Contest 1 Second Place Solution

Edward Zhang, Sauman Das (Super Sophomores)

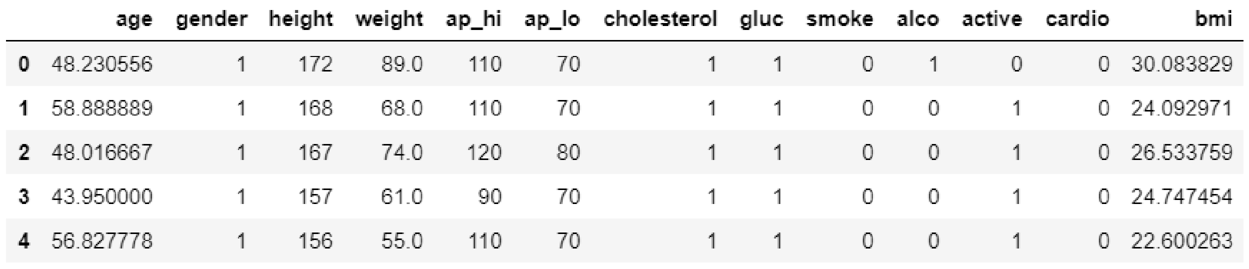
August 26, 2020

Abstract

In this paper, we outline our strategy for solving the first contest presented by the TJ National Machine Learning Open (NMLO). The first contest of the TJ NMLO requires competitors to predict whether someone has a cardiovascular disease given info about their health. We achieved an accuracy of 72.714 percent using random forest classifier. Next we go over our data cleaning, experimental data analysis, feature engineering, modeling, and testing.

1 Data

Before beginning to write the code for this contest, we looked at the data to understand what we were working with and to see which variables could be important and vice versa. According to the description, there are 4000 rows (cardiovascular disease cases) and 12 columns, details about each case. The column features include age, gender, height, weight, systolic blood pressure, diastolic blood pressure, cholesterol levels, glucose levels, smoking, alcohol, and activity.



	age	gender	height	weight	ap_hi	ap_lo	cholesterol	gluc	smoke	alco	active	cardio	bmi
0	48.230556	1	172	89.0	110	70	1	1	0	1	0	0	30.083829
1	58.888889	1	168	68.0	110	70	1	1	0	0	1	0	24.092971
2	48.016667	1	167	74.0	120	80	1	1	0	0	1	0	26.533759
3	43.950000	1	157	61.0	90	70	1	1	0	0	1	0	24.747454
4	56.827778	1	156	55.0	110	70	1	1	0	0	1	0	22.600263

Table 1: First five rows of the data

1.1 Data Cleaning

The first thing we did was to check for missing values and the type of each variable of each case analysis. We found that there were **0 missing values** and All of the variable types were either **int 64** or **floats.** Thus, we did not have to any data imputation nor one hot encoding since there were no categorical variables. Next, we dropped all duplicate values in the dataset since there is no effect with training.

2 Exploratory Data Analysis

We started to explore the data with a preliminary correlation heatmap. We hoped to gain some insight on some relationships between the parameters before we did any graphing.

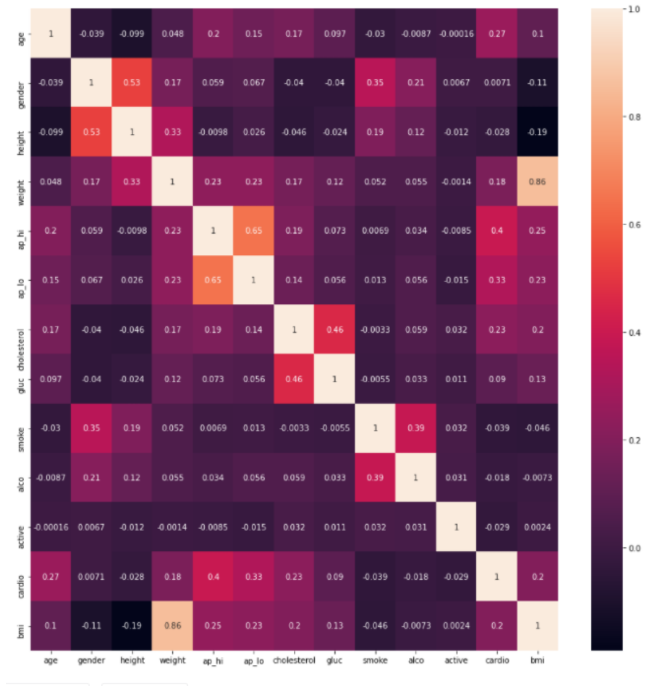
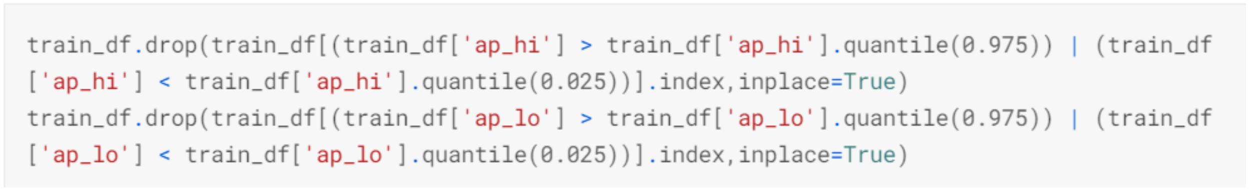


Figure 1: Correlation Heatmap

We can see from correlation map easily; cholesterol, blood pressure (ap_hi and ap_low both) and age have a powerful relationship with cardiovascular diseases.

First, we looked at blood pressure. After doing some research we found that in some cases diastolic pressure were higher than systolic, which is incorrect (according to the Mayo Clinic, diastolic pressure cannot be higher than systolic). We simply dropped these rows



```
train_df.drop(train_df[(train_df['ap_hi'] > train_df['ap_hi'].quantile(0.975)) | (train_df
['ap_hi'] < train_df['ap_hi'].quantile(0.025))].index,inplace=True)
train_df.drop(train_df[(train_df['ap_lo'] > train_df['ap_lo'].quantile(0.975)) | (train_df
['ap_lo'] < train_df['ap_lo'].quantile(0.025))].index,inplace=True)
```

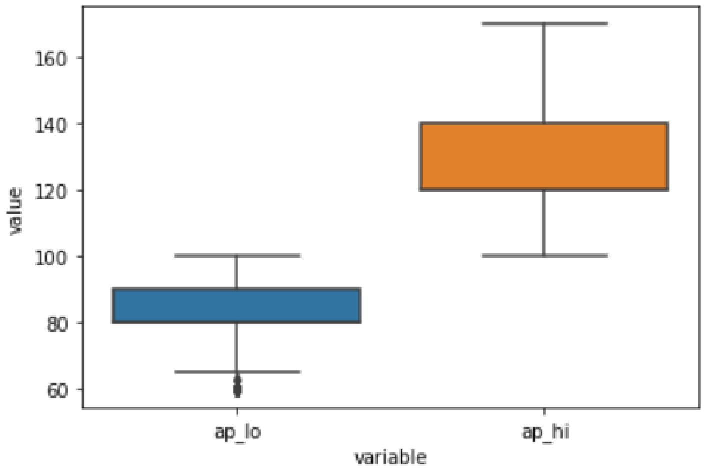


Figure 2: Boxplots of Blood Pressure

Height and weight were next, we began by graphing the data.

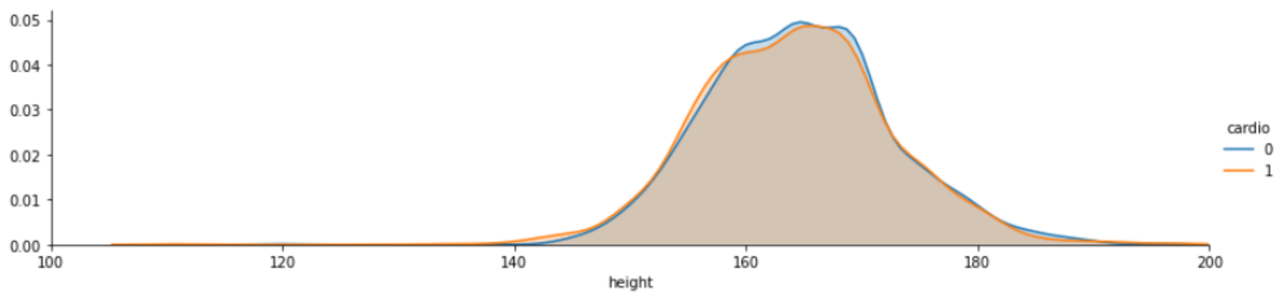


Figure 3: Height on Cardiovascular Disease

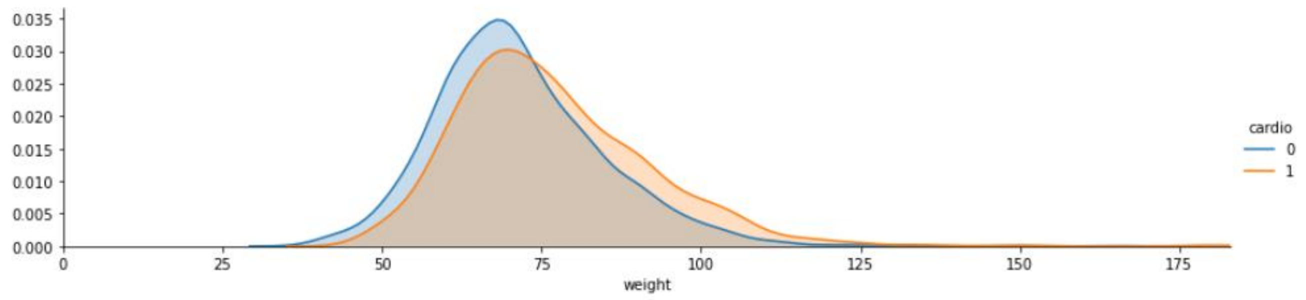


Figure 4: Weight on Cardiovascular Disease

Note: the rest of the data did not show much of a correlation, and we did not augment the data. Therefore, we are not going to include it in this paper.

3 Feature Engineering

For this contest, we did not do too much feature engineering since most of the data we augmented did not have that big of an affect. We only added two more features: **BMI** and **High Blood Pressure**

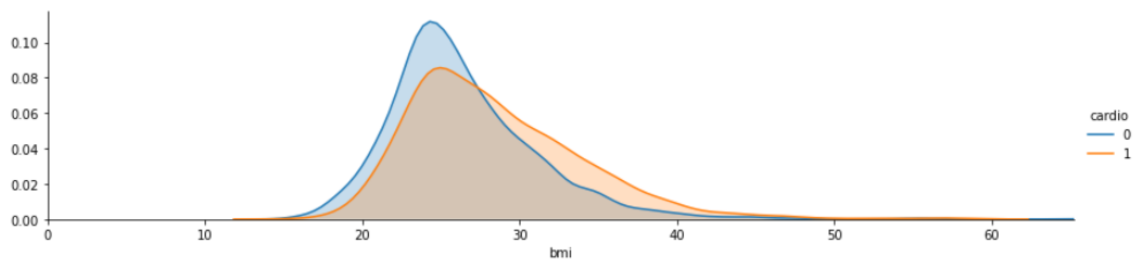


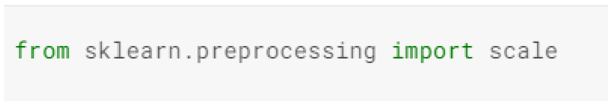
Figure 5: BMI on Cardiovascular Disease

We found that BMI was a better representation of both height and weight. There is a now a noticeable difference in the two distributions.

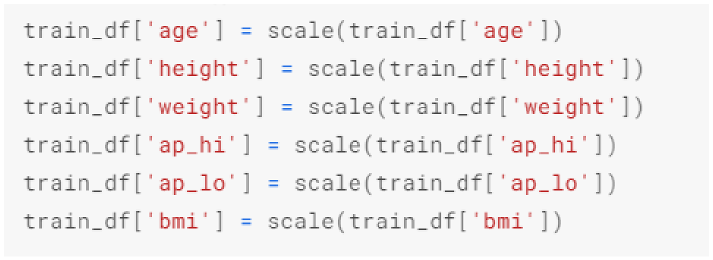
```
train_df['bmi'] = train_df['weight']/((train_df['height']/100.0)**2)
train_df['high_bp'] = [int(hi > 130 or lo > 80) for hi, lo in zip(train_df['ap_hi'], train_df
['ap_lo'])]
```

4 Preprocessing

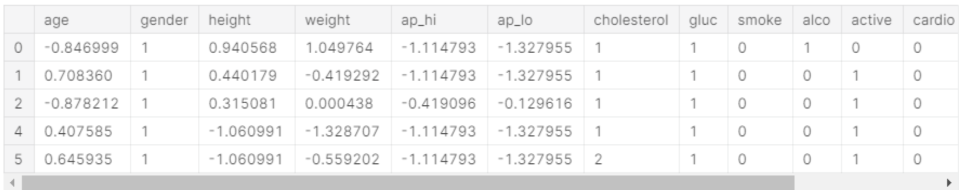
Now that we got most of our data ready, we decided to scale it so modeling would be much easier and faster. From sklearn we imported the scale function and scaled most of our numerical variables.



```
from sklearn.preprocessing import scale
```

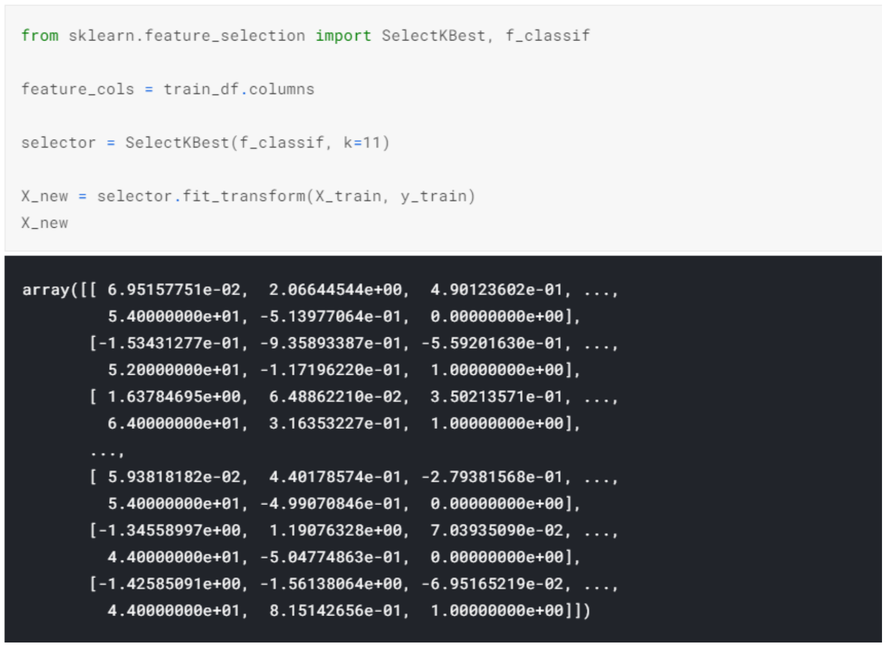


```
train_df['age'] = scale(train_df['age'])
train_df['height'] = scale(train_df['height'])
train_df['weight'] = scale(train_df['weight'])
train_df['ap_hi'] = scale(train_df['ap_hi'])
train_df['ap_lo'] = scale(train_df['ap_lo'])
train_df['bmi'] = scale(train_df['bmi'])
```



	age	gender	height	weight	ap_hi	ap_lo	cholesterol	gluc	smoke	alco	active	cardio
0	-0.846999	1	0.940568	1.049764	-1.114793	-1.327955	1	1	0	1	0	0
1	0.708360	1	0.440179	-0.419292	-1.114793	-1.327955	1	1	0	0	1	0
2	-0.878212	1	0.315081	0.000438	-0.419096	-0.129616	1	1	0	0	1	0
4	0.407585	1	-1.060991	-1.328707	-1.114793	-1.327955	1	1	0	0	1	0
5	0.645935	1	-1.060991	-0.559202	-1.114793	-1.327955	2	1	0	0	1	0

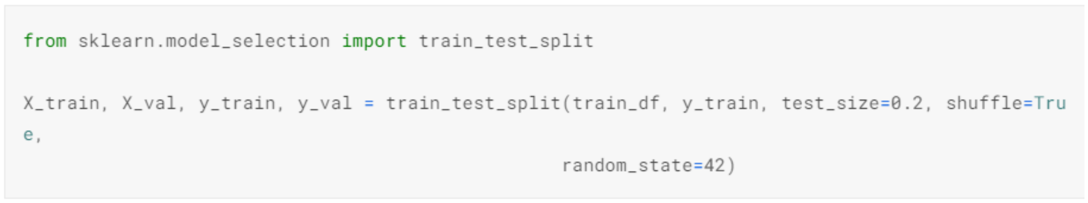
Next, we used Select K Best to choose the n most relevant features for modeling. This is important because it essentially drops the features that are not really important and could cloud out the model.



```
from sklearn.feature_selection import SelectKBest, f_classif

feature_cols = train_df.columns

selector = SelectKBest(f_classif, k=11)

X_new = selector.fit_transform(X_train, y_train)
X_new
```

```
array([[ 6.95157751e-02,  2.06644544e+00,  4.90123602e-01, ...,
         5.40000000e+01, -5.13977064e-01,  0.00000000e+00],
       [-1.53431277e-01, -9.35893387e-01, -5.59201630e-01, ...,
         5.20000000e+01, -1.17196220e-01,  1.00000000e+00],
       [ 1.63784695e+00,  6.48862210e-02,  3.50213571e-01, ...,
         6.40000000e+01,  3.16353227e-01,  1.00000000e+00],
       ...,
       [ 5.93818182e-02,  4.40178574e-01, -2.79381568e-01, ...,
         5.40000000e+01, -4.99070846e-01,  0.00000000e+00],
       [-1.34558997e+00,  1.19076328e+00,  7.03935090e-02, ...,
         4.40000000e+01, -5.04774863e-01,  0.00000000e+00],
       [-1.42585091e+00, -1.56138064e+00, -6.95165219e-02, ...,
         4.40000000e+01,  8.15142656e-01,  1.00000000e+00]])
```

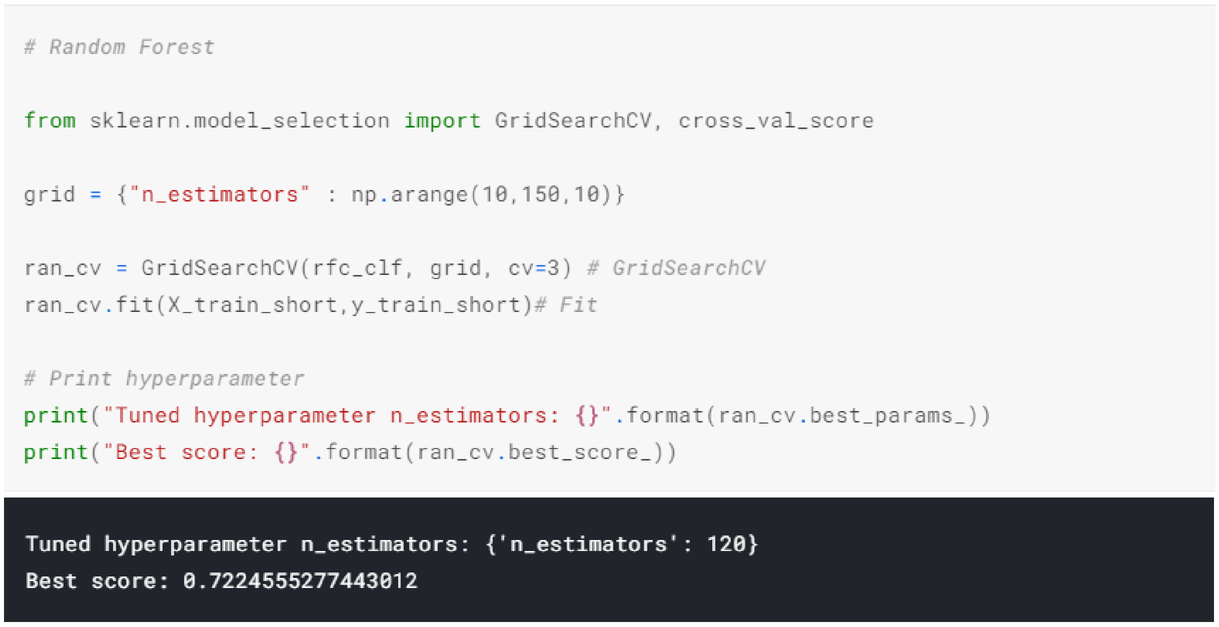
Finally, we split the training data into train and validation sets, to train our model.



```
from sklearn.model_selection import train_test_split

X_train, X_val, y_train, y_val = train_test_split(train_df, y_train, test_size=0.2, shuffle=Tru
e,
                                                   random_state=42)
```

5 Modeling

We initially were going to use XGBoost and Light XGboost models, but they were prohibited from use in this competition. Nevertheless, we tried both **random forest classifier** and **decision tree**, but we got better results with random forest. Finally, we used **grid-search-cv** to hyper-tune our model, so that all of the parameters such as the number of trees is optimal.



```
# Random Forest

from sklearn.model_selection import GridSearchCV, cross_val_score

grid = {"n_estimators" : np.arange(10,150,10)}

ran_cv = GridSearchCV(rfc_clf, grid, cv=3) # GridSearchCV
ran_cv.fit(X_train_short,y_train_short)# Fit

# Print hyperparameter
print("Tuned hyperparameter n_estimators: {}".format(ran_cv.best_params_))
print("Best score: {}".format(ran_cv.best_score_))
```

```
Tuned hyperparameter n_estimators: {'n_estimators': 120}
Best score: 0.7224555277443012
```

6 Final Thoughts

We initially were **6th** place on the public leaderboard, but when the final results were up, we got **2nd** on the private leaderboard. My partner Sauman and I chose a model that did not perform the best on the public leaderboard because we knew that we probably **over fit** to the public leaderboard. I believe a lot of people overfit on the public leaderboard and therefore, fell down a bit on the private. Overall, this was a fun competition and we enjoyed every day of it.