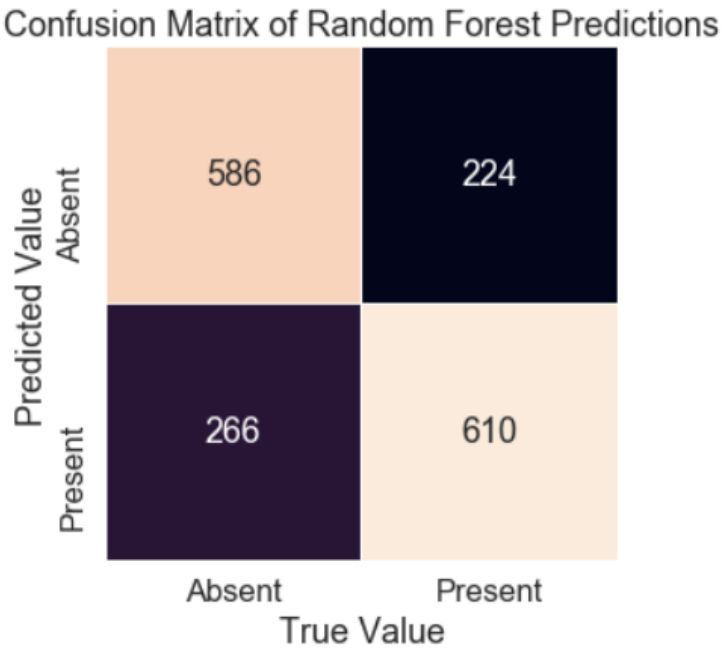
```
              0       0.90      0.91      0.91      8384
              1       0.91      0.90      0.91      8325

       accuracy                           0.91     16709
      macro avg       0.91      0.91      0.91     16709
   weighted avg       0.91      0.91      0.91     16709
```



- From the results above, I noticed that increasing the estimators from 500 and 1000 had no real impact on accuracy but doubled our training time. As such, I decided to see if I could improve the estimator on streamlined parameters by running the same training on the *PCA-fit-transformed* data. My streamlined parameters were *n_estimators: 500, 1000 (honestly probably just forgot to remove the 1000)* and *max depth 16 and None*. Training time: 3 min, 28 s

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_max_depth	param_max_leaf_nodes	param_n_estimators	params	split0_test_s
3	21.445260	0.383719	0.862175	0.032081	16	None	1000	{'max_depth': 16, 'max_leaf_nodes': None, 'n_e...	0.71
1	22.040696	0.366387	0.903287	0.067937	None	None	1000	{'max_depth': None, 'max_leaf_nodes': None, 'n...	0.71
2	10.518409	0.186738	0.426109	0.024728	16	None	500	{'max_depth': 16, 'max_leaf_nodes': None, 'n_e...	0.71
0	11.120979	0.524395	0.438548	0.023613	None	None	500	{'max_depth': None, 'max_leaf_nodes': None, 'n...	0.71

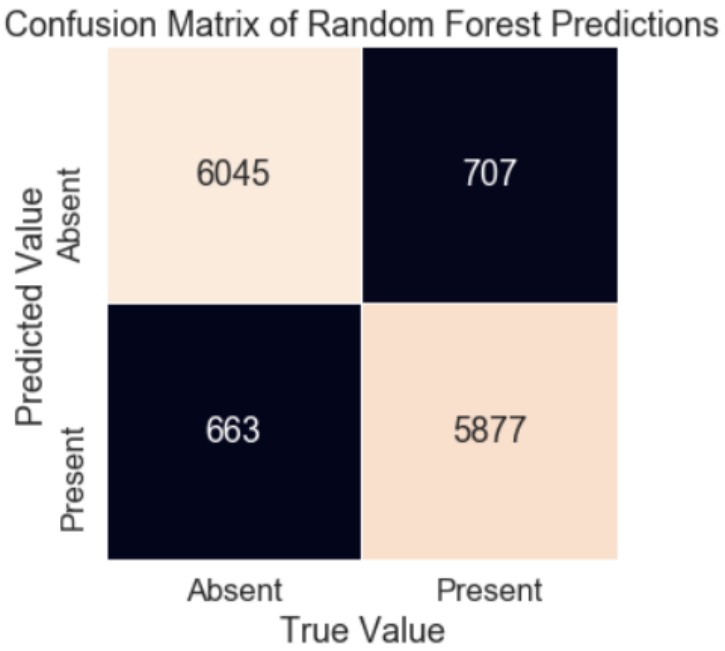
```
              0       0.72      0.69      0.71       852
              1       0.70      0.73      0.71       834

       accuracy                           0.71      1686
      macro avg       0.71      0.71      0.71      1686
   weighted avg       0.71      0.71      0.71      1686
```



- I also in parallel ran a training with the same search space but *without PCA* and with *downsampled* data. The results were inline with the non-pca results with upsampling from the step above. Training time: 32 min, 32 s

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_max_depth	param_max_leaf_nodes	param_n_estimators	params	split0_test_s
2	69.359679	0.657515	3.400653	0.211308	16	None	500	{'max_depth': 16, 'max_leaf_nodes': None, 'n_e...	0.89
3	133.795372	1.368959	6.354284	0.097263	16	None	1000	{'max_depth': 16, 'max_leaf_nodes': None, 'n_e...	0.89
0	70.058522	2.911247	3.309319	0.054433	None	None	500	{'max_depth': None, 'max_leaf_nodes': None, 'n...	0.89
1	145.133632	3.037852	7.090715	0.415743	None	None	1000	{'max_depth': None, 'max_leaf_nodes': None, 'n...	0.89

	precision	recall	f1-score	support
0	0.90	0.90	0.90	6708
1	0.90	0.89	0.90	6584
accuracy			0.90	13292
macro avg	0.90	0.90	0.90	13292
weighted avg	0.90	0.90	0.90	13292

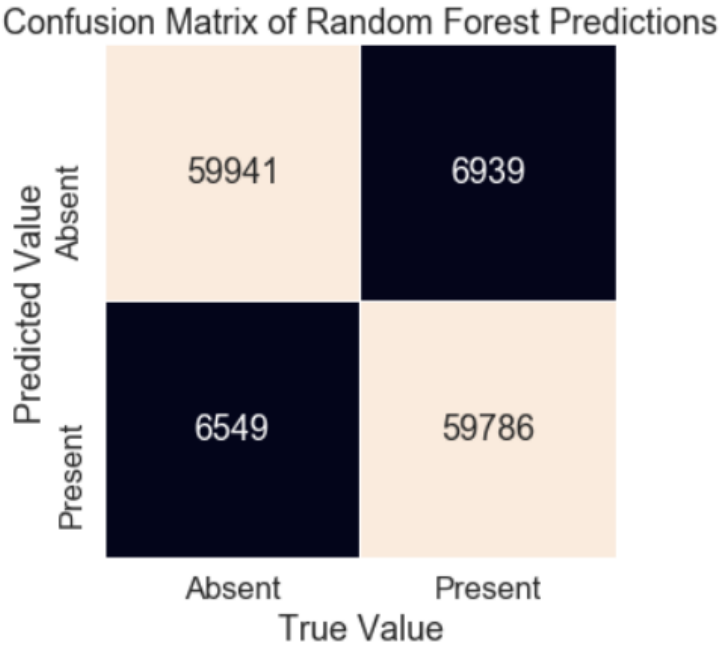


Note: with the same parameters, PCA (as expected) ran in a MUCH faster training time, but had a drastic loss in accuracy, so the time gain is less enticing.

- Now the big one: I let the model sit overnight with our streamlined parameters {'n_estimators' : [500], 'max_leaf_nodes' : [None], 'max_depth' : [16] } and on the entire dataset (minus 10, actually, for a reason that will be explained later). Training time: 2 hours, 13 min, 38 s.

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_max_depth	param_max_leaf_nodes	param_n_estimators	params	split0_test_s
0	1131.073169	27.25878	42.052197	0.772673	16	None	500	{'max_depth': 16, 'max_leaf_nodes': None, 'n_e...	0.89

	precision	recall	f1-score	support
0	0.90	0.90	0.90	66490
1	0.90	0.90	0.90	66725
accuracy			0.90	133215
macro avg	0.90	0.90	0.90	133215
weighted avg	0.90	0.90	0.90	133215



The Results

The final random forest model performs well over my desired target (80%) and was able to scale to the entire 1 million data points! It trained in a very reasonable time and generalized well to our hold out sets over 4 Fold Cross Validation. With more data preparation and tuning I am sure that I could achieve an even better result. But this is satisfactory for now.

One More Thing

Remember those last 10 data points that I held out from our dataset? I wanted to to see, if I took the last 10 points and laid out their ground truths, what the model would predict about them. This was more of a whimsical excercise than an actual motivated research technique, but it was still an interesting thought. The results are below:

	Predicted	Ground Truth
0	1	1
1	1	0
2	1	1
3	0	0
4	0	0
5	0	0
6	1	1
7	1	1
8	0	0
9	1	1

As was expected with a 90% model, the random forest classifier correctly categorized 9 out the 10 last data entries. And with that, I conclude my first investigation on the heart_statlog dataset. But now it's obvious that I should ask: what else could I do? Well, with the work that's already been done, I can think of a few things that might improve our random forest classifier:

- Cleaning our data with other techniques. Maybe a different feature extraction method using composite or filtered attributes.
- Seeing what techniques like varimax rotation could do to our PCA's. For that matter, seeing how we could take advantage of the PCA's in any way! Seeing as the dimensionality reduction drastically decreased our training time
- Trying other sampling techniques such as ADASYN or maybe a *non*-random undersampling
- Running a slightly modified random forest algorithm, such as a boosted tree

Maybe I'll come back to this at a later stage but for now, these results suffice.