ML Model & Forensics

November 9, 2020

1 Overview

Hypothesis: *We can use supervised machine learning to predict if a patient might be classified as having chronic kidney disease based on biometrics*

In this experiment we look at the captured data of patients with or without ESRD and based on this generate a machine learning model that predicts if a patient does or does not have ESRD. After this it will list the highest variables contributing to the predictions. The goal is to take this information and have a patient input the information and get the accuracy of a result and pinpoint what of their specific variables contribute the most to this. We will be using a randomforest classification as well as cross validation to determine the best parameters for the model.

1.0.1 Importing all the necessary packages

```
In [1]: import pandas as pd
        import numpy as np
        import matplotlib.pyplot as plt
        from sklearn.ensemble import RandomForestClassifier
        from sklearn.model_selection import train_test_split
        from sklearn.model_selection import GridSearchCV
        from sklearn.metrics import accuracy_score
        from sklearn.metrics import auc
        from sklearn.metrics import roc_auc_score
        from sklearn.metrics import confusion_matrix
        import warnings
        warnings.filterwarnings('ignore')
```

1.0.2 Loading the data

The data has already been linked and stored in a bucket in GCP. **Data Rules** -Ingesting data is from an open dataset from UCI research. -No PHI is obtained from the data. -Data can be gathered from a health screening with blood work and past/current doctor's insight. --This is the input data of a patient, only input information known within 60 days. -Continue to add input data as training data until it reaches 2000 observations. -Return risk of Kidney Disease in percentage and recommended highest risk factors. **Data Features** -After Unsupervised learning is processed through the dataset, identify the more important features to make these necessary when patient/doctor inputs data. --Setting off a warning that accuracy would be impacted by this. **Description of the variables** age - age bp - blood pressure sg - specific gravity al - albumin su - sugar rbc - red blood

cells pc - pus cell pcc - pus cell clumps ba - bacteria bgr - blood glucose random bu - blood urea sc - serum creatinine sod - sodium pot - potassium hemo - hemoglobin pcv - packed cell volume wc - white blood cell count rc - red blood cell count htn - hypertension dm - diabetes mellitus cad - coronary artery disease appet - appetite pe - pedal edema ane - anemia

```
In [2]: path = "./kidney_disease_output.csv"
        kidney = pd.read_csv(path).iloc[:, 1:] #removing the id column
        # changing format to have all numeric factors for the variables
        kidney["rbc"] = kidney["rbc"].eq("abnormal").mul(1)
        kidney["pc"] = kidney["pc"].eq("abnormal").mul(1)
        kidney["pcc"] = kidney["pcc"].eq("present").mul(1)
        kidney["ba"] = kidney["ba"].eq("present").mul(1)
        kidney["htn"] = kidney["htn"].astype("int64")
        kidney["dm"] = kidney["dm"].astype("int64")
        kidney["cad"] = kidney["cad"].astype("int64")
        kidney["appet"] = kidney["appet"].eq("poor").mul(1)
        kidney["pe"] = kidney["pe"].astype("int64")
        kidney["ane"] = kidney["ane"].astype("int64")
        #print(kidney.shape)
        #print(kidney.dtypes)
        #print(kidney.describe())
        display(kidney.head())
```

	age	bp	sg	al	su	rbc	pc	pcc	ba	bgr	...	pcv	wc	\
0	48	80	1.020	1	0	0	0	0	0	121	...	44	7800	
1	7	50	1.020	4	0	0	0	0	0	148	...	38	6000	
2	62	80	1.010	2	3	0	0	0	0	423	...	31	7500	
3	48	70	1.005	4	0	0	1	1	0	117	...	32	6700	
4	51	80	1.010	2	0	0	0	0	0	106	...	35	7300	

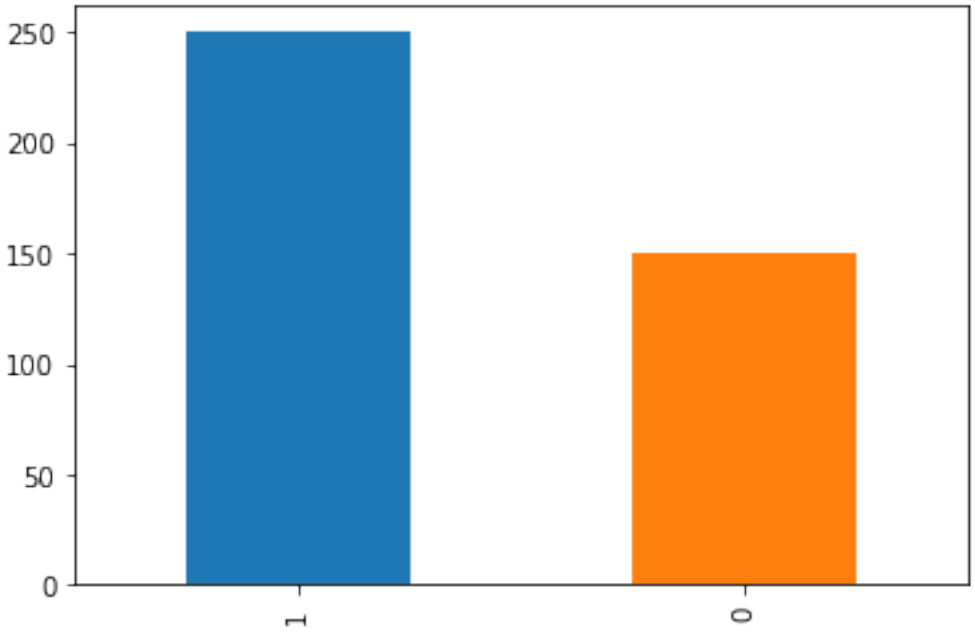
	rc	htn	dm	cad	appet	pe	ane	classification
0	5.2	1	1	0	0	0	0	1
1	4.7	0	0	0	0	0	0	1
2	4.7	0	1	0	1	0	1	1
3	3.9	1	0	0	1	1	1	1
4	4.6	0	0	0	0	0	0	1

[5 rows x 25 columns]

We want to ensure that we don't have an unbalanced set of data by checking the classification counts.

```
In [3]: kidney["classification"].value_counts().plot(kind = 'bar')
```

```
Out[3]: <matplotlib.axes._subplots.AxesSubplot at 0x239ff400d30>
```



1.0.3 Predicting the outcome of a patient's inputs as 1-ESRD or 0-no ESRD

We would need a machine learning model that has a high accuracy and utilizes all the variables provided, we verify this by having the performace produced at the end once the model has been run. Since this is an experiment we will be splitting our whole data for training/test purposes. Once this goes into escalability the "test" portion is what the patient will input.

```
In [4]: # Have a function that will take care of presenting the predictions and the model perf
        # This is to ensure that our predictions are accurate.
        predictions = []
        def evaluate(model, test_features, test_labels, pred=False):
            global predictions
            predictions = model.predict(test_features)
            errors = abs(predictions - test_labels)
            accuracy = accuracy_score(test_labels, predictions) * 100
            roc_auc_s = roc_auc_score(test_labels, predictions) * 100
            if(pred==True):
                print("Prediction(s):")
                print(predictions)
            print('Model Performance')
            print('Average Error: {:0.4f} degrees.'.format(np.mean(errors)))
            print('Accuracy = {:0.2f}%.'.format(accuracy))
            print("ROC AUC Score = {:0.2f}%" .format(roc_auc_s))
            conf_mat = confusion_matrix(y_true=test_labels, y_pred=predictions)
            print("Confusion matrix:\n", conf_mat)
```

```
            return accuracy
            return roc_auc_s
```

```
In [5]: y = kidney.classification
        X = kidney.iloc[:, 0:24]
        X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.3, random_state=
```

```
In [6]: param_grid = {
            "n_estimators":[50, 100, 200] # we want to test by having a random forest of diffe
        }
        rf = RandomForestClassifier()
        grid_search = GridSearchCV(estimator=rf, param_grid=param_grid, cv=5, n_jobs=-1, verbos
        grid_search.fit(X_train, y_train)
        grid_search.best_params_
```

```
Fitting 5 folds for each of 3 candidates, totalling 15 fits


[Parallel(n_jobs=-1)]: Using backend LokyBackend with 12 concurrent workers.
[Parallel(n_jobs=-1)]: Done   8 out of  15 | elapsed:    3.7s remaining:    3.3s
[Parallel(n_jobs=-1)]: Done  15 out of  15 | elapsed:    3.9s finished


Out[6]: {'n_estimators': 50}
```

```
In [7]: # Once we have the best estimator for the randomforest model, we run the testing datas
        best_grid = grid_search.best_estimator_
        grid_accuracy = evaluate(best_grid, X_test, y_test, True)
```

```
Prediction(s):
[1 0 1 0 1 0 1 1 0 1 1 1 1 1 1 1 1 1 0 1 1 0 1 1 1 1 1 1 1 1 0 1 1 0 1 1 0
 0 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 1 1 1 1 0 1 0 1 0 1 1 0 1 1 1 1 1 1 0 1 0
 1 1 1 1 1 1 1 0 1 1 1 0 1 1 1 1 1 1 1 1 1 0 1 1 0 0 1 0 0 0 1 0 1 0 0 1 1
 1 1 1 0 0 1 1 1 1 1 1 1 1 0 0 0 0 1 0 1 1 1 0 1 1 0 0 1 0 0 1 1 1 0 1 1 0
 1 1 1 1 0 1 1 1 0 1 1 1 0 1 0 1 1 1 1 1 1 1 1 0 0 0 0 0 1 0 1 1 0 1 1 0 0
 0 1 0 0 1 1 1 0 1 0 1 1 1 1 1 1 1 1 0 1 1 0 1 0 0 1 0 1 1 0 1 0 1 1 1 0 1
 0 0 0 0 1 1 1 1 1 1 0 1 0 0 1 1 1 0 1 1 1 0 1 0 1 0 1 1 1 1 0 1 1 1 0 0 1 0
 0 1 0 1 0 1 0 1 0 1 1 1 1 1 0 0 0 1 1 1 1 1 0]
Model Performance
Average Error: 0.0143 degrees.
Accuracy = 98.57%.
ROC AUC Score = 98.19%
Confusion matrix:
 [[ 95   3]
 [  1 181]]
```

1.0.4 Analysis

We are able to see that we can indeed accurately predict the result of a patient's condition based on their biometrics to a 98%+ accuracy and this is supported by the ROC AUC and the confusion matrix. The next part will take care of identifying which of the variables are the most important. We can use a method for feature reduction to rank them; however, we will be using all of the biometrics, but having these to be the *must be input* for accurate predictions.

```
In [9]: from sklearn.feature_selection import SelectKBest
        from sklearn.feature_selection import chi2
        bestfeatures = SelectKBest(score_func=chi2, k=10)
        fit = bestfeatures.fit(X, y)
        dfscores = pd.DataFrame(fit.scores_)
        dfcolumns = pd.DataFrame(X.columns)
        featureScores = pd.concat([dfcolumns,dfscores], axis=1)
        featureScores.columns = ["Variables", "Score"]
        print(featureScores.nlargest(10, "Score"))
```

```
   Variables         Score
16        wc  12732.774341
9        bgr   2428.048707
10        bu   2336.198900
11        sc    360.413289
15       pcv    325.110615
3         al    216.000000
14      hemo    125.369169
4         su     94.800000
18       htn     88.200000
19        dm     81.600000
```

```
In [10]: output = pd.concat([X_test.reset_index(drop=True), y_test.reset_index(drop=True)], axi
         predictions = pd.DataFrame(data=predictions).rename(columns={0:"prediction"})
         output = pd.concat([output.reset_index(drop=True), predictions.reset_index(drop=True)]
         output["accurate"] = np.where((output["classification"]==output["prediction"]), 1, 0)
```

```
In [11]: output.to_csv("./output.csv")
```

```
In [1]: %%R
        library(tidyverse)
        install.packages("cowplot")
        library(cowplot)
```

```
UsageError: Cell magic `%%R` not found.
```