**Hila Borenstein**

**ML Project**

**Index:**

1. [**Project Background** 3](#_Toc26258531)
2. [Database description 3](#_Toc26258532)
3. [Features information 4](#_Toc26258533)
4. [Goal 4](#_Toc26258534)
5. [**Project Requirements** 4](#_Toc26258535)
6. [**Workflow** 6](#_Toc26258536)
7. [Preprocessing-creating csv files 6](#_Toc26258537)
8. [Preprocessing- Dataset selection 7](#_Toc26258538)
9. [Preprocessing -Feature Selection 8](#_Toc26258539)
10. [**Learning process and feature escalation** 9](#_Toc26258540)
11. [K nearest neighbors 9](#_Toc26258541)
12. [SVM 10](#_Toc26258542)
13. [Decision Tree 11](#_Toc26258543)
14. [Random Forest 12](#_Toc26258544)
15. [Bagging 13](#_Toc26258545)
16. [AdaBoost 15](#_Toc26258546)
17. [. GBoost (Gradient Boosting) 17](#_Toc26258547)
18. [XGBoost (Extreme Gradient Boosting) 19](#_Toc26258548)
19. [Voting 21](#_Toc26258549)
20. [**Results** 21](#_Toc26258550)

# Project Background

## Database Description

The dataset describes medical attributes of people and if they had heart disease.

The data is based of survey done between the years 2011-2015.

The original output can be found under ‘original input’ folder.

Dataset was taken from:

<https://www.kaggle.com/alexteboul/heart-disease-health-indicators-dataset/tasks?taskId=6741>

**Important Risk Factors**

Research in the field has identified the following as **important risk factors** for heart disease and other chronic illnesses like diabetes (not in strict order of importance):

* blood pressure (high)
* cholesterol (high)
* smoking
* diabetes
* obesity
* age
* sex
* race
* diet
* exercise
* alcohol consumption
* BMI
* Household Income
* Marital Status
* Sleep
* Time since last checkup
* Education
* Health care coverage
* Mental Health

**Important**: Frames to analyze for each experiment: 500-55500 (in order to avoid data noises)

## Features Information

* The original features :

|  |  |  |  |
| --- | --- | --- | --- |
| Attribute name: | Full name | explanation | values |
| HighBP | High blood pressure | Adults who have been told they have high blood pressure by a doctor, nurse, or other health professional | Values:0/1 |
| HighChol | High cholesterol | Have you EVER been told by a doctor, nurse or other health professional that your blood cholesterol is high? | Values:0/1 |
| CholCheck | Cholesterol check | Cholesterol checks within past five years | Values:0/1 |
| BMI | Body Mass Index (BMI) | Body Mass Index (BMI) |  |
| Smoker |  | Have you smoked at least 100 cigarettes in your entire life? [Note: 5 packs = 100 cigarettes] | Values:0/1 |
| Stroke |  | (Ever told) you had a stroke. |  |
| Diabetes |  | (Ever told) you have diabetes (If "Yes" and respondent is female, ask "Was this only when you were pregnant?". If Respondent says pre-diabetes or borderline diabetes, use response code 4.)  0 is no diabetes, 1 is pre-diabetes, and 2 is diabetes | 0,1,2 |
| PhysActivity | **Physical Activity** | Adults who reported doing physical activity or exercise during the past 30 days other than their regular job | Values:0/1 |
| Fruits | **Diet** | Consume Fruit 1 or more times per day | Values:0/1 |
| Veggies | **Diet** | Consume Vegetables 1 or more times per day | Values:0/1 |
| HvyAlcoholConsump | Heavy Alcohol consumption | Heavy drinkers (adult men having more than 14 drinks per week and adult women having more than 7 drinks per week) | Values:0/1 |
| AnyHealthcare | **Health Care** | Do you have any kind of health care coverage, including health insurance, prepaid plans such as HMOs, or government plans such as Medicare, or Indian Health Service? | Values:0/1 |
| NoDocbcCost | **Health Care** | Was there a time in the past 12 months when you needed to see a doctor but could not because of cost? | Values:0/1 |
| GenHlth | General health | Would you say that in general your health is: | Values: 1-5 |
| MentHlth | **Mental Health** | Now thinking about your mental health, which includes stress, depression, and problems with emotions, for how many days during the past 30 days was your mental health not good? | Values: 0-30 |
| PhysHlth | physical health | Now thinking about your physical health, which includes physical illness and injury, for how many days during the past 30 days was your physical health not good? | Values: 0-30 |
| DiffWalk | difficulty walking | Do you have serious difficulty walking or climbing stairs | Values:0/1 |
| Sex |  | Indicate sex of respondent. | Values:0/1 |
| Age |  | Fourteen-level age category |  |
| Education |  | What is the highest grade or year of school you completed? | Values:  1 Never attended school or only kindergarten  2 Grades 1 through 8 (Elementary) 1  3 Grades 9 through 11 (Some high school)  4 Grade 12 or GED (High school graduate)  5 College 1 year to 3 years (Some college or technical school)  6 College 4 years or more (College graduate)  9 Refused |
| Income |  | Is your annual household income from all sources: (If respondent refuses at any income level, code "Refused.") | Value Label  1 Less than $10,000  2 Less than $15,000 ($10,000 to less than $15,000)  3 Less than $20,000 ($15,000 to less than $20,000)  4 Less than $25,000 ($20,000 to less than $25,000)  5 Less than $35,000 ($25,000 to less than $35,000)  6 Less than $50,000 ($35,000 to less than $50,000)  7 Less than $75,000 ($50,000 to less than $75,000)  8 $75,000 or more  77 Don’t know/Not sure  99 Refused  BLANK Not asked or missing |
| HeartDiseaseorAttack | classification |  | Values:0/1 |

## Goal

Find the main criteria to distinguish between people with heart disease

# Project Requirements

|  |  |  |
| --- | --- | --- |
| python package | information | site |
| pandas | pandas is an open source, BSD-licensed library providing high-performance, easy-to-use data structures and data analysis tools for the [Python](https://www.python.org/) programming language. | <https://pandas.pydata.org/> |
| matplotlib | Matplotlib is a Python 2D plotting library which produces publication quality figures in a variety of hardcopy formats and interactive environments across platforms. | <https://matplotlib.org/> |
| scikit-learn | Open-source ML library for Python. Built on NumPy, SciPy, and Matplotlib.  Scikit-learn is a library in Python that provides many unsupervised and supervised learning algorithms. | <https://scikit-learn.org/stable/> |
| Light GBM | LightGBM is a gradient boosting framework that uses tree-based learning algorithms. It is designed to be distributed and efficient with the following advantages:   * Faster training speed and higher efficiency. * Lower memory usage. * Better accuracy. * Support of parallel and GPU learning   Capable of handling large-scale data. | <https://lightgbm.readthedocs.io/en/latest/> |
| XGBoost | XGBoost provides a parallel tree boosting (also known as GBDT, GBM) that solve many data science problems in a fast and accurate way. | <https://xgboost.readthedocs.io/en/latest/build.html> |
| seaborn | [Seaborn is a Python data visualization library based on matplotlib. It provides a high-level interface for drawing attractive and informative statistical graphics.](https://matplotlib.org/) | <https://seaborn.pydata.org/> |
| numpy | NumPy is the fundamental package for scientific computing with Python. | <https://numpy.org/> |
| Dask (XGBoost) | [XGBoost](https://xgboost.readthedocs.io/) is a powerful and popular library for gradient boosted trees. | <https://ml.dask.org/modules/generated/dask_ml.xgboost.XGBClassifier.html> |
| argparse | The [argparse](https://docs.python.org/3/library/argparse.html" \l "module-argparse" \o "argparse: Command-line option and argument parsing library.) module makes it easy to write user-friendly command-line interfaces. | <https://docs.python.org/3/library/argparse.html> |
| PyDotPlus | PyDotPlus provides a Python Interface to Graphviz’s Dot language. | <https://pypi.org/project/pydotplus/> |
| IPython | IPython provides the following features:   * Interactive shells (terminal and [Qt](https://en.wikipedia.org/wiki/Qt_(framework))-based). * A browser-based [notebook interface](https://en.wikipedia.org/wiki/Notebook_interface) with support for code, text, mathematical expressions, inline plots and other media. * Support for interactive data visualization and use of GUI toolkits. * Flexible, embeddable interpreters to load into one's own projects. * Tools for [parallel computing](https://en.wikipedia.org/wiki/Parallel_computing). | <https://ipython.org/> |
| Graphviz | Graphviz is open source graph visualization software. | <https://www.graphviz.org/> |
| Anaconda | Anaconda is a [free and open-source](https://en.wikipedia.org/wiki/Free_and_open-source) distribution of the [Python](https://en.wikipedia.org/wiki/Python_(programming_language)) and [R](https://en.wikipedia.org/wiki/R_(programming_language)) programming languages for [scientific computing](https://en.wikipedia.org/wiki/Scientific_computing) ([data science](https://en.wikipedia.org/wiki/Data_science), [machine learning](https://en.wikipedia.org/wiki/Machine_learning) applications, large-scale data processing, [predictive analytics](https://en.wikipedia.org/wiki/Predictive_analytics), etc.), that aims to simplify [package management](https://en.wikipedia.org/wiki/Package_management) and deployment. | <https://www.anaconda.com/> |

# Workflow

## Preprocessing-creating csv files

## Preprocessing- Dataset selection

## Preprocessing -Feature Selection

@This code part was taken from

<https://github.com/WillKoehrsen/feature-selector/blob/master/Feature%20Selector%20Usage.ipynb>

and developed by Will Koehrsen (Data Scientist at Cortex Intel)

I used several methods to figure out which features are more significant

The code is under FeatureSelection.py

* **identify\_missing**

The first method for finding features to remove is straightforward: find features with a fraction of missing values above a specified threshold.

* **identify\_single\_unique**

[find any columns that have a single unique value.](https://github.com/Featuretools/featuretools/blob/master/featuretools/selection/selection.py) A feature with only one unique value cannot be useful for machine learning because this [feature has zero variance](https://www.r-bloggers.com/near-zero-variance-predictors-should-we-remove-them/).

* **identify\_collinear**

[Collinear features](https://www.quora.com/Why-is-multicollinearity-bad-in-laymans-terms-In-feature-selection-for-a-regression-model-intended-for-use-in-prediction-why-is-it-a-bad-thing-to-have-multicollinearity-or-highly-correlated-independent-variables) are features that are highly correlated with one another. In machine learning, these lead to decreased generalization performance on the test set due to high variance and less model interpretability.

* **identify\_zero\_importance**

The next method is designed only for supervised machine learning problems where we have labels for training a model and is non-deterministic. The identify\_zero\_importance function finds features that have zero importance according to a gradient boosting machine (GBM) learning model. In a tree-based model, the [features with zero importance are not used to split any nodes](https://www.salford-systems.com/blog/dan-steinberg/what-is-the-variable-importance-measure), and so we can remove them without affecting model performance.

The parameters we pass in are as follows:

* + **task**: either “classification” or “regression” corresponding to our problem
  + **eval\_metric:** metric to use for early stopping (not necessary if early stopping is disabled)
  + **n\_iterations :** number of training runs to average the feature importances over
  + **early\_stopping:** whether use early stopping for training the model
* **identify\_low\_importance**

The next method builds on zero importance function.

The function finds the lowest importance features that do not contribute to a specified total importance.

For example, the call below finds the least important features that are not required for achieving 99% of the total importance:

*fs.identify\_low\_importance(cumulative\_importance = 0.99)*

Based on the plot of cumulative importance and this information, the gradient boosting machine considers many of the features to be irrelevant for learning. Again, the results of this method will change on each training run.

The low\_importance method borrows from one of the methods of [using Principal Components Analysis (PCA)](https://towardsdatascience.com/pca-using-python-scikit-learn-e653f8989e60)where it is common to keep only the PC needed to retain a certain percentage of the variance (such as 95%). The percentage of total importance accounted for is based on the same idea.

The feature importance-based methods are really only applicable if we are going to use a tree-based model for making predictions. Besides being stochastic, the importance-based methods are a black-box approach in that we don’t really know why the model considers the features to be irrelevant.

## Learning process and feature escalation

### K nearest neighbors

*The principle behind nearest neighbor methods is to find a predefined number of training samples closest in distance to the new point and predict the label from these. The number of samples can be a user-defined constant (k-nearest neighbor learning) or vary based on the local density of points (radius-based neighbor learning). The distance can, in general, be any metric measure: standard Euclidean distance is the most common choice.*

I used K nearest neighbors from scikit-learn and modified it per my requirement.

The parameters I changed in order to achieve good performance were:

* N\_neighbors- Number of neighbors to use.
* weights: weight function used in prediction, Possible values:
  + ‘uniform’: uniform weights. All points in each neighborhood are weighted equally.
  + ‘distance’: weight points by the inverse of their distance. in this case, closer neighbors of a query point will have a greater influence than neighbors which are further away.

I used 15 neighbors, as you can see in the below graph it is maximizing the model performance.

Also setting weights="distance" or “uniform” have the same results.

|  |  |
| --- | --- |
| weights="distance" | weights="uniform" |
|  |  |

**K nearest neighbors Results:**

balanced dataset gives us accuracy of (test size:0.2)**:**

|  |  |  |
| --- | --- | --- |
| neighbors | weights | Accuracy |
| 15 | distance | 90.21% |

### SVM

*A Support Vector Machine (SVM) is a classifier formally defined by a separating hyperplane. In other words, given labeled training data (*supervised learning*), the algorithm outputs an optimal hyperplane which categorizes new examples. In two-dimensional space this hyperplane is a line dividing a plane in two parts where in each class lay in either side.*

I used SVM neighbors from scikit-learn and modified it per my requirement. I chose kernel 'rbf'. With max iteration 5000

**SVM Results:**

balanced dataset gives us accuracy of (test size:0.2):

|  |  |  |
| --- | --- | --- |
| max iteration | kernel | Accuracy |
| 5000 | rbf | 85.7% |

### Decision Tree

*A decision tree is a*[*decision support*](https://en.wikipedia.org/wiki/Decision_support_system)*tool that uses a*[*tree-like*](https://en.wikipedia.org/wiki/Tree_(graph_theory))[*model*](https://en.wikipedia.org/wiki/Causal_model)*of decisions and their possible consequences.*

I used Decision Tree from scikit-learn and modified it per my requirement.

The features evaluation is built in function.

The parameters I changed in order to achieve good performance were:

* + - * Criterion- The function to measure the quality of a split. Supported criteria are “gini” for the Gini impurity and “entropy” for the information gain.
      * max\_depth- The maximum depth of the tree. If None, then nodes are expanded until all leaves are pure or until all leaves contain less than min\_samples\_split samples.
      * min\_samples\_split- The minimum number of samples required to split an internal node

I used max\_depth= 8 and min\_samples\_split=2, in order to fully develop the decision tree.

There was no significant impact for changing the criterion. Gini criterion and Entropy criterion gave almost same accuracy.

|  |  |
| --- | --- |
| Criterion ="entropy" | Criterion ="gini" |
|  |  |

**Decision Tree Results:**

balanced dataset gives us accuracy of (test size:0.2)**:**

|  |  |  |  |
| --- | --- | --- | --- |
| min\_samples\_split | criterion | max depth | Accuracy |
| 2 | entropy | 8 | 85.51% |

**Features importance**

The average model gave the following list of feature importance:

|  |
| --- |
| entropy |
|  |

### Random Forest

[*Random forest*](https://www.sciencedirect.com/topics/computer-science/random-decision-forest)*is a classifier that evolves from*[*decision trees*](https://www.sciencedirect.com/topics/computer-science/decision-trees)*. It consists of many decision trees. To classify a new instance, each decision tree provides a classification for input data;*[*random forest*](https://www.sciencedirect.com/topics/computer-science/random-decision-forest)*collects the classifications and chooses the most voted prediction as the result. The input of each tree is sampled data from the original*[*dataset*](https://www.sciencedirect.com/topics/engineering/dataset)*. In addition, a subset of features is randomly selected from the optional features to grow the tree at each node. Each tree is grown without pruning. Essentially, random forest enables a large number of weak or weakly-correlated classifiers to form a strong classifier.*

I used Random forest from scikit-learn and modified it per my requirement.

The features evaluation is built in function.

The parameters I changed in order to achieve good performance were:

* + - * n\_estimators- The number of trees in the forest.
      * max\_features- The number of features to consider when looking for the best split.
      * max\_depth- The maximum depth of the tree. If None, then nodes are expanded until all leaves are pure or until all leaves contain less than min\_samples\_split samples.
      * min\_samples\_split- The minimum number of samples required to split an internal node

I used max\_depth= 8 and min\_samples\_split=2, in order to fully develop each tree in the forest.

Best performance achieved when number of features to use at each tree was 17 as you can see in the below graph.

|  |
| --- |
| Check best n estimators |
|  |

|  |
| --- |
| Check max features to use |
|  |

**Random Forest Results:**

balanced dataset gives us accuracy of(test size:0.2)**:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| min\_samples\_split | max features | n estimators | max depth | Accuracy |
| 2 | 17 | 800 | 8 | 90.79% |

**Features importance:**

The average model gave the following list of feature importance:

|  |
| --- |
| Features importance: |
|  |

### Bagging

*Special case of Random Forest, all features are considered for splitting a node.*

Since Bagging is a private case of Random Forest, I used Random forest as I described at previous section, the only change I made is to set max\_features ="None" which will use all features for all trees.

**Bagging Results:**

balanced dataset gives us accuracy of (test size:0.2)**:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| min\_samples\_split | max features | n estimators | max depth | Accuracy |
| 2 | None | 800 | None | 89.85% |

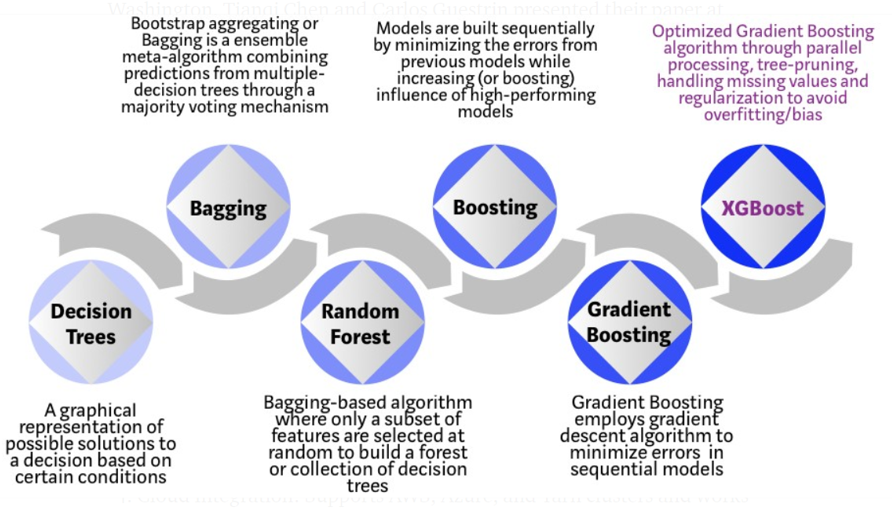
**Features importance:**

The average model gave the following list of feature importance:

|  |
| --- |
|  |
|  |

**Boosting Methods:**

*Boosting is a method of converting a set of weak learners into strong learners.* *A weak learner is defined to be a*[*classifier*](https://en.wikipedia.org/wiki/Classification_(machine_learning))*that is only slightly correlated with the true classification (it can label examples better than random guessing). In contrast, a strong learner is a classifier that is arbitrarily well-correlated with the true classification. To convert a weak learner into strong learner, we take a family of weak learners, combine them and vote. This turns this family of weak learners into strong learners.*



### AdaBoost

*An AdaBoost classifier is a meta-estimator that begins by fitting a classifier on the original dataset and then fits additional copies of the classifier on the same dataset but where the weights of incorrectly classified instances are adjusted such that subsequent classifiers focus more on difficult cases.*

I used AdaBoost from scikit-learn and modified it per my requirement.

The features evaluation is built in function (for trees models).

The parameters I changed in order to achieve good performance were:

* + - * base\_estimator: It is a weak learner used to train the model. It uses DecisionTreeClassifier as default weak learner for training purpose. You can also specify different machine learning algorithms.
      * n\_estimators: Number of weak learners to train iteratively. In case of perfect fit, the learning procedure is stopped early.
      * learning\_rate: Learning rate shrinks the contribution of each classifier by learning\_rate. There is a trade-off between learning\_rate and n\_estimators.

I used base\_estimator = Decision Tree (max\_depth=None, criterion=entropy), and learning\_rate=0.3

|  |
| --- |
| Check best n estimators |
|  |

|  |
| --- |
| Check best learning rate |
|  |

**AdaBoost Results:**

balanced dataset gives us accuracy of (test size:0.2)**:**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| base estimator | min\_samples\_split | criterion | max depth | n estimators | learning rate | Accuracy |
| DT | 2 | entropy | None | 250 | 0.3 | 88.92% |

**Features importance:**

The average model gave the following list of feature importance:

|  |
| --- |
|  |
|  |

### GBoost (Gradient Boosting)

*The basic concept of Gradient Boosting remains the same as AdaBoost, except here we don’t play with the weights, but fit the model on residuals (measurement of the difference in prediction and original outcome) rather than original outcomes. AdaBoost is implemented using iteratively refined sample weights while Gradient Boosting uses an internal regression model trained iteratively on the residuals. This means that the new weak learners are formed keeping in mind the inputs that have high residuals.*

I used AdaBoost from scikit-learn and modified it per my requirement.

The features evaluation is built in function (for trees models).

The parameters I changed in order to achieve good performance were:

* + - * learning\_rate: learning rate shrinks the contribution of each tree by learning\_rate. There is a trade-off between learning\_rate and n\_estimators.
      * n\_estimators: The number of boosting stages to perform. Gradient boosting is fairly robust to over-fitting so a large number usually results in better performance.
      * min\_samples\_split: The minimum number of samples required to split an internal node.
      * max\_depth: The maximum depth limits the number of nodes in the tree.
      * max\_features: The number of features to consider when looking for the best split.

I used max\_depth=7 due to time constrains.

I used n\_estimators = 100

|  |
| --- |
| n estimators graph |
|  |

Best performance achieved when learning rate was 0.1 for both hypotheses as you can see in the below graphs.

|  |
| --- |
| Learning rate graph |
|  |

**GBoost Results:**

balanced dataset gives us accuracy of (test size:0.2)**:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| learning rate | n estimators | min\_samples\_split | max depth | max features | Accuracy |
| 0.1 | 100 | 2 | 7 | None | 90.99% |

**Features importance:**

The average model gave the following list of feature importance:

|  |
| --- |
| Features importance: |
|  |

### XGBoost (Extreme Gradient Boosting)

*Extreme Gradient Boosting it is a specific implementation of the Gradient Boosting method which uses more accurate approximations to find the best tree model. It employs several nifty tricks that make it exceptionally successful, particularly with structured data. The most important are*

* *computing****second-order gradients, i.e. second partial derivatives****of the loss function, which provides more information about the direction of gradients and how to get to the minimum of our loss function. While regular gradient boosting uses the loss function of our base model as a proxy for minimizing the error of the overall model.*
* *advanced****regularization****which improves model generalization.*
* *additional advantages: training is very fast and can be parallelized / distributed across clusters.*

I used XGBoost software library and modified it per my requirement.

The features evaluation is built in function

The parameters I changed in order to achieve good performance were:

* + - * learning\_rate: learning rate shrinks the contribution of each tree by learning\_rate. There is a trade-off between learning\_rate and n\_estimators.
      * n\_estimators: The number of boosting stages to perform. Gradient boosting is fairly robust to over-fitting so a large number usually results in better performance.
      * max\_depth: The maximum depth limits the number of nodes in the tree.
      * max\_features: The number of features to consider when looking for the best split.

I used max\_depth=7 due to time constrains.

I used n\_estimators = 100

|  |
| --- |
| N estimators |
|  |

Best performance achieved when learning rate was 0.3

|  |
| --- |
| Learning rate |
|  |

**XGBoost Results:**

balanced dataset gives us accuracy of (test size:0.2)**:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| learning rate | n estimators | max depth | max features | Accuracy |
| 0.1 | 100 | 7 | None | 91% |

**Features importance:**

The average model gave the following list of feature importance:

|  |
| --- |
|  |
|  |

### Voting

*Voting is combining the predictions from multiple machine learning algorithms. Voting classifier is a wrapper for set of different ones that are trained and valuated in parallel in order to exploit the different peculiarities of each algorithm.*

**Voting Results:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| first model | second model | third Model | Voting | Accuracy |
| KNN (90.35%) | DT (85.55%) | XGBoost (90.96%) | hard | 90.58% |
| KNN (90.35%) | RF (90.81%) | XGBoost (90.85%) | hard | 90.83% |
| KNN (90.14%) | SVM (82.07%) | XGBoost (90.71%) | hard | 90.42% |
| AdaBoost (89.52%) | Gboost (90.68%) | XGBoost (90.78%) | hard | 90.76% |
| RF (90.84%) | AdaBoost (89.67%) | XGBoost (90.80%) | hard | 90.81% |

# Results

* My models and performance

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | KNN | SVM | Decision Tree | Random Forest | Bagging | AdaBoost | Gboost | XGBoost | Voting |
|  | 90.21% | 85.7% | 85.51% | 90.79% | 89.85% | 88.92% | 90.99% | 91% | 90.81% |

* + The model results will appear in the command screen.

Also 2 files will be created automatically in results folder:

* + - Txt file named *"log\_<Model\_Name>\_ for\_heart\_disease\_prediction.txt*"

(for example: log\_XGBoost\_Test1VSTest2.txt)

This txt file contains:

* + - 1. General Information

Number of observations and test size

* + - 1. Preprocessing -Feature Selection

The features that were removed before learning, and does not contribute to the learning process

* + - 1. Model Information

Model name and model parameters

* + - 1. Model Results

Classification Report, confusion matrix and model accuracy

* + - 1. Features Importance

Ranked from high to low

* + - csv file named *"predictedResults <Model\_Name>\_for\_heart\_disease\_prediction.csv"*
    - For Decision tree model png file will be created for tree visualization.

**Feature escalation**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | DT | RF | Bagging | AdaBoost | Gboost | XGBoost | avg |
| Age | 0.11548 | 0.211864 | 0.095577 | 0.165186 | 0.197886 | 0.105701 | 0.148614833 |
| AnyHealthcare | 0.00734 | 0.001006 | 0.007949 | 0.008722 | 0.002649 | 0.011658 | 0.006553833 |
| BMI | 0.18819 | 0.018918 | 0.188197 | 0.183119 | 0.042816 | 0.010856 | 0.105348833 |
| CholCheck | 0.00402 | 0.000952 | 0.004122 | 0.00906 | 0.003347 | 0.018852 | 0.006726 |
| Diabetes | 0.0255 | 0.018085 | 0.031824 | 0.024895 | 0.025867 | 0.021246 | 0.024568833 |
| DiffWalk | 0.02022 | 0.047392 | 0.023025 | 0.016554 | 0.047573 | 0.047908 | 0.033778 |
| Education | 0.06954 | 0.006303 | 0.074805 | 0.05905 | 0.017297 | 0.011381 | 0.039729333 |
| Fruits | 0.0341 | 0.001703 | 0.036302 | 0.029251 | 0.005368 | 0.01048 | 0.019533833 |
| GenHlth | 0.10226 | 0.331476 | 0.083981 | 0.077792 | 0.282106 | 0.207372 | 0.180830667 |
| HighBP | 0.0253 | 0.073534 | 0.019869 | 0.026923 | 0.051893 | 0.234321 | 0.071973 |
| HighChol | 0.01817 | 0.06062 | 0.017126 | 0.023658 | 0.055375 | 0.064402 | 0.039892 |
| HvyAlcoholConsump | 0.00954 | 0.000826 | 0.010348 | 0.009707 | 0.002901 | 0.016002 | 0.0082205 |
| Income | 0.10257 | 0.010888 | 0.108206 | 0.092202 | 0.025497 | 0.01446 | 0.058971167 |
| MentHlth | 0.06298 | 0.007512 | 0.066015 | 0.073935 | 0.022821 | 0.011033 | 0.040715333 |
| NoDocbcCost | 0.0124 | 0.004033 | 0.014909 | 0.011641 | 0.00679 | 0.012211 | 0.010330167 |
| PhysActivity | 0.02938 | 0.001559 | 0.031693 | 0.023509 | 0.004104 | 0.010596 | 0.0168065 |
| PhysHlth | 0.08309 | 0.010794 | 0.087104 | 0.065929 | 0.025363 | 0.012714 | 0.047498333 |
| Sex | 0.021 | 0.071852 | 0.019721 | 0.027712 | 0.06325 | 0.07047 | 0.045667167 |
| Smoker | 0.02484 | 0.010264 | 0.028222 | 0.034887 | 0.019395 | 0.027672 | 0.0242125 |
| Stroke | 0.01821 | 0.108825 | 0.022535 | 0.015053 | 0.092007 | 0.070961 | 0.054597833 |
| Veggies | 0.02591 | 0.001596 | 0.02847 | 0.021215 | 0.005695 | 0.009704 | 0.015431167 |