**Predicting Heart Disease Using Machine Learning Classification Models**

Isaac McPadden

University of Denver

COMP 4448 Data Science Tools 2

Dr. Neba Nfonsang

# Predicting Heart Disease Using Machine Learning Classification Models

According to the US Centers for Disease Control and Prevention (CDC), heart disease is the leading cause of death in the United States. Despite this, it is rarely diagnosed until after a person experiences life threatening symptoms such as a heart attack or arrythmia. Each year, the CDC’s Behavioral Risk Factor Surveillance System (BRFSS) conducts a survey of Americans about various health behaviors and outcomes. With recent improvements in computer processing, it may be possible to use that data to identify people with a high risk of heart disease before they experience dangerous symptoms.

## Purpose

The purpose of this project is to develop and compare different classification models using different machine learning algorithms to determine if any computer models can predict if a person has heart disease. The machine learning classification algorithms will be stochastic gradient descent (SGD), random forest (RF), k-nearest-neighbors (KNN), linear support vector classification (LSVC), and general support vector classification (SVC). SGDs, LSVCs, and other SVCs in general create separations between data points that maximize the classification score. SGDs generalize better and are more efficient, particularly for large datasets, but if SVCs are appropriate for the classification problem, they will outperform SGDs. Random Forest generates many decision trees (if condition 1 and condition 2 and…, then outcome) and finds the best for the classification problem. KNN is a clustering algorithm that finds data points “nearest” a data point of interest and compares the classification of the neighbors to determine the classification of the point of interest.

## Significance

Heart disease is a leading cause of death in the US and often goes unnoticed until a person suffers a heart attack. If it is possible to predict if a person has heart disease based on lifestyle factors, then medical intervention is possible before a heart attack, potentially saving thousands of lives annually and millions of dollars in medical costs.

## Research Question

The research question I am trying to answer is: Can machine learning classification algorithms be used to predict if a person has heart disease, and if so, which classification algorithm is best, SGD, RF, KNN, LSVC, or SVC?

## Description of the Dataset

The dataset I am using is a subset of the CDC’s 2015 BRFSS survey. Kaggle user Alex Teboul selected 21 variables relevant to heart disease and a target variable of whether a person had been diagnosed with heart disease or had a heart attack or not. He cleaned the data and modified it to be either binary or ordinal (categories represented as numbers). The variables fall into three categories: health metrics such as BMI, self-reported health behaviors such as a general health rating, and demographics such as age and income.

# Data Preprocessing

## Data Preparation

After confirming the dataset was cleaned and formatted, I imported the data into a dataframe and min-max scaled the data so every column had a value between 0 and 1. Many of the values were ordinal, not continuous, so standard scaling was not a good option. I also made a subframe with fewer columns that I hypothesized to be most important to a heart disease diagnosis in case the original dataframe was too large.

## Exploratory Data Analysis and Visualization

Exploratory Data Analysis (EDA) was performed using Y-Data’s Pandas Profiling Python library. Most of the variables are binary and unbalanced. Age and BMI have a gaussian distribution and education and income are both skewed right. The target variable, HeartDiseaseorAttack, is especially unbalanced, with ninety percent of the values being 0, and ten percent being 1. This is problematic because a model could guess 0 for every single person and still score ninety percent accurate. Despite this, I did train some models on the unbalanced data because it is likely representative of heart disease in the general population, and I wanted to test the importance of balance in the target variable.

Correlation between variables was mostly weak with two exceptions as can be seen in the heatmap below.

A picture containing text, screenshot, line, font

Description automatically generated

Education and income are unsurprisingly positively correlated with each other. Difficulty walking and general health were also positively correlated. This is because the survey chose 1 as good and 5 as bad for general health and 0 as no difficulty walking and 1 for difficulty walking. Thus, some variables are inverted but modeling should correct for that. The weak correlation of variables suggests minimal redundancy in the data. There were many rows of identical data, however that is expected because the data have over 200,000 records and only 22 columns, all of which have a very limited number of different possible values.

## Data Splitting

The data were split into training and testing data sets with a ratio of 0.7 to 0.3. Initial model runs were made using the full dataframe. After that, I chose a subset of variables that seemed likely to be most important based on my limited knowledge of heart disease. There were 8 variables in that feature set. Next, I used scikit learn’s recursive feature elimination function to statistically choose the 8 best variables. Then, I selected all records with a HeartDiseaseorAttack value of 1 and randomly selected and equal number of records with a HeartDiseaseorAttack value of 0 to create a dataframe with a balanced target variable. Last, I did another recursive feature elimination down to 5 variables to determine if further simplification of the model had better results.

# Model Building and Evaluation

## Model Building

Initial model building consisted of the five classifier algorithms from scikit learn scoring for accuracy and using every feature in the dataset. While accuracy scores were high between 90 and 91 percent for all models, the F1 score was below 20 percent for all models. With major health conditions such as heart disease, the F1 score is more important because it punishes false negatives and false positives. A false negative in this situation would cause a patient to miss out on critical healthcare. A false positive in this situation could result in unnecessary medication or invasive surgery. Thus, I rebuilt the models using the F1 score as that scoring metric and continued to use F1 score as the scoring metric for the remainder of the project.

Each model consists of the classifier, a parameter grid, a grid searcher, and the model fitter. For example, the KNN model is built as follows:

# KNN  
knn = KNeighborsClassifier()  
param\_grid\_knn = {'n\_neighbors': [3, 5, 7,11,19], 'weights':['uniform', 'distance']}  
grid\_knn = GridSearchCV(knn, param\_grid\_knn, cv=8, n\_jobs = 6, refit = True, scoring='f1')  
grid\_knn.fit(X\_train, y\_train)

When the grid search is completed, a printout of the best grid parameters and the training metrics is made and the model is used with the test data to predict if each record has been diagnosed with heart disease.

After that I fed the predictions into a test scoring function that prints out the scores for the test data.

def scores\_func(estimator, ytest, ypred):  
 print(f'{estimator} scores: ')  
 cr = classification\_report(ytest, ypred)  
 print(cr)  
 cm = confusion\_matrix(ytest, ypred)  
 ps = precision\_score(ytest, ypred)  
 rs = recall\_score(ytest, ypred)  
 acc = accuracy\_score(ytest, ypred)  
 f1s = f1\_score(ytest, ypred)  
 print(f'Accuracy: {acc}')  
 print(f'F1 Score: {f1s}')  
 print(f'Precision: {ps}')  
 print(f'Recall: {rs}')  
 disp = ConfusionMatrixDisplay.from\_predictions(ytest, ypred)

Continuing with the KNN example:

# K-Nearest Neighbors Scores  
scores\_func('K-Nearest Neighbors', y\_test, y\_pred\_knn)

A screenshot of a computer

Description automatically generated

Note that this output is from later once the model was tuned.

A picture containing text, screenshot, colorfulness, square

Description automatically generated

## Model Optimization And Model Selection

The parameter grids chosen for each model were somewhat limited due to computing power and the time cost associated with adding terms to the grids. They were largely made based off examples found online as a starting point and modified if the hyperparameter properties gave me reason to believe it needed more or fewer options. The parameter grids for each model are listed below.

param\_grid\_sgd = {'loss': ['hinge', 'log\_loss'], 'penalty': ['l1'], 'alpha': [0.0001, 0.001, 0.01, 0.1], 'learning\_rate': ['constant', 'optimal', 'invscaling', 'adaptive']}

param\_grid\_rf = {'n\_estimators': [50, 100, 200], 'max\_depth': [None, 3, 5, 10, 20], 'min\_samples\_split': [2, 5, 10]}

param\_grid\_knn = {'n\_neighbors': [3, 5, 7,11,19], 'weights':['uniform', 'distance']}

param\_grid\_svc = {'C': [0.1, 1, 10, 100, 1000], 'max\_iter': [2500]}

param\_grid\_svc2 = {'C': [0.1, 1, 10, 100], 'gamma': [1, 0.1, 0.01, 0.001], 'kernel': ['rbf'], 'max\_iter': [2500]}

Initial F1 scores for the models were not good, even with parameter optimization, so I started modifying the data fed into the models. As stated in the section on Data Splitting, I started by reducing the number of features supplied to the model, first by hypothesizing which variables might be important, and then by recursive feature elimination to get the same number of features but with statistically supported choices. The F1 scores showed modest improvement but only the general SVC model had an F1 score over 30 percent. It was clear that the imbalanced target variable was interfering with the result so I made a balanced subset of the initial data with the same 8 RFE features. Results greatly improved with all models having F1 scores over 70 percent. I attempted to improve the models by simplifying the data and providing only 5 RFE features but the only one that had a better F1 score was the LSVC model. The best performing model was the RF with 8 RFE features and balanced target with an F1 score of 77.53 percent.

## Model Comparison

F1 Scores



A screenshot of a graph

Description automatically generated with medium confidence

Accuracy Scores



A screenshot of a computer program

Description automatically generated with low confidence

# Conclusion

The best classification model for this problem was the random forest classifier supplied with balanced target data and 8 features chosen by recursive feature elimination. The optimal parameters are a max depth of 10, a minimal samples split of 10, and an n estimators value of 50. The resulting model had an F1 score of 77.53 percent and an accuracy of 76.49 percent, both of which were the best scores for models with balanced data.

## Lessons Learned

* Having balanced data supplied to a classification model is more important than having a target variable representative of the general population.
* RFE can greatly improve the performance and speed of classification models.
* Iterative models need a maximum number of iterations or the model will never finish training in some cases.
* Human health is more complicated than a survey can account for.

## Recommendations

Unfortunately, 77.53 percent is not good enough to replace a proper visit to a doctor to assess risk of heart disease. It is possible that this model could be improved by doing a more comprehensive check of RFE variable counts or by using a deep learning algorithm. That said, including a risk assessment from a model could prove to be a useful data point for a doctor considering their patient’s risk of heart disease.

# References

* <https://www.cdc.gov/heartdisease/>
* <https://www.kaggle.com/datasets/alexteboul/heart-disease-health-indicators-dataset>
* <https://www.kaggle.com/code/alexteboul/heart-disease-health-indicators-dataset-notebook/notebook>
* <https://scikit-learn.org/>

# Appendix: Figures of Training Scores and Test Accuracy

A picture containing text, screenshot, rectangle, parallel

Description automatically generated

A screenshot of a graph

Description automatically generated with low confidence

A picture containing text, screenshot, parallel, rectangle

Description automatically generated