Machine Learning Engineer Nanodegree

Capstone Project Report
Urvi Patel Yi
December 2018

Definition

Project Overview

“Every year, over 100,000 people die from a heart attack or related stroke in the UK alone, and heart disease and stroke remain the two biggest overall causes of death worldwide.” This was quoted in an August 2018 article published in MedExpress, a medical blog published by the University of Oxford. (https://medicalxpress.com/news/2018-08-technology-fatal-heart.html) The focus of the article was on how new technologies may be used to predict heart attacks in patients. Of course, it is impossible to talk about heart attacks, without also talking about heart disease.

Heart disease is a condition in which plaque builds up on the walls of arteries. This can potentially cause blood clots that restrict the flow of blood, which in turn can cause heart attacks. (Source: http://www.heart.org/en/health-topics/consumer-healthcare/what-is-cardiovascular-disease). Preventing heart disease is, therefore, a key factor in preventing heart attacks.

I am personally invested in investigating instances of heart disease because my father passed away from his first heart attack when I was seventeen years old. He had not been diagnosed with heart disease at the time, which meant that we were completely unaware of the risk when the heart attack struck. I believe a prediction system for heart disease will enable patients to better understand their risk for heart attacks.

Problem Statement

I will attempt to predict an instance of heart disease given a set of health measurements from a patient. The health measurements will be the features that will be the input into the machine learning model. The target will be to predict the presence of heart disease or the absence of heart disease from the patient. This will be treated as a binary classification problem.

Evaluation Metrics

The UCI Heart Disease dataset is a balanced dataset: Exactly 50% of the patients were diagnosed with heart disease, and exactly 50% of the patients were diagnosed healthy.

Even though this is a balanced data set, I will use an F-Beta score to evaluate this model, with Beta = 1.5.

I chose Beta = 1.5 because I want this to be a high recall model, meaning that false positives are preferred over false negatives. This means that it is acceptable if a patient is falsely diagnosed with heart disease because they can be referred to other tests to gain confidence in the result. It is unacceptable if a patient with heart disease is misdiagnosed with a false negative--because then the disease will go unchecked and may result in death.

This is a Type 2 error, where unhealthy patients are misdiagnosed as healthy, and it should be avoided. Because the F-Beta score takes into account both precision and recall, I will use this as an evaluation metric instead of the accuracy score.

Analysis

Data Exploration

I used the Heart Disease Datasets from the UCI Machine Learning Repository to train and test my machine learning models. (Source: https://archive.ics.uci.edu/ml/datasets/heart+Disease)

The original datasets contain patient data from three sources:

1. Cleveland Clinic Foundation
2. Hungarian Institute of Cardiology, Budapest
3. University Hospital, Zurich, Switzerland

Cleveland UCI Heart Disease Dataset contains 303 samples with 14 features each.
Hungarian UCI Heart Disease Dataset contains 294 samples with 14 features each.
Swiss UCI Heart Disease Dataset contains 123 samples with 14 features each.

The Cleveland dataset has been most widely used by Machine Learning researchers.

Originally, 76 different attributes were collected from each patient, but the datasets only include 14 of these attributes.

1. **age**: age in years
2. **sex**: sex (1 = male; 0 = female)
3. **cp**: chest pain type
    - 1 = typical angina
    - 2 = atypical angina
    - 3 = non-anginal pain
    - 4 = asymptomatic
4. **trestbps**: resting blood pressure (in mm Hg on admission to the hospital)
5. **chol**: serum cholesterol in mg/dl
6. **fbs**: (fasting blood sugar > 120 mg/dl) (1 = true; 0 = false)
7. **restecg**: resting electrocardiographic results
    - 0 = normal

- 1 = having ST-T wave abnormality (T wave inversions and/or ST elevation or depression of > 0.05 mV)
- 2 = showing probable or definite left ventricular hypertrophy by Estes' criteria

8. **thalach**: maximum heart rate achieved
9. **exang**: exercise induced angina (1 = yes; 0 = no)
10. **oldpeak**: ST depression induced by exercise relative to rest
11. **slope**: the slope of the peak exercise ST segment
    - 1 = upsloping
    - 2 = flat
    - 3 = downsloping
12. **ca**: number of major vessels (0-3) colored by flourosopy
13. **thal**: 3 = normal; 6 = fixed defect; 7 = reversable defect
14. **num** / **goal**: diagnosis of heart disease (angiographic disease status)
    - 0 = < 50% diameter narrowing
    - 1 = 50% diameter narrowing

    (in any major vessel: attributes 59 through 68 are vessels)

The first 13 attributes will be the features into my machine learning model. The 14th attribute (num/goal, renamed to **heartdiease**) will be my target variable.

One interesting note: The "**num**" or "**goal**" field is the diagnosis of heart disease in the patient. In the Hungarian and Swiss datasets, this attribute is integer valued from 0 (no presence) to 4. In the Cleveland dataset, this is simply 0 (no presence of heart disease) or 1 (presence of heart disease). The Cleveland dataset contains almost half of all data available. Because of this, for the Hungarian and Swiss datasets, I converted all nonzero values of "**goal**" to a value of 1 for my analysis.

In the raw combined dataset, the distribution of the goal values is as follows:

Goal	Count
0	360
1	209

2	68
3	65
4	18

As I suspected, this dataset is imbalanced because the severity of heart disease (values 2 - 4) were not distinguished when the Cleveland data was collected.

I converted all values between 2 - 4 to a value of 1 to indicate the presence of heart disease. A value of "0" indicated the absence of heart disease.

Heart Disease	Count
0	360
1	360

After the conversion, the dataset appears exactly balanced.

To compare this to the real world: I was unable to find an estimated percentage of the population that has heart disease. I did, however, find a statistic from the American Heart Association that states that an estimated 17.3% of the global population die from heart disease every year. (Source: https://www.heart.org/idc/groups/ahamah-public/@wcm/@sop/@smd/documents/downloadable/ucm_470704.pdf) This number may not account for people who have heart disease but have not died from it.

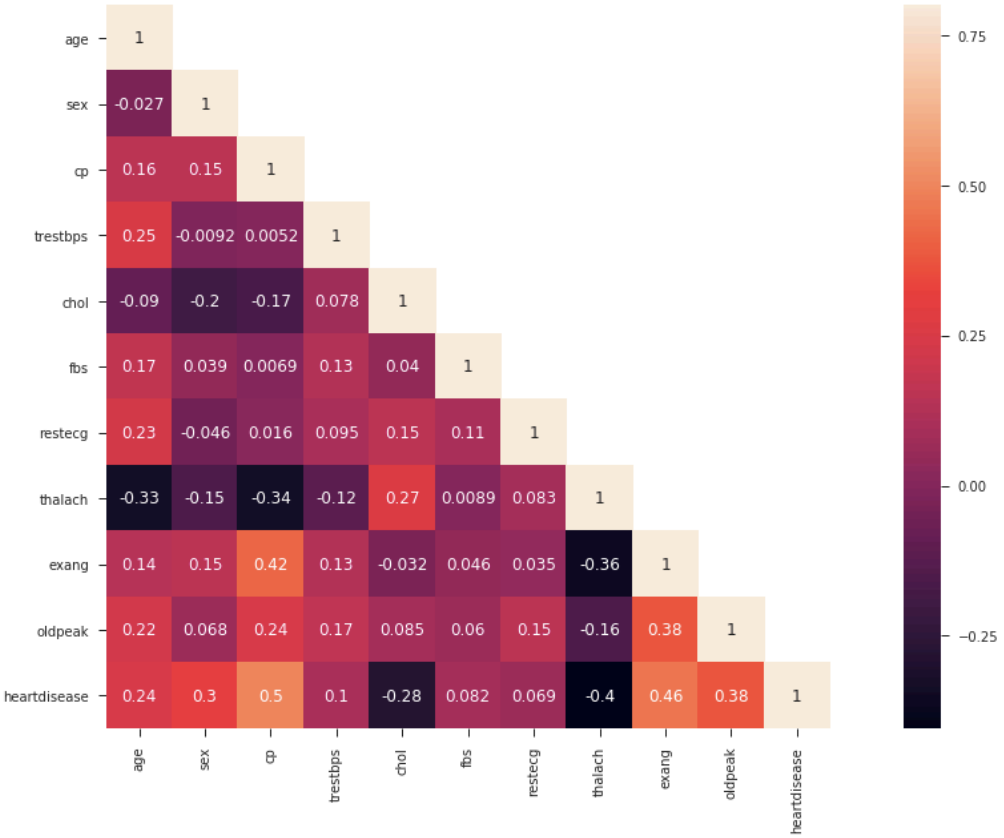
	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	heartdisease
count	720.000000	720.000000	720.000000	720	720.0	720	720	720	720	720.0	720	720	720	720.000000
unique	NaN	NaN	NaN	86	307.0	5	7	188	5	77.0	7	8	7	NaN
top	NaN	NaN	NaN	120	0.0	0	0	150	0	0.0	?	?	?	NaN
freq	NaN	NaN	NaN	78	123.0	309	320	32	272	288.0	207	413	320	NaN
mean	51.888889	0.738889	3.179167	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	0.850000
std	9.193720	0.439546	0.953646	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	1.076878
min	28.000000	0.000000	1.000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	0.000000
25%	45.000000	0.000000	2.000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	0.000000
50%	53.000000	1.000000	4.000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	0.500000
75%	58.000000	1.000000	4.000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	1.000000
max	77.000000	1.000000	4.000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	4.000000

Another interesting note about the data is that some of the columns have a lot of missing data. For example, in data statistics below, the **slope**, **ca**, and **thal** columns show that the most frequent (top) value is "?". These data columns were removed during the data preprocessing.

Exploratory Visualization

I used various methods to visually inspect the data. The most useful visualization I found was the correlation matrix. The correlation visualization below shows each feature in the dataset and the correlation coefficients associated with each other feature in the dataset, including the target, **heartdisease**.

From the visual inspection of the heat map, I determined that there is no close correlation between the **heartdisease** target and the other features in the dataset because all features show a correlation coefficient < 0.75 in the **heartdisease** row. I also did not see any high correlation between other features in the data set.



Algorithms and Techniques

I used a variety of supervised learning algorithms to classify the data. This is a binary classification problem: 0 indicates the absence of heart disease, and 1 indicates the presence of heart disease in the patient. Supervised learning algorithms are appropriate in this case because the dataset contains a column with values for the target (**heartdisease**) I will attempt to predict.

I worked on this project entirely in Python 3.6 using Jupyter Notebooks. I used various python libraries throughout this project, including pandas for data exploration, seaborn for data visualization, scikit-learn for building an SVM model, and keras for building a deep neural network model.

After downloading the data from the UCI Machine Learning Repository, I explored the data and performed some data cleanup activities. For example, I dropped features that contained a lot of missing values (**slope**, **ca**, and **thal**). I filled in any remaining missing values with the mean of the feature, and finally converted all values to a numerical format.

Next, I chose a model evaluation metric and fine-tuned it. I chose to use an F-Beta score because I wanted the scores to be biased toward recall. I then built a Naive Predictor model to calculate the accuracy and F-Beta (Beta = 1.5) scores. I used the Naive Predictor model to help fine tune the Beta value for the F-Beta score.

As a final step to processing the data, I split the data into a training and testing set for the rest of the analysis. The testing set was used to evaluate the model and understand if my model is overfitting to the dataset.

After performing all data exploration, visualization, and cleanup steps and after choosing my evaluation metric, I finally built two supervised learning models and tried to predict the presence of heart disease in a patient. The two models I chose were: (1) a Benchmark Model

(an SVM model based on some published results on this dataset) and (2) a deep neural network I built and fine-tuned using keras.

I fit the data to both the Benchmark Model (SVM) and the deep neural network, and I calculated the F-Beta scores for each approach. I used this F-Beta score to evaluation the performance of each model and compare the models against each other.

Benchmark Model

Previously, the UCI Heart Disease datasets have been explored using an SVM to an accuracy of 64%. This model is presented in the Packt Publishing's Real World Machine Learning Projects with Scikit Learn Workshop (Source: https://github.com/PacktPublishing/Real-World-Machine-Learning-Projects-with-Scikit-Learn/blob/master/Section%203%20-%20Code.zip)

For the Packt SVM model, the **heartdisease** target values were not converted to values 0 or 1. I used the ideas from the Packt SVM model to build my own SVM model that trained on the dataset that I pre-processed. I used this as my benchmark model. The benchmark model gave me an F-Beta (Beta = 1.5) score of 82%. This is the score I attempted to improve upon with the deep neural network.

Methodology

Data Preprocessing

In the data preprocessing step, I first converted the values in the **heartdisease** target values to 0 or 1. Any value greater than 1 was converted to a value of 1. A value of 0 indicates the absence of heart disease, and a value of 1 indicates the presence of heart disease.

As mentioned previously, I converted these values because the Cleveland dataset (which composes almost half of all available data) did not distinguish between values 1-4. This makes the dataset biased toward a value of 1.

Before converting the **heartdisease** values, the distribution of values was the following:

Heartdisease	Count
0	360
1	209
2	68
3	65
4	18

After converting the values, the distribution of **heartdisease** values was exactly balanced:

Heart Disease	Count
0	360
1	360

Next, I examined a description of the data using **data.describe()**. I saw that some features (**slope**, **ca**, and **thal**) had a top value of "?". The top value is the most frequent value in the

column. This means that the **slope**, **ca**, and **thal** columns had mostly missing data. Because of this, I dropped these features from my data set.

Next, I replaced all "?" (a string) with **null** values. I further replaced all the **null** values with the mean of each feature. After looking at a description of the dataset again, I realized that many of the columns were still in a string format. I used **data = data.apply(pd.to_numeric)** to convert all of the remaining data to numerical data.

After converting all the remaining data to numerical values, I was able to see a clearer statistical picture of the data, as shown below.

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	heartdisease
count	720.000000	720.000000	720.000000	717.000000	697.000000	637.000000	718.000000	718.000000	718.000000	714.000000	720.000000
mean	51.888889	0.738889	3.179167	131.804742	204.774749	0.109890	0.568245	140.565460	0.337047	0.789636	0.500000
std	9.193720	0.439546	0.953646	18.529331	109.275156	0.312999	0.833534	25.647172	0.473031	1.066961	0.500348
min	28.000000	0.000000	1.000000	80.000000	0.000000	0.000000	0.000000	60.000000	0.000000	-2.600000	0.000000
25%	45.000000	0.000000	2.000000	120.000000	182.000000	0.000000	0.000000	122.000000	0.000000	0.000000	0.000000
50%	53.000000	1.000000	4.000000	130.000000	227.000000	0.000000	0.000000	142.000000	0.000000	0.200000	0.500000
75%	58.000000	1.000000	4.000000	140.000000	270.000000	0.000000	1.000000	160.000000	1.000000	1.500000	1.000000
max	77.000000	1.000000	4.000000	200.000000	603.000000	1.000000	2.000000	202.000000	1.000000	6.200000	1.000000

Implementation

Naive Predictor Model and F-Beta ScoreS

Even though this is a balanced dataset, I used an F-Beta score with Beta = 1.5 as my evaluation metric. I chose this metric because this is a medical diagnosis classification problem. I wanted the score I choose to be biased toward high recall because it is more acceptable to misdiagnose healthy patients as sick in this case than it is to misdiagnose sick patients as healthy.

I used a Naive Predictor model to fine tune the Beta value I wanted to use. In the Naive Predictor, I predicted that all 720 patients in the data set have heart disease. I then calculated Accuracy and F-Beta scores on this Naive Predictor model.

Because 50% of all patients in this dataset were diagnosed to have heart disease and 50% were actually healthy, I got an accuracy score of only 50%, as expected, because only half of the predictions were correct.

I then calculated the F-Beta score using two values of Beta: 1.5 and 2.0.

Originally, I had planned to use an F2 score to evaluate the models. However, when I calculated the F-2 score on the Naive Predictor model, I got a score of 83%. I determined that this score was too biased toward recall. I decided to lower the Beta value so that the metric is a little closer to precision (but still high recall).

I adjusted the Beta to Beta = 1.5 and calculated the F-Beta (Beta = 1.5) score on the Naive Predictor model. This time, I got a score of 76%. I thought this was still fairly high and biased toward recall, but I decided that this is acceptable due to the severity of the diagnosis.

Benchmark Model

As discussed previously, I chose to recreate the Benchmark model previously published by Packt Publishing, which had gotten an accuracy score of 64% on minimally preprocessed data.

I built the SVM using scikit-learn with an **rbf** kernal. I then fit the model to my preprocessed data, which had been split into training and testing sets using **train_test_split()**.

The F-Beta (Beta 1.5) score for this model fitted to this dataset was 82%. This is an improvement compared to the Naive Predictor case.

Deep Neural Network

I chose a Sequential deep neural network in keras to further analyze the data and see if I could improve on the result from the Benchmark model. Again, I calculated the F-Beta (Beta = 1.5) score for this model. Keras does not include a metric for F-Beta scores, so I modified a script that I found on Kaggle to do the calculations for me. (The only modification I made was to change the Beta from 2 to 1.5.) This script calculated F-Beta score for each epoch during training.

I tried different numbers of layers, various activation functions in the hidden layers, and different optimizers to. I also experimented with the number of dropout layers as well as the dropout rate. Many of these architecture changes gave me F-Beta scores of less than 76%, even after training for over 500 epochs. Once, I added a dropout layer in between each hidden layer, believing that this would help reduce overfitting to the training set, but the F-Beta on this model plummeted to below 50%.

After rapidly experimenting with various parameters, I finally chose the following architecture, which initially got me close to an F-Beta score of 83%:

- Input Layer with input dimension 10 (for each feature), using a **relu** Activation Function
- 2 Hidden Layers, with output sizes doubling from 32 to 64 using **tanh** activation functions
- 1 Dropout Layer with a dropout rate of 25% in between the two hidden layers
- Output layer with **sigmoid** Activation function
- I used an **adam** optimizer and trained the model for 500 epochs

Refinement

I used the following methodology to determine how many epochs to train the model:

1. First I tried training the neural network with 1000 epochs. I discovered that this was too many epochs because I got a training accuracy of 91%, but a testing accuracy of 82%. This indicated that I was overfitting the data to the training set.

2. Then I tried training the neural network with 50 epochs. I discovered that this was too low because I got Training Accuracy of 75%, and a testing accuracy of 79%. This indicates the model underfit the data.

3. I wanted to see if I could improve the results, so I increased the epochs by 50 epochs for a few runs and compared the training and testing accuracy scores. I stopped at 500 epochs. The training accuracy was 84%, and the testing accuracy was 83%. If I tried increasing the epochs more than that, but the scores stayed mostly the same until the model began overfitting to the training set, which increased the score on the training set, but not the test set.

Results

Model Evaluation and Validation

I re-compiled and ran the neural network model several times, and I was able to consistently get F-Beta scores around 82% - 85% for the training and testing sets. This indicates to me that the model is sufficiently robust, even when the training set is shuffled differently. Also, adding a dropout layer with dropout rate 25% gives me confidence that this model is robust to changes in the input data.

Justification

The F-Beta score achieved with the neural network is performs just as well or slightly better than the SVM Benchmark Model. To summarize,

Benchmark Model F-Beta (Beta = 1.5) Score: 82%
Deep Neural Network Model F-Beta (Beta = 1.5): 82% - 84%

However, even though the Deep Neural Network performs just as well as the SVM Benchmark Model, the neural network took a significant amount of time more to build, fine-tune, and train. For these reasons, I believe the Benchmark model is actually the better model to use for this data set.

Conclusion

Free-Form Visualization

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_42 (Dense)             (None, 32)                352
_________________________________________________________________
dense_43 (Dense)             (None, 64)                2112
_________________________________________________________________
dropout_14 (Dropout)         (None, 64)                0
_________________________________________________________________
dense_44 (Dense)             (None, 32)                2080
_________________________________________________________________
dense_45 (Dense)             (None, 1)                 33
=================================================================
Total params: 4,577
Trainable params: 4,577
Non-trainable params: 0
_________________________________________________________________
```

This is a visualization of the final deep neural network I chose:

Note this this has only one dropout layer. (I experimented with adding a dropout layer in between each hidden layer, but my training score went down significantly when I made this change.)

My model contains the Input Layer, 2 Hidden Layers, 1 Dropout Layer, and and 1 Output Layer.

Reflection

In this project, I found real data from real patients on the occurrence of heart disease globally. I was attracted to exploring this data because heart disease (and subsequently, heart attacks and strokes) is a leading cause of death around the world and it affects me personally. This real world dataset contained a lot of missing values. If the feature contained too many missing

values, I removed it from the dataset. The remaining missing datapoint were filled with the mean for each feature.

After cleaning up the data set, I constructed a metric for my models. I chose an F-Beta score for my evaluation metric because I wanted to be able to score the model for high recall. I constructed a Naive Predictor model in which I predicted that all patients have heart disease. I used this model to fine tune the value of Beta for the F-Beta score. For my final metric, I chose an F-Beta score with a Beta = 1.5.

Then, split my whole dataset into training and test sets, which I fed both into a Benchmark Model (an SVM based on published results), and into a deep neural network I built and fine-tuned using keras. The F-Beta scores I got on the Benchmark and the Neural Network were similar (around 82%), but I concluded that the SVM is a better approach because it took less time to train.

The most difficult part of this project was fine tuning the neural network. It was very difficult to train the neural network so that the model got an F-Beta score above 75%. I think a big reason for this is that this dataset is very small, and deep neural networks generally perform better on larger data sets.

One of the most surprising aspects of this project was that I found it very surprising to see just how much I could improve the accuracy of my results by only pre-processing the data a little. For example, in the original version of this project, I decided to keep the heart disease values 0-4. I was getting accuracy values close to 65% using an SVM model. After noting the bias in the dataset, I converted the values in the heart disease column to binary 0 and 1. After that, my accuracy jumped up to 81%. (Note: At that stage in the project, I was doing some initial evaluation of the data set, and I had not settled on the F-Beta score for my evaluation metric.)

Overall, I found that it was interesting to explore such a small dataset with a deep neural network. However, I do not think this is a good approach to this problem, because the SVM Benchmark Model can predict the occurrences of heart disease just as well as the deep neural network, with the added benefit that there is much less fine-tuning involved.

Improvement

I believe, to truly improve the score from the deep neural network model, more data is needed. Real world data is preferable in this case because this is all numerical data based on real patients. I am not entirely sure how I would augment this dataset to include more data or to simulate data based on the existing set without destroying the validity of the dataset.

Another option I did not explore was to normalize the data set. This occurred to me as I was reviewing my project report, and it's possible this would have improved the F-Beta scores for both the Benchmark and the Neural Network models.

A neural network approach was not a good choice in this case because of how small the data set is. One of the Udacity reviewers also recommended that I look into XGBoost (https://machinelearningmastery.com/develop-first-xgboost-model-python-scikit-learn/) or LightGBM (https://github.com/Microsoft/LightGBM/blob/master/examples/python-guide/simple_example.py). These may be better algorithms to use on a small dataset, and I am interested in exploring them when I have the chance.