Predicting Heart Disease From Physiological Indicators

Contents

Authors: Natalie Cho, Yurui Feng, Elena Ganacheva, Tony Zoght; The University of British Columbia, Master of Data Science.
2022-11-24

Introduction

Responsible for 16% of the world's total deaths in 2019, heart disease is the world's leading cause of death according to the World Health Organization. The development of heart disease can not be contributed to a single factor in isolation, making early detection difficult given so many risk factors.

The goal of this project is to predict the presence of heart disease based on common early signs and the Heart Disease UCI dataset from the UC Irvine Machine Learning Repository to answer the question: given common early signs and physiological indicators, can we predict the presence of cardiac disease based on symptoms such as chest pain, blood pressure, or resting ECG?

Responding to this question may aid in the early detection of heart disease and may help with earlier treatment, crucial to improving an individual's chances of survival.

Methods

Data

The heart disease data set [Janosi *et al.*, 1988]used in this project is obtained from the UC Irvine machine learning repository [Dua and Graff, 2017]. The creators Andras et al. originally donated the dataset in 1988 with 76 features. Nevertheless, the dataset we use contains 13 features with a binary target variable of 0 and 1, where 0 indicates no presence of heart and 1 indicates presence. Out of the 13 features, there are 8 categorical features and 5 numeric features. These features includes various physiological parameters like resting blood pressure and serum cholestoral levels, as well as potential signs of heart disease like chest pain. The original paper utilized Bayesian model to estimate the probability of having heart disease presence [Detrano *et al.*, 1989].

There are 303 observations in the heart disease dataset with no missing values. We have a slightly imbalance dataset: there are 165 positive cases (i.e. presence of heart disease) and 138 negative cases. We split the dataset into training and test data in a stratified fashion. The number of cases in the two splits is shown in the table below:

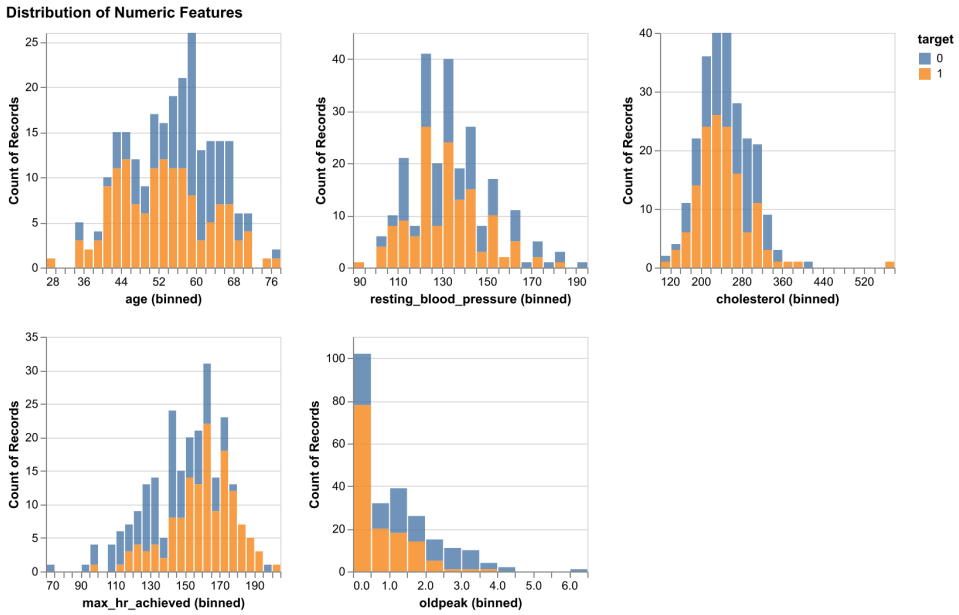
```
import pandas as pd
```

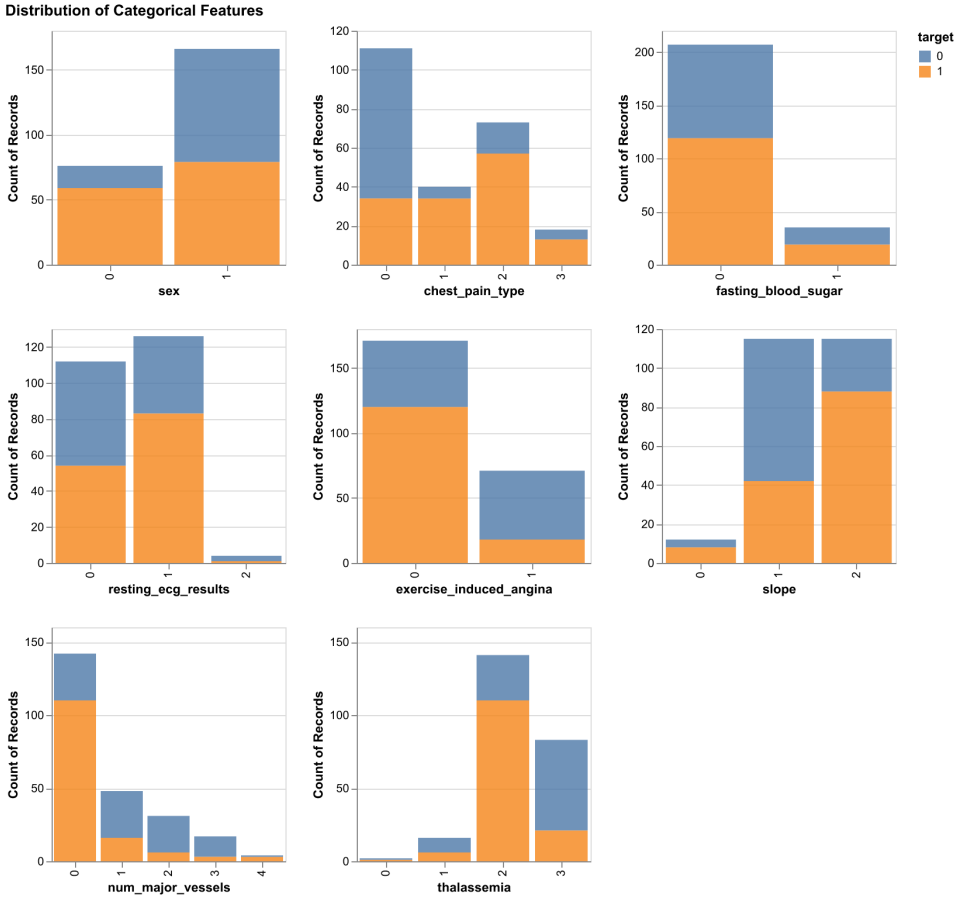
```
pd.read_csv("../../results/class_count.csv", index_col = 0)
```

	Presence of HD:	No presence of HD:
Training	138	104
Test	27	34

EDA and Methods

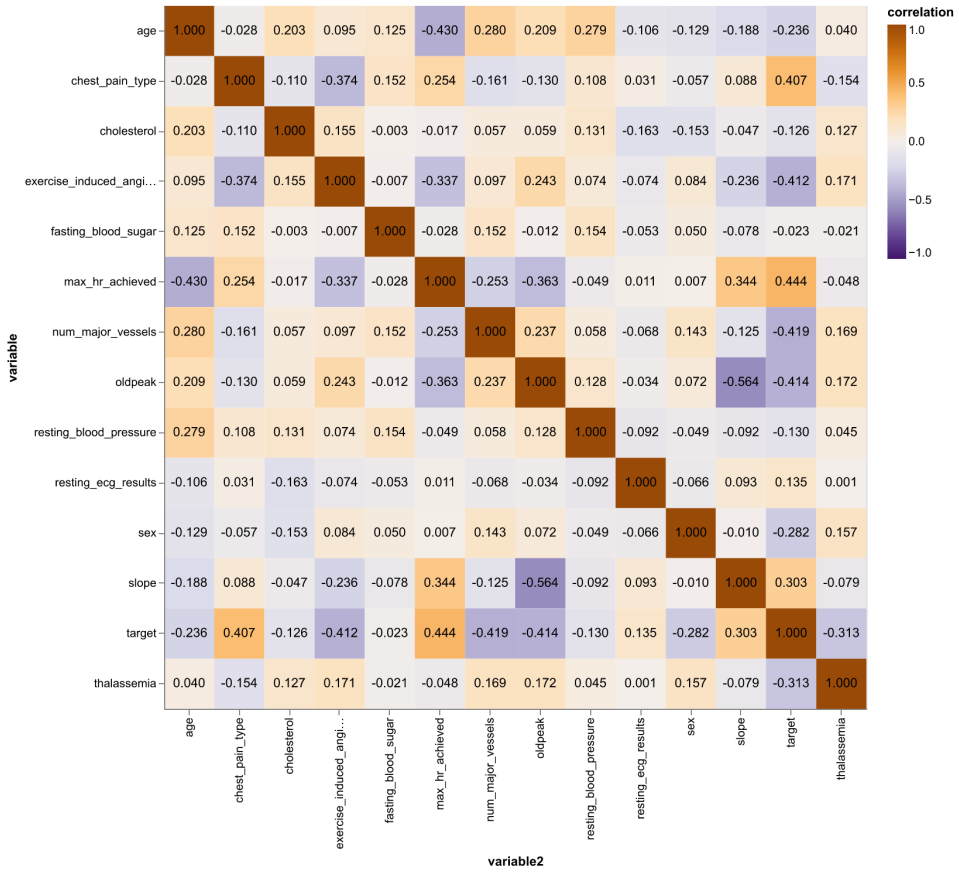
During the EDA stage, we first created histograms for the numeric features and bar plots for the categorical features, all of them were colored by the target.



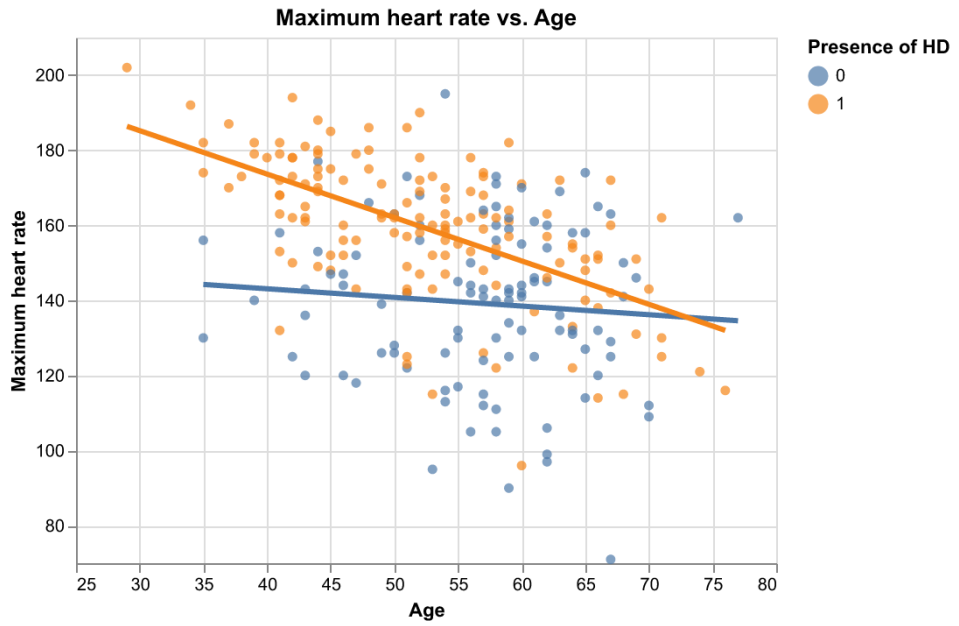


The two plots above demonstrated the varying ranges of the feautres and it helps to highlight what sort of preprocessing will be needed in order to incorporate the features into our predictive model.

Next, we created a pairwise correlation matrix for all the features and also the target to see if any particular feature might be more useful when predicting the target class.



It seems that there are some moderate correlations between some pairs of features. Also, `max_hr_achieved` has the highest correlation with target ($\rho = 0.44$) while `max_hr_achieved` have a moderate negative correlation with `age` ($\rho = -0.41$). We plot `max_hr_achieved` against `age` and colored the plot by age; in addition, we fit two simple linear regression line to see if the distinction of target class is large enough just by using these two features.



For our project, we used Python as the language of our analysis pipeline [Perez *et al.*, 2011]. In the EDA stage, the plots and tables are created using Altair [VanderPlas *et al.*, 2018] and Pandas [Wes McKinney, 2010]. Machine learning models were built using the scikit-learn library [Pedregosa *et al.*, 2011] and Scipy is also used to generate hyperparameter distributions [Virtanen *et al.*, 2020]. Other python libraries used for our project also include docopt [de Jonge, 2020], Vega-Lite [Satyanarayan *et al.*, 2017] and Numpy [Harris *et al.*, 2020].

Output, Results and Discussions

Model Selection

In order to predict heart disease in patients based on the different health indicators provided in the dataset, we decided to try two different models, `LogisticRegression` and `SVC` (support vector classifier), with default parameters as well as the `DummyClassifier` as a base model for comparison. We conducted 5-fold cross validation on the train set and extracted the mean fit time, score time, test score and train score for each model as well as the standard deviations to compare the models and select one for hyperparameter optimization. We used the f1 score as opposed to accuracy as our scoring metric due to the class imbalance we observed during our EDA. The results are listed in the table below:

```
import pandas as pd
```

```
pd.read_csv("../../results/model_selection_results.csv", index_col=0, header = 0,
names=["Dummy: Mean", "Dummy: Std", "SVC: Mean", "SVC: Std", "LogisticRegression:
Mean", "LogisticRegression: Std"])
```

	Dummy: Mean	Dummy: Std	SVC: Mean	SVC: Std	LogisticRegression: Mean	LogisticRegression: Std
fit_time	0.002	0.001	0.009	0.001	0.012	0.001
score_time	0.002	0.001	0.006	0.001	0.005	0.000
test_score	0.726	0.007	0.870	0.038	0.836	0.012
train_score	0.726	0.002	0.908	0.003	0.885	0.008

Table 1: Cross validation and Train mean f1 scores and standard deviation by model

Based on these results, we could see that the `DummyClassifier` model already performs quite well at predicting the presence of disease. However, both `SVC` and `LogisticRegession` had higher mean cross validation f1 scores and were already performing better than the `DummyClassifier` with default values. `SVC` has the higher mean cross validation f1 score compared to `LogisticRegression` so we decided to continue with this model for downstream hyperparameter tuning.

Hyperparameter Optimization

To conduct hyperparameter optimization, we decided to use `RandomizedSearchCV` with 50 iterations to search through optimal values for `C` and `gamma` for the `SVC` model. We used the loguniform distribtuion from 10^{-3} to 10^{3} for both hyperparameters. We also looked at f1, recall, and precision scores to compare the models. The mean cross validation f1 scores from the top 5 results of this search are listed below:

```
optim_df = pd.read_csv("../../results/optimization_results.csv", index_col=0)
optim_df = optim_df.iloc[:,0:5].T
optim_df.iloc[:,0:8]
```

	mean_fit_time	mean_score_time	param_svc__C	param_svc__gamma	mean_test_f1	std_test_f1	mean_train_f1	std_train_f1
1	0.021412	0.023231	5.542653	0.004940	0.870002	0.036278	0.885719	0.008873
2	0.020799	0.021039	2.191347	0.008990	0.864217	0.036591	0.879164	0.005463
3	0.029400	0.020799	0.358907	0.074742	0.857567	0.037695	0.894765	0.004161
4	0.029684	0.022001	0.768407	0.225271	0.857033	0.033634	0.951647	0.006082
5	0.023398	0.019897	113.254281	0.003156	0.854928	0.025702	0.898059	0.006965

Table 2: Cross validation and Train mean f1 scores and standard deviation of top 5 models

After hyperparameter optimization, we did not get an f1 score higher than the default `SVC` model, but we can see that there is a range in the difference between the validation scores and train scores even when just looking at the top 5 models. This data can help us ensure our model is neither underfit nor overfit and performs as well as possible without being overly complex. The recall and precision scores of the top 5 models are listed below:

```
optim_df = optim_df.drop(["rank_test_recall",'rank_test_precision'], axis = 1)
optim_df.iloc[:,8:]
```

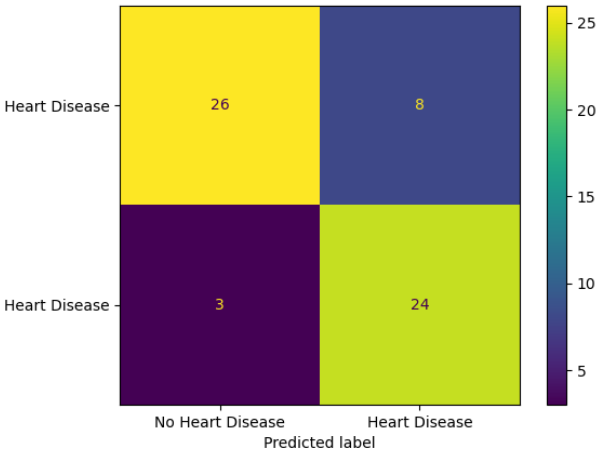
	mean_test_recall	std_test_recall	mean_train_recall	std_train_recall	mean_test_precision	std_test_precision	mean_train_precision	std_tra
1	0.941799	0.043671	0.954709	0.011475	0.809546	0.043052	0.826077	
2	0.941799	0.043671	0.949271	0.012293	0.799584	0.043485	0.818798	
3	0.927513	0.022625	0.954709	0.008097	0.799002	0.057458	0.842102	
4	0.905820	0.017576	0.980066	0.006777	0.813958	0.048279	0.924971	
5	0.898148	0.043807	0.942031	0.015798	0.817520	0.032966	0.858261	

Table 3: Cross validation and Train mean recall and precision scores and standard deviation of top 5 models

We can see that our recall scores are higher than our precision scores with this model. Since our problem is related to disease detection, we are more concerned with keeping recall high as we are care most about reducing false negatives. A false negative in this case will involve predicting no heart disease when heart disease is indeed present which is quite dangerous.

Test Results

After we had a model we were reasonably confident in, we assessed it using our test data. We achieved a f1 score of about 0.81 which is consistent with our cross-validation results and means we can be relatively confident in our model's performance to predict heart disease on deployment data. Below we have included the confusion matrix based on our test data to help visualize its performance:



Conclusions

In conclusion, the `SVC` model seems to be a good candidate for this heart disease prediction task. The gap between cross-validation scores and test scores was only about 6% so we are hopeful this model will be effective on deployment data as well. However, as our dataset was relatively small, it would be interesting to see how this model would scale up and whether the limited data used in both training and testing would impact the results.

Limitations & Future Work

In future, it would be good to try out a wider variety of models such as `RandomForestClassifier` or `linearSVC` and conduct hyperparameter optimization on multiple models before making a decision on which model to select. It would also be interesting to use `LogisticRegression` to look at feature importances and see if we can simplify our model by removing features with less relevance while still achieving similar results. Having a larger dataset to work with during training may also improve our model and improve its performance in deployment.

Bibliography

DJS+89 Robert Detrano, Andras Janosi, Walter Steinbrunn, Matthias Pfisterer, Johann-Jakob Schmid, Sarbjit Sandhu, Kern H Guppy, Stella Lee, and Victor Froelicher. International application of a new probability algorithm for the diagnosis of coronary artery disease. *The American journal of cardiology*, 64(5):304–310, 1989.

DG17 Dheeru Dua and Casey Graff. UCI machine learning repository. 2017. URL: http://archive.ics.uci.edu/ml.

HMvdW+20 Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. Array programming with NumPy. *Nature*, 585(7825):357–362, September 2020. URL: https://doi.org/10.1038/s41586-020-2649-2, doi:10.1038/s41586-020-2649-2.

JSPD88 Andras Janosi, William Steinbrunn, Matthias Pfisterer, and Robert Detrano. Heart Disease. UCI Machine Learning Repository, 1988.

PVG+11 F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

PGH11 Fernando Perez, Brian E Granger, and John D Hunter. Python: an ecosystem for scientific computing. *Computing in Science ||& Engineering*, 13(2):13–21, 2011.

SMWH17

Arvind Satyanarayan, Dominik Moritz, Kanit Wongsuphasawat, and Jeffrey Heer. Vega-lite: a grammar of interactive graphics. *IEEE transactions on visualization and computer graphics*, 23(1):341–350, 2017.

VGH+18 Jacob VanderPlas, Brian Granger, Jeffrey Heer, Dominik Moritz, Kanit Wongsuphasawat, Arvind Satyanarayan, Eitan Lees, Ilia Timofeev, Ben Welsh, and Scott Sievert. Altair: interactive statistical visualizations for python. *Journal of Open Source Software*, 3(32):1057, 2018. URL: https://doi.org/10.21105/joss.01057, doi:10.21105/joss.01057.

VGO+20 Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, C J Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R. Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1.0 Contributors. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272, 2020. doi:10.1038/s41592-019-0686-2.

deJonge20 Edwin de Jonge. *docopt: Command-Line Interface Specification Language*. 2020. R package version 0.7.1. URL: https://CRAN.R-project.org/package=docopt.

WesMcKinney10 Wes McKinney. Data Structures for Statistical Computing in Python. In Stéfan van der Walt and Jarrod Millman, editors, *Proceedings of the 9th Python in Science Conference*, 56 – 61. 2010. doi:10.25080/Majora-92bf1922-00a.