Data Preparation

Missing Values

First of all, our dataset is 25 columns with 400 rows which contains *NA* and unclear data values. There are '?', '\t43' and 'abnormal' type values. There are multiple ways to handle these cases. In this case, '?' values replaced with NA and '\t' values are extracted from the data. Categorical Columns and their respective binary labels are presented in the table 1 below. NA values in *age*, *p, sg, bgr, bu, sc, sod, hemo, pcv, wbcc, rbcc* columns are filled with their mean. Moreover, column *rbc* has around 152 NA values and since it is a categorical feature, I removed it from the feature's dataset since it has not an important feature according to feature importance ranking, which can be seen in table 2. I tried to define the score features by using a *sklearn.feature_selection* module. Analyzed the best features by using *chi2* as a scoring function inside of the *SelectKBest* module. Note that all features used in pipeline.

Rest of the columns consists *NA* are dropped out. The reason I did not dropped NA's at the beginning is that we have few rows left when I dropped at first. This resulted an overfitting case on the dataset. After all, there left 254 rows in the dataset.

Columns	Labels	
	1	0
htn	yes	no
dm	yes	no
cad	yes	no
pe	yes	no
ane	yes	no
rbc	abnormal	normal
pc	abnormal	normal
pcc	present	notpresent
ba	present	notpresent
appet	good	poor
class	ckd	nockd

Table 1: Categorical Columns with their respective binary labels

Feature	Score
wbcc	20,303.4
bu	3,753.4
bgr	1,748.7
sc	437.8
al	344.0
pcv	262.3
su	111.1
htn	93.2
hemo	88.7
age	81.3
pc	77.0
dm	77.0

Table 2: Features with Scores

Cross Validation & Model Selection

For this dataset, I tried multiple models such as: ***Decision Tree Classifier, Bagging Classifier, Random Forest Classifier, AdaBoost Classifier & Gradient Boosting Classifier.*** Main aim is to use these models in a cross validation scheme, with Strafied CV, K = 5 and Kfold, K = 5, and select the best model to tune.

With a seed of 84, the performance criteria of the assignment is *accuracy* score, below we can see the accuracy scores of each model, individually.

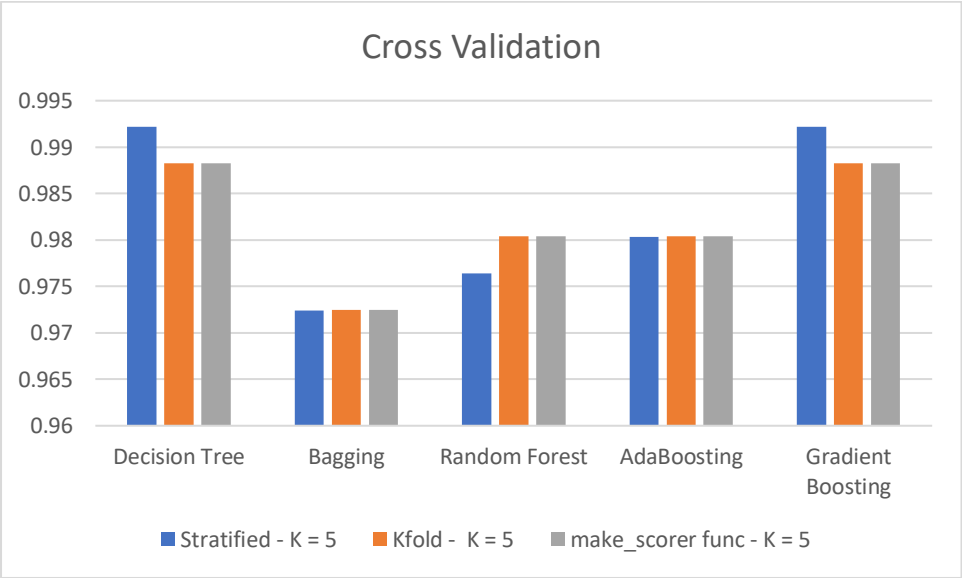


Chart 1: Estimators with Accuracy scores

I used *cross_val_score* method from sklearn.metrics library with scoring parameter as 'accuracy'. I also used the *make_scorer* method since sometimes the train and test accuracy scores differs much more with or without cross validation scheme. I used K = 5 since data is pretty small and all the classifiers' parameters are default values. As can be seen from the chart 1, Decision Tree and Gradient Boosting were pretty good according to the others. At the end, I continued with Gradient

Boosting and Stratified method since it has one of the highest classifier with an accuracy score of 0.99.

Hyper-Tuning

Main aim is to beat the scores at pre-step, which is pretty high. I used RandomSearch to find the best parameter list, and use these best parameters in GridSearch to achive the optimal. Following parameters are used in random search cv.

```
params = {
        'n_estimators': list(np.linspace(10, 300, 10, dtype = int)),
        'learning_rate': [0.01, 0.05, 0.1, 0.15],
        'max_depth': [3, 4, 5, 6, 7],
        'random_state': [42],
        'max_features': [10, 15, 20]
        }
```

Random Search CV resulted with a train score of 1.0 and test score of 0.987. The best parameters found in Random Search is:

```
best_params = {
        'random_state': 84,
        'n_estimators': 300,
        'max_features': 15,
        'max_depth': 3,
        'learning_rate': 0.15
        }
```

Lastly, in Grid Search CV, used the following parameters:

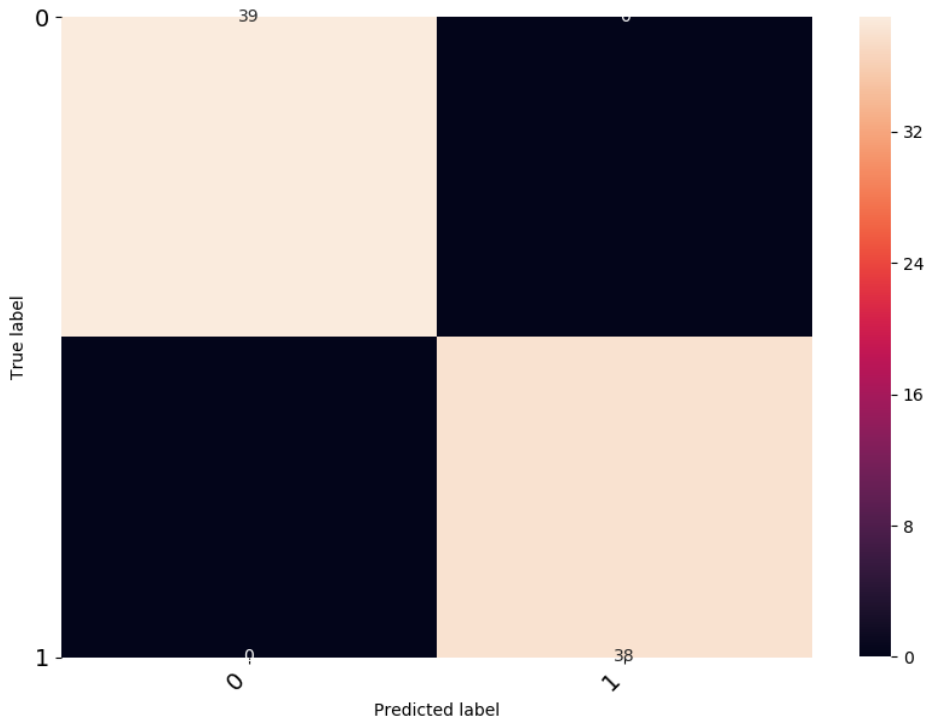
```
grid_search = {
    'max_depth': [model.best_params_['max_depth'] - 1,
                  model.best_params_['max_depth'],
                  model.best_params_['max_depth'] + 1],
    'learning_rate': [model.best_params_['learning_rate'] - 0.01,
                      model.best_params_['learning_rate'],
                      model.best_params_['learning_rate'] + 0.01],
     'max_features': [model.best_params_['max_features'] - 2,
                      model.best_params_['max_features'],
                      model.best_params_['max_features'] + 2],
    'n_estimators': [model.best_params_['n_estimators'] - 10,
                     model.best_params_['n_estimators'],
                     model.best_params_['n_estimators'] + 10]
```

The reason I used best parameters with a range is to find exact solutions since Grid Search searches all the nodes in the space. So, Grid Search resulted with a train score of 1.0 and test score of 1.0,

which is greater than initial results. The model does not overfit since these accuracy metrics are test scores.

```
Optimal_params = {
        'learning_rate': 0.14,
        'max_depth': 2,
        'max_features': 13,
        'n_estimators': 310
        }
```

Confusion Matrix can be found below, which found all the False Negatives and True Positives correctly.



References

- https://scikit-learn.org/stable/
- https://www.datacamp.com/community/news/feature-selection-using-selectkbest-0dv0fo0qqe48
- https://www.kaggle.com/csyhuang/predicting-chronic-kidney-disease/notebook
- https://www.kaggle.com/plarmuseau/kidney-disease-classifiers
- https://www.kaggle.com/pratik2901/sequential-model-for-chronic-kidney-disease/comments
- https://thesai.org/Downloads/Volume10No8/Paper_13-Detection_of_Chronic_Kidney_Disease.pdf