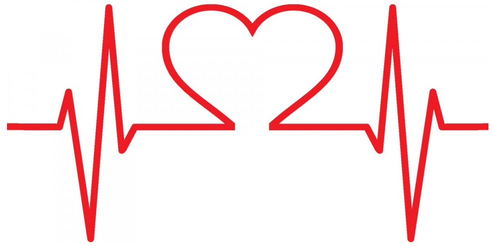
BLOG

MACHINE LEARNING WITH A HEART – BENCHMARK

MON 23 JULY 2018

So you want to harness the power of machine learning but need a place to start? We've got just the task for you: detecting heart disease! Heart disease is the number one cause of death worldwide(https://www.world-heart-federation.org/resources/cardiovascular-diseases-cvds-global-facts-figures/), so if you're looking to use data science for good you've come to the right place. To learn how to prevent heart disease we must first learn to reliably detect it. That's where you––yes you––come in!



To join the competition, follow this link(http://www.drivendata.org/competitions).

In our brand new warm up competition(https://www.drivendata.org/competitions/54/machine-learning-with-a-heart/page/107/) we're asking you to predict the presence or absence of heart disease given various data about a patient, including resting blood pressure, maximum heart rate, and EKG(https://www.mayoclinic.org/tests-procedures/ekg/about/pac-20384983) readings, as well as other information like age and sex. The data comes from the Statlog Heart dataset via the UCI Machine Learning repository(http://archive.ics.uci.edu/ml/datasets/statlog+(heart)). This is one of the smallest, least complex datasets on DrivenData, and a great place to dive into the world of data science competitions.

In this post, we'll walk through a very simple first pass model for predicting heart disease from patient data, showing you how to load the data, make some predictions, and then submit those predictions to the competition.

To get started, we import libraries for loading, manipulating, and visualizing the data.

In [1]:
```
%matplotlib inline

from pathlib import Path

import numpy as np
import pandas as pd

import matplotlib.pyplot as plt
import seaborn as sns
```

In [2]:
```
DATA_DIR = Path('..', 'data', 'final', 'public')
```

LOADING THE DATA

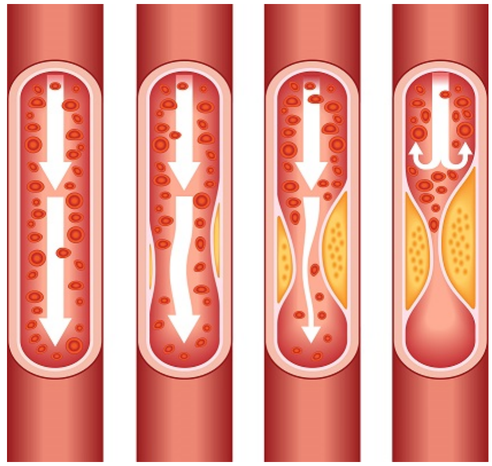


Image and quote from the [Centers for Disease Control and Prevention](https://www.cdc.gov/heartdisease/facts.htm): As plaque builds up in the arteries of a person with heart disease, the inside of the arteries begins to narrow, which lessens or blocks the flow of blood. Plaques can also rupture (break open) and when they do a blood clot can form on the plaque, blocking the flow of blood.

On the data download page(https://www.drivendata.org/competitions/54/machine-learning-with-a-heart/data/), we provide everything you need to get started:

- **Training Values:** These are the features you'll use to train a model. There are 13 features in data, including resting blood pressure, maximum heart rate, and EKG readings, as well as other information like age and sex. Each patient is identified by a unique (random) `patient_id`, which you can use as an index.
- **Training Labels:** These are the labels. Every `patient_id` in the training values data has a corresponding label in this file. A `0` indicates no heart disease present, whereas a `1` indicates the presence of heart disease.
- **Test Values:** These are the features you'll use to make predictions after training a model. We don't give you the labels for these samples, it's up to you to generate probabilities of the presence or ansence of heart disease for these `patient_id`s!
- **Submission Format:** This gives us the filenames and columns of our submission prediction, filled with all `0.5` as a baseline. Your submission to the leaderboard must be in this exact form (with different prediction values, of course) in order to be scored successfully!

Since this is a benchmark, we're only going to use a subset of the features in the dataset. It's up to you to take advantage of all the information!

In [3]:
```
# for training our model
train_values = pd.read_csv(DATA_DIR / 'train_valu
train_labels = pd.read_csv(DATA_DIR / 'train_labe
```

Let's take a look at the head of our training features

In [4]:
```
train_values.head()
```

Out[4]:

	slope_of_peak_exercise_st_segment	thal	re
patient_id			
0z64un	1	normal	12
ryoo3j	2	normal	11
yt1s1x	1	normal	12
l2xjde	1	reversible_defect	15
oyt4ek	3	reversible_defect	17

In [5]:
```
train_values.dtypes
```

Out[5]:
```
slope_of_peak_exercise_st_segment          int64
thal                                      object
resting_blood_pressure                     int64
chest_pain_type                            int64
num_major_vessels                          int64
fasting_blood_sugar_gt_120_mg_per_dl       int64
resting_ekg_results                        int64
serum_cholesterol_mg_per_dl                int64
oldpeak_eq_st_depression                 float64
sex                                        int64
age                                        int64
max_heart_rate_achieved                    int64
exercise_induced_angina                    int64
dtype: object
```

And the labels

In [6]:
```
train_labels.head()
```

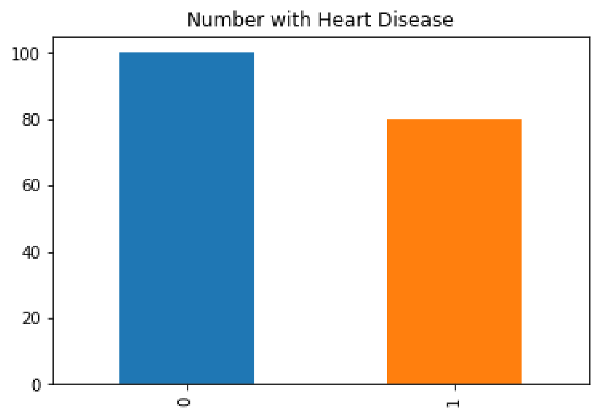
Out[6]:

	heart_disease_present
patient_id	
0z64un	0
ryoo3j	0
yt1s1x	1
l2xjde	1
oyt4ek	0

EXPLORE THE DATA

In [7]:
```
train_labels.heart_disease_present.value_counts()
```

Out[7]: <matplotlib.axes._subplots.AxesSubplot at 0x114



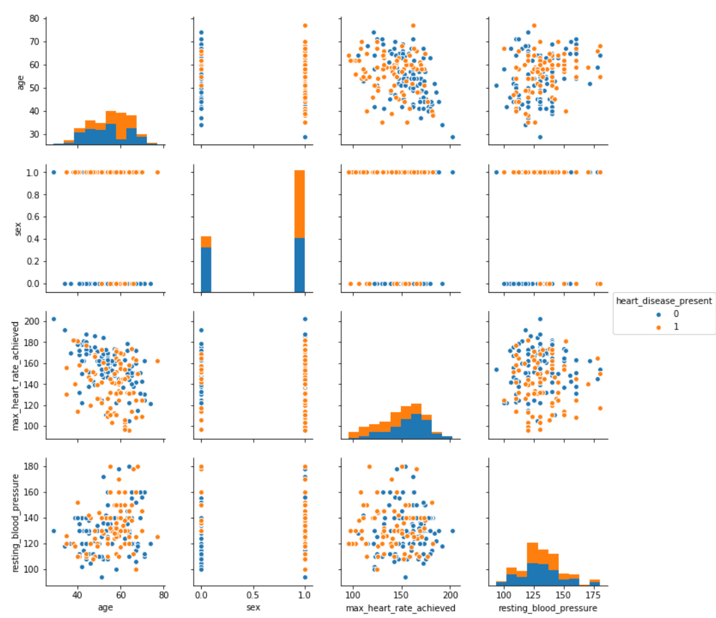
The data is relatively well-balanced, so we won't take any steps here to equalize the classes.

In [8]:
```
selected_features = ['age',
                     'sex',
                     'max_heart_rate_achieved',
                     'resting_blood_pressure']
train_values_subset = train_values[selected_featu
```

A quick look at the relationships between our features and labels

In [9]:
```
sns.pairplot(train_values.join(train_labels),
             hue='heart_disease_present',
             vars=selected_features)
```

Out[9]: <seaborn.axisgrid.PairGrid at 0x10f246978>



THE ERROR METRIC – LOGLOSS

The metric in this competition is logarithmic loss, or *log loss*, which uses the *probabilities* of class predictions and the true class labels to generate a number that is closer to zero for better models, and exactly zero for a perfect model.

You can see from the formula for log loss(http://wiki.fast.ai/index.php/Log_Loss) that highly *confident* (probability close to one) *wrong* answers will contribute more to the total log loss number. This property of log loss makes it more informative alternative to accuracy. Below we'll use the Scikit Learn implementation of log loss to evaluate our model before submitting to the leaderboard.

BUILD THE MODEL

When it comes to classic first pass models, few can contend with logisitc regression. This linear model is fast to train, easy to understand, and typically does pretty well "out of the box".

Below we'll combine the Scikit Learn logistic regression model with a preprocessing tool using `Pipeline` and `GridSearchCV`––two of `sklearn`'s tools for streamlining the process of model training and hyperparamter optimization. You may be new to machine learning, but there's no better time to start developing good habits, and using pipelines is a good habit(https://signal-to-noise.xyz/post/sklearn-pipeline/) that you won't regret learning.

Logisitc Regression

In [10]:
```
# for preprocessing the data
from sklearn.preprocessing import StandardScaler

# the model
from sklearn.linear_model import LogisticRegressi

# for combining the preprocess with model trainin

from sklearn.pipeline import Pipeline

# for optimizing parameters of the pipeline
from sklearn.model_selection import GridSearchCV
```

In `Pipeline`s you pass a list of "steps" in the form of tuples with a step name in the first field and the object associated to the step in the second. Pipeline then creates one "estimator" object that you can pass data into for training and predcition.

In [11]:
```
pipe = Pipeline(steps=[('scale', StandardScaler()
                       ('logistic', LogisticRegre
pipe
```

Out[11]:
```
Pipeline(memory=None,
     steps=[('scale', StandardScaler(copy=True,
          intercept_scaling=1, max_iter=100, mu
          penalty='l2', random_state=None, solv
          verbose=0, warm_start=False))])
```

Grid search allows you try out different parameters in the pipeline. Below, we try out different values for Scikit Learn's logistic regression(http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html) "C" parameter as well as its regularization method.

To specify that these parameters are for the `LogisticRegression` part of the pipeline and not the `StandardScaler` part, the keys in our parameter grid (a python dictionary) take the form `stepname__parametername`. (Note the **double underscore**!)

The CV in `GridSearchCV` is another best practice to prevent overfitting(http://scikit-learn.org/stable/modules/cross_validation.html) your model.

In [12]:
```
param_grid = {'logistic__C': [0.0001, 0.001, 0.01
              'logistic__penalty': ['l1', 'l2']}
gs = GridSearchCV(estimator=pipe,
                  param_grid=param_grid,
                  cv=3)
```

With the parameter grid we've created and cross-validation, we're about to test 30 different models and take the best one!

In [13]:
```
gs.fit(train_values_subset, train_labels.heart_di
```

Out[13]:
```
GridSearchCV(cv=3, error_score='raise',
       estimator=Pipeline(memory=None,
     steps=[('scale', StandardScaler(copy=True,
          intercept_scaling=1, max_iter=100, mu
          penalty='l2', random_state=None, solv
          verbose=0, warm_start=False))]),
       fit_params=None, iid=True, n_jobs=1,
       param_grid={'logistic__C': [0.0001, 0.00
       pre_dispatch='2*n_jobs', refit=True, ret
       scoring=None, verbose=0)
```

Let's take a look at the best parameters.

In [14]:
```
gs.best_params_
```

Out[14]:
```
{'logistic__C': 1, 'logistic__penalty': 'l2'}
```

And the in-sample log loss score. Notice that since log loss wants the class probilities we call `predict_proba` and not simply `predict`, which would return the predicted labels, not their probabilites.

In [15]:
```
from sklearn.metrics import log_loss

in_sample_preds = gs.predict_proba(train_values[s
log_loss(train_labels.heart_disease_present, in_s
```

Out[15]: 0.546686009615344

TIME TO PREDICT AND SUBMIT

For the log loss, we'll be using the **class probabilities, not the class predictions**. This means we call the `predict_proba` method as above. Further, we'll take every row in the second column, which corresponds to the positive case of `heart_disease_present`.

Let's load up the data, process it, and see what we get on the leaderboard.

In [16]:
```
test_values = pd.read_csv(DATA_DIR / 'test_values
```

Select the subset of features we used to train the model.

In [17]:
```
test_values_subset = test_values[selected_feature
```

Make Predictions

Again, note that we take only the second column.

In [18]:
```
predictions = gs.predict_proba(test_values_subset
```

Save Submission

We can use the column name and index from the submission format to ensure our predictions are in the form.

In [19]:
```
submission_format = pd.read_csv(DATA_DIR / 'submi
```

In [20]:
```
my_submission = pd.DataFrame(data=predictions,
                             columns=submission_f
                             index=submission_for
```

In [21]:
```
my_submission.head()
```

Out[21]:

	heart_disease_present
patient_id	
olalu7	0.604407
z9n6mx	0.053553
5k4413	0.752436
mrg7q5	0.097412
uki4do	0.791249

In [22]:
```
my_submission.to_csv('submission.csv')
```

Check the head of the saved file

In [23]:
```
!head submission.csv
```

```
patient_id,heart_disease_present
olalu7,0.6044072540043635
z9n6mx,0.05355289481734789
5k4413,0.7524357630080116
mrg7q5,0.09741203030747234
uki4do,0.7912486789601635
kev1sk,0.08747047862618101
9n6let,0.5409852616595142
jxmtyg,0.6565676676376131
51s2ff,0.301744665671968
```

Submit to leaderboard

Woohoo! We processed your submission!

Your score for this submission is:

0.5381

Woohoo! It's a start! And that's exactly what we intend with these benchmarks. We're sure you'll be able to top this model in no time, and we can't wait to see what you come up with(http://www.drivendata.org/competitions/). Happy importing!