ENSEMBLE METHODS

Satyajeet Awati

Submitted for the Degree of Master of Science in

Data Science and Analytics

Department of Computer Science
Royal Holloway University of London
Egham, Surrey TW20 0EX, UK

September 06, 2023

Declaration

This report has been prepared on the basis of my own work. Where other published and unpublished source materials have been used, these have been acknowledged.

Word Count: 11,109

Student Name: Satyajeet Awati

Date of Submission: 06/09/2023

Signature:

Abstract

In this study we are going to explore ensemble methods, a powerful techniques in machine learning that helps to make predictions more accurately. As we all know the main goal of machine learning is to build algorithms or models that can learn the patterns, relationships from data and use it to make accurate predictions on unseen or new data. This works very well until regular models doesn't meet the complicated or noisy data. But ensemble methods can deal with this type of data more efficiently. Ensemble methods help with this problem by combining the predictions of many regular models to make a better decision.

We started this study by looking at what other researchers and practitioners have done in this area. Then, we explained how ensemble methods work and why they are useful. In this study we have tested ensemble methods on different machine learning algorithms to see how well they perform. By looking at performance difference we found that ensemble methods can handle complex data more efficiently.

In this study we also looked at different ways ensemble methods can be used and how they can be combined with other regular machine learning techniques to make predictions even better. We have also introduces some real-world examples to show how ensemble methods can be used in practical situations. We found those real-world examples from some references to support our study.

Overall, this study gives us a better understanding of ensemble methods like bagging, boosting and shows how they can improve regular machine learning models. This will help researchers and practitioners create more reliable and effective solutions for different data problems and handle complicated and noisy data well.

Contents

1 INTRODUCTION

Machine learning is sub-part of Artificial Intelligence (AI), and this part focuses on development and enhancement of already developed algorithms and models those who allow computers to learn and give predictions without being explicitly programmed to do particular task. Machine learning is continuously evolving, and with help of it researchers try to explore innovative methodologies to achieve better and better predictive accuracy and robustness. Ensemble methods in machine learning, is arisen as very powerful technique to improve the performance by combining multiple individual models into a more predictive models. In this report we are going to explore some of ensemble methods, explaining how they work, their main techniques, benefits, and real-life uses.

1.1 Background

Machine learning is about creating models that can learn from provided data and provide us with predictions on data they haven't seen before. It involves use of data to improve the performance of a system by learning from data. The real core idea is to make computers to identify patterns, relationships in the provided data and use that insights to make predictions or decisions. Machine learning is mainly divided into two types supervised and unsupervised machine learning.

In supervised machine learning, the algorithm is trained on labelled datasets, where each input is attached with corresponding target value or output. Main goal of this type is to learn to map inputs to outputs ad allow algorithm to make predictions on new data. It involves types like classification and regression. In classification task algorithm have to predict from which class the new data input belongs to, it's like email is spam or not spam. In regression task algorithm produce continues output for the given input, it's like weather prediction.

In unsupervised machine learning, dataset doesn't contain any labels for data. Algorithm have to learn discover patterns, structures and relationships with the data provided. It involves tasks like clustering and dimensionality reduction.

For our project on ensemble methods we are mainly focusing on classification supervised learning methods. But some time the data which is used may have some noise or some blank spaces, so it may influence the models predictive accuracy. That's where ensemble methods come in to help. They try to handle this problem by combining more than one model together for producing better accuracy. For example suppose we have a hard question and we ask that question to a group of people, so that each person's answer may not be whole correct but together they can do better. Ensemble methods do the same thing. Instead of relying on only one model, ensemble methods collect the predictions from multiple models and use their combined predictions to make a final prediction. By using this technique ensemble methods can outperform the traditional single model and deals with bad data more effectively.

1.2 Objectives

- Data pre-processing
- Understand ensemble techniques
- Ensemble methods using various machine learning algorithms
- Comparing various Machine learning models
- Improve predictive accuracy
- Provide Valuable insights for the decision makers like stakeholders, clients

1.3 Dissertation Questions

1. What are the theoretical foundations and underlying principles of popular ensemble techniques such as bagging, boosting?
2. What are the advantages and limitations of using ensemble methods?
3. How can ensemble methods be combined with other machine learning techniques to achieve even better predictive accuracy?
4. How do different diversity measures impact the performance of ensemble methods?
5. How do ensemble methods compare to other machine learning algorithms in terms of performance and practicality?

1.4 Dissertation Outline

Section 2 puts light on a literature review, giving the overview of ensemble methods and the various machine learning algorithms used for it. Also the overview of relevant features, limitations and the existing study.

Section 3 puts light on the methodology used in this dissertation such as Data Pre-processing, Model Selection, Training and Testing of model, Model evaluation, and optimization.

Section 4 puts light on the Datasets used in the dissertation by giving information of the data sources.

Section 5 puts light on the results and analysis of the developed models, including performance metrics, Comparison of the models and selection of the best model.

Section 6 puts light on professional issues we faced and studied.

Section 7 concludes the dissertation by summarizing the research objectives, key findings and final remarks.

Section 8 provides with user manual to handle project.

At last, it has the references used to support the research for this dissertation.

1.5 Key Techniques in Ensemble Methods (Used In Dissertation):

Bagging:

Bagging stands for Bootstrap Aggregating, is an ensemble machine learning technique used to improve accuracy of traditional predictive models. The main idea behind bagging is to train multiple independent traditional models on the different parts on training data, by bootstrapping the original training data. Then these all models work together and with help of majority voting we can decide the combined prediction, which is more stable and accurate. It's like having selected teammates to give their opinions and finding a balanced answer.

Boosting:

Boosting is like learning from mistakes. It starts with a simple weak learner. It checks where learner goes wrong, and then adds more learners that fix those mistakes. For this project we are using AdaBoost boosting model, which stands for Adaptive Boosting. As these learners getting better every time they can easily tackle tricky data very efficiently. It's a bit like getting better at a game by learning from each round's mistakes.

2 LITERATURE REVIEW

2.1 Overview

Ensemble methods are gaining significant attention in the field machine learning due to their ability to improve the predictive accuracy and generalization performance. This literature review aims to provide a broad overview of ensemble methods, their theoretical foundations, their advantages, their limitations, and practical applications/real time examples.

2.2 Ensemble Methods and Bagging

In machine learning, ensemble methods have brought a big change to how we use machine learning by making predictions better and more reliable. One of the popular ensemble methods called Bagging has been a key part of this change. In this review, we will explore the basic ideas behind bagging ensemble methods, look at how they work in real-world situations, and see how they are explored by researches and practitioners.

Theoretical Foundations and Advantages

Ensemble methods mainly work on the idea that if we bring together the predictions of many models, we can make better, more trustworthy and robust predictions. This is kind of like a situation when a group of people with different perspectives combining makes a better decision than just a single person with only one perspective. One of the first and most important ensemble methods is bagging, which was created by Breiman in 1996. Bagging works like training many models on different parts of the data and then putting all their predictions together, usually by averaging or voting [1]. This makes the predictions more stable, accurate and reliable by decreasing the risk of big mistakes and complex data problems.

Ensemble methods, including Bagging, offer several advantages:

- Ensemble methods make predictions better.
- They're better at not making predictions that are too specific to the training data.
- They can handle weird or outlier data points better by looking at lots of models' opinions.
- They're good at understanding tricky data with lots of details.
- Using different types of models together helps the ensemble methods better at making predictions.

Bagging: Performance and Variants

Many real-world tests have proven that Bagging works really well. In one study, Caruana and others compared Bagging to single models and found that Bagging gave more accurate results for lots of different types of data and problems [2]. This showed that using Bagging could make regular models better. Another study by Freund and Schapire made Bagging even better by selecting only some features for each model, making the whole ensemble method more powerful [3]. This is called Feature Bagging or Random Subspace Method, and it makes the ensemble do even better.

Practical Applications of Bagging

- Medical Diagnosis: Bagging has been used to predict disease outcomes and classify medical images [4].
- Finance: In the financial sector, Bagging has been used for credit risk assessment and stock market prediction [5].
- Natural Language Processing: Bagging has been effective in sentiment analysis and spam email detection [5].

Challenges and Future Directions

Even though Bagging is really helpful, there are still some difficulties or limitations. Understanding of the combined predictions of many models can be tricky sometimes. Also, deciding how many models we need to use and how much time it will take to compute can be a tough choice and also it will take lot of experiments to know perfect parameters for particular dataset. In the future, researchers might try mixing Bagging with other methods or using it along with more advanced ways of doing machine learning.

2.3 AdaBoost Ensemble Method

Ensemble methods are becoming really very important for making machine learning models more accurate, strong and reliable. One of these ensemble methods called as AdaBoost (short-form for Adaptive Boosting), is very important. In this review we will give you a simple understanding of what AdaBoost is all about – its basic ideas, how well it works in real situations, and how people use it practically.

Theoretical Foundations of AdaBoost

AdaBoost is a very smart method invented by Freund and Schapire in 1997. AdaBoost is like a coach who is training a team of players who are individually good at performance. The coach trains them one by one, by paying more attention to the mistakes they are making and try to improve them every time. Then coach combines all their extra efforts to make a strong team that does really well [3]. Similarly AdaBoost provides more importance to the players who make mistakes so that they can get better with each round of training. This way, the model becomes really good at making accurate predictions.

Performance of AdaBoost

People have looked at how well AdaBoost works on many different types of data and problems. Freund and Schapire showed that AdaBoost can turn weak models into a strong once, even if those models are not very good on their own [3]. Another study by Drucker and his team found that AdaBoost usually did better than other methods in tasks where you have to decide between two choices [6]. AdaBoost is good at handling messy data and understanding complicated patterns, which is why it does so well.

Practical Applications of AdaBoost

- Face Detection: AdaBoost has been widely utilized in computer vision, particularly for face detection in images and videos [7].
- Bioinformatics: The method has been applied to protein structure prediction and DNA sequence analysis [8].
- Natural Language Processing: AdaBoost has been employed for sentiment analysis, part-of-speech tagging, and text categorization [9].

Challenges and Future Directions

Even though AdaBoost works really very well, it can sometimes struggle with messy data or unusual points that don't fit well with other data points. This might make it focus too much on those points and leads to not work as well as it can. Scientists are trying to find ways to fix this problem. They're looking into using methods that are better at handling these issues or changing how AdaBoost works to deal with problems where there are more than two choices to pick from otherwise it works well when there are only two choices.

2.4 K-Nearest Neighbors (k-NN) Algorithm

The k-Nearest Neighbors (k-NN) algorithm is a basic algorithm but it’s strong model used in machine learning for deciding if something belongs to a particular group or class. This review is here to give you a simple understanding of what k-NN is all about – how it's built on ideas, where it's used in real life, and how people have twisted it to work better in different situations.

Theoretical Foundations of k-NN

The k-Nearest Neighbors (k-NN) algorithm works on idea that things those who are alike in some ways are probably in from same group. Imagine if you have a new thing, and you want to know which group it belongs to. The k-NN algorithm looks at the k points in the training data that are most similar to the new point, and based on what those k things are, it decides which group the new thing should be in. The choice of k and distance matric (such as Euclidean) significantly affect the algorithm’s performance.

Performance of k-NN

Lots of tests have been done to see how well the k-NN algorithm works. Even though it's quite simple, k-NN has shown that it can do really well on different types of data and problems. Hastie et al. showed that k-NN can approximate complex decision boundaries and achieve high accuracy in both classification and regression tasks [10]. This makes k-NN good at getting things right when the data doesn't follow a straight path.

Practical Applications of k-NN

- Recommender Systems: k-NN has been worked for personalized recommendations in e-commerce and content platforms [10].
- Image Classification: k-NN is used for image recognition tasks, where the feature space captures image characteristics [11].
- Anomaly Detection: In cyber-security, k-NN aids in identifying anomalies in network traffic or user behaviour [12].

Challenges and Variations

The k-NN algorithm have its own challenges or limitations we can say. The performance of this model is heavily depend on appropriate k value selection and effective distance matrices. Because of this reason noise and irrelevant features can affect its accuracy. Researchers have proposed variations, such as weighted k-NN, which assigns different weights to neighbouring instances based on their distances or relevance.

2.5 Logistic Regression

Logistic regression is a very important algorithm in machine learning. This review is here to give you a simple understanding of what logistic regression is all about – the basic ideas behind it, how well it works in real situations, and where people use it in the real world.

Theoretical Foundations of Logistic Regression

We can say Logistic regression is a way to decide between two choices. Imagine we have to say "yes" or "no" to something. It's like using math to guess how probable it is for that thing to be a "yes." It's like how regular math tries to guess a number, but here we're guessing how likely something is. We use a special curve to make sure our guess is between 0% and 100%. So, logistic regression helps us say how likely it is for something to be a "yes" or a "no."

Performance of Logistic Regression

Studies have demonstrated the flexibility and effectiveness of logistic regression across various domains. Hosmer and Lemeshow conducted extensive research on logistic regression's goodness-of-fit and its ability to predict binary outcomes accurately [13]. Logistic regression often performs well when the relationship between predictors and the response variable is roughly linear.

Practical Applications of Logistic Regression

- Medical Diagnosis: It's used to predict disease outcomes, identify risk factors, and assess patient health conditions [14].
- Marketing: Logistic regression aids in customer segmentation, churn prediction, and targeted marketing campaigns [15].
- Credit Scoring: The technique is used for credit risk assessment, determining the likelihood of loan default [16].

Extensions and Challenges

Logistic regression is made to work with more than two choices and deal with different kind of information. So this idea can be possible if we do it like one-vs-rest and changing how can represent the data. But there are some difficulties like it assumes by itself things change in a straight line, it can be overly affected by odd data, or sometimes it happen like information we offer is kind of too similar. Scientists and practitioners are looking into ways to fix these problems, like using special math tricks to make it work better.

2.6 Bagging and k-Nearest Neighbors (K-NN)

The combination of ensemble techniques like Bagging with the k-Nearest Neighbors (k-NN) algorithm represents a convincing method to improve predictive accuracy. This literature review aims to explore the theoretical underpinnings, empirical evidence, and potential applications of this hybrid approach.

Theoretical Foundations of Bagging and KNN

Bagging (Bootstrap Aggregating) involves training multiple models on bootstrapped samples and aggregating their predictions, reducing variance and improving stability [1]. KNN, on the other hand, makes predictions based on the majority class among the k-nearest neighbors in the feature space [17]. Combining these methods pursues to bond the strengths of both, potentially reducing the weaknesses of each.

Performance of Bagging with KNN

When earlier people tried mixing Bagging and KNN together, they found some good signs. This mix uses bagging to stop models from getting too specific details and KNN to understand tricky data. Some tests done by Li and the team showed that when they put Bagging and KNN together, predictions got better on different types of data [18]. This mix worked really well, especially when dealing with messy data or when there's not the same number of things in different groups.

Practical Applications of Bagging and KNN

- Medical Diagnosis: The hybrid approach can aid in more accurate disease prediction and patient risk assessment [33].
- Image Classification: By combining Bagging with KNN, image recognition tasks can achieve improved accuracy and robustness [19].
- Anomaly Detection: In cyber-security, the hybrid technique can be effective in identifying unusual patterns in network traffic [20].

Challenges and Future Directions

Even though mixing Bagging and KNN seems helpful and giving us more benefits, there are some difficulties. Figuring out the best probable settings for how they work together is a bit tricky, along with that making sure they work well without taking too much time to calculate is an important point. In the future, scientists might look more into how many models to use, ways to measure distances and how many things to compare (k-value) to make this mix work the best.

2.7 Bagging with Logistic Regression

Bagging with Logistic Regression, this teamwork can provide us very healthy results by making predictions better and models stronger.in this review we will dig into the basic ideas, real-world tests, and where this mix of methods could be used.

Theoretical Foundations of Bagging and Logistic Regression

Bagging (Bootstrap Aggregating) involves training multiple models on bootstrapped samples and combining their predictions to improve stability and reduce variance [1]. Logistic Regression, on the other hand, models the log-odds of the probability for binary classification tasks using a linear relationship between features and the log-odds of the outcome [13]. When we put these methods together, we hope to use the best parts of each to fix their individual weaknesses.

Performance of Bagging with Logistic Regression

When people tried mixing Bagging and Logistic Regression, they found strong new model. This mix uses Bagging to stop models from getting too specific and Logistic Regression to make things easier to understand. Some tests done by Wang and the team showed that when they put Bagging and Logistic Regression together, predictions got better on different types of data [21]. This mix worked really well, especially when dealing with messy data or odd points in the data.

Practical Applications of Bagging and Logistic Regression

- Credit Scoring: The hybrid approach can lead to improved credit risk assessment and more accurate prediction of loan defaults [22].
- Healthcare: Combining Bagging and Logistic Regression can enhance disease outcome prediction and patient risk assessment [23].
- Marketing Analytics: The hybrid technique can contribute to better customer churn prediction and targeted marketing strategies [24].

Challenges and Future Directions

Even though mixing Bagging and Logistic Regression looks very powerful, there are some difficulties we can’t ignore. Figuring out the best settings for how they work together is an important task. Choosing how many models to use in the mix and setting things just at right proportion is difficult. Also, making sure the features we use don't cause confusion is important too. In the future, scientists might try different ways of putting together models along with Logistic Regression to see what works even better.

2.8 Decision Tree Algorithm

Decision tree is a very powerful and flexible machine learning algorithm that can provide a transparent representation of decision processes. In this review of existing literature seeks to offer understanding regarding the fundamental principles, real-world use cases, and progressions within decision tree algorithms.

Theoretical Foundations of Decision Trees

The main idea behind the decision trees is to divide the data into smaller groups according to the characteristics of input features. This provides hierarchical representation of data division. This hierarchical representation consists of nodes, wherein internal nodes represent decisions based on specific features and terminal nodes indicate predicted results. Constructing the decision trees involves the identification of best features and criteria for dividing the data. With the objective of maximizing measures like information gain, impurity, or other suitable metrics.

Performance of Decision Trees

Practical research have highlighted the efficacy of decision trees across various fields. Decision trees shine in grasping complex connections, managing varied data types, and offer us models which are easily understandable. The foundational concept of constructing decision trees through recursive binary splitting was introduced by Breiman et al.'s research on Classification and Regression Trees [25]. Decision trees have showcased strong performance in both classifying and predicting outcomes.

Practical Applications of Decision Trees

- Healthcare: They aid in disease diagnosis, patient risk assessment, and treatment recommendation based on patient attributes [26].
- Finance: Decision trees are used for credit risk assessment, fraud detection, and stock price prediction [31].
- Customer Relationship Management: They assist in customer segmentation, churn prediction, and personalized marketing strategies [32].

Advancements and Challenges

Progress in decision tree algorithms involves ensemble techniques such as Random Forests, Boosting which combine multiple decision trees to enhance accuracy and flexibility. Some obstacles include like addressing over-fitting, managing imbalanced data, and effectively handling datasets with high dimensions. Researchers are actively investigating strategies like pruning and regularization to tackle these difficulties.

2.9 AdaBoost with Decision Tree

The integration of the AdaBoost ensemble technique with Decision Trees presents an approach to improving classification and regression tasks. In this review we will dig into the basic ideas, real-world tests, and where this mix of methods could be used.

Theoretical Foundations of AdaBoost and Decision Trees

AdaBoost (Adaptive Boosting) is a teamwork method that aims to make weak learners better by training them repeatedly and giving more attention to cases they struggle with [3]. On the other hand, Decision Trees are models that divide the data into sections, allowing for complex decisions [25]. Combining AdaBoost with Decision Trees takes advantage of both method's strengths.

Performance of AdaBoost with Decision Trees

Studies performed on merging AdaBoost and Decision Trees have shown that it really helps in improving accuracy. This hybrid method uses AdaBoost to make models better and Decision Trees to understand complicated data patterns. Drucker and his team found that putting AdaBoost and Decision Trees together was better at classifying things than just using Decision Trees on their own. This teamwork was most helpful for datasets where the groups are complex [6].

AdaBoost and Decision Trees finds applications in various domains

- Image Classification: The combination can lead to improved accuracy in image recognition tasks, where complex visual patterns are prevalent [7].
- Customer Churn Prediction: AdaBoost with Decision Trees can enhance the accuracy of predicting customer churn and designing retention strategies [26].
- Medical Diagnosis: The hybrid approach can contribute to more accurate disease prediction and patient risk assessment [27].

Challenges and Future Directions

Even though merging AdaBoost and Decision Trees is very beneficial, there are still difficulties to consider. It's important to adjust parameters carefully to get the best results from it. Deciding on how many models to use, and how fast they learn all impact how well they work. In the future, more studies could look into how different weak learners affect the results and how the number of models and their complexity are connected.

2.10 AdaBoost with Logistic Regression

The combination of AdaBoost with Logistic Regression offers a convincing approach to boost classification accuracy and strengthen model flexibility. In this review of existing literature seeks to explore into the underlying theories, practical observations, and possible real-world uses of this combined strategy.

Theoretical Foundations of AdaBoost and Logistic Regression

AdaBoost, which stands for Adaptive Boosting, is an iterative approach to train the models. This assigns a greater importance to samples that were misclassified in each successive iteration. On the other hand, Logistic Regression constructs models for binary classification tasks by forming a linear connection between features and the logarithm of the odds of the predicted outcome.

Performance of AdaBoost with Logistic Regression

Practical investigations into the fusion of AdaBoost and Logistic Regression have produced positive outcomes. This hybrid strategy leverages AdaBoost's potential to decrease model inconsistency along with the interpretability and directness of Logistic Regression. In-depth analyses conducted by Zhu et al. indicated that the integration of AdaBoost and Logistic Regression resulted in enhanced accuracy for classifying data and mitigated over-fitting concerns across a range of datasets [28].

Practical Applications of AdaBoost and Logistic Regression

- Medical Diagnosis: The hybrid technique can contribute to more accurate disease outcome prediction and patient risk assessment [29].
- Marketing Analytics: Combining AdaBoost with Logistic Regression enhances customer churn prediction and targeted marketing strategies [24].
- Fraud Detection: The hybrid approach can improve fraud detection by identifying anomalous patterns in transactions [30].

Challenges and Future Directions

Even if the fusion of AdaBoost and Logistic Regression shows positive potential, it does come with some challenges. The selection of less complex individual models and the size of the ensemble have remarkable effects on the performance. Dealing with related features and the careful choice of optimal model parameters are also important factors to consider. Researchers might investigate into exploring various approaches within ensemble techniques that include Logistic Regression along with various boosting algorithms.

3. METHODOLOGY

The methodology for ensemble methods combines multiple individual models to make more accurate predictions than the individual traditional model predicts. Below we will discuss the steps:

3.1 Select Base Models:

3.1.1 Selecting base models for Bagging Ensemble method

As our main goal is to make a model strong enough so it can perform better and provide better predictions. In bagging ensemble method, we create new small datasets (bags) by randomly choosing data from original dataset. So out of many traditional algorithm choices, we decided to choose k-Nearest Neighbors (k-NN) and Logistic Regression as base models for bagging ensemble method. We will further discuss reason behind choosing these two algorithms.

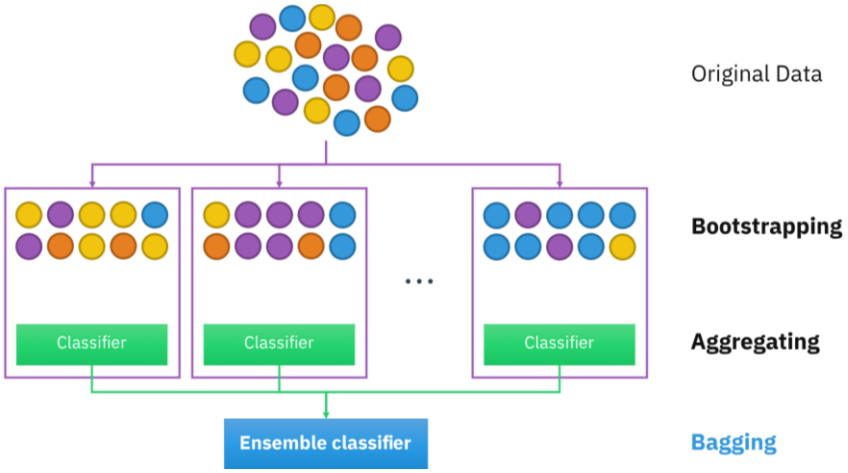


Image Credit: https://en.wikipedia.org/wiki/Bootstrap_aggregating

Bagging with k-Nearest Neighbors (k-NN):

k-Nearest Neighbors (k-NN) was chosen as one of our base models due to its ability to identify similarities within the data. Let's assume we are solving puzzle, and we want to know where we can fit the missing piece. K-NN works like looking at the pieces near to empty spot. This will help us to fit that missing piece even if the puzzle is hard. It's good at seeing small details that other methods might miss. Additionally, k-NN's simplicity goes well with the ensemble methodology, allowing it to contribute effectively to the aggregated predictions. I have built this algorithm from scratch as per project requirements.

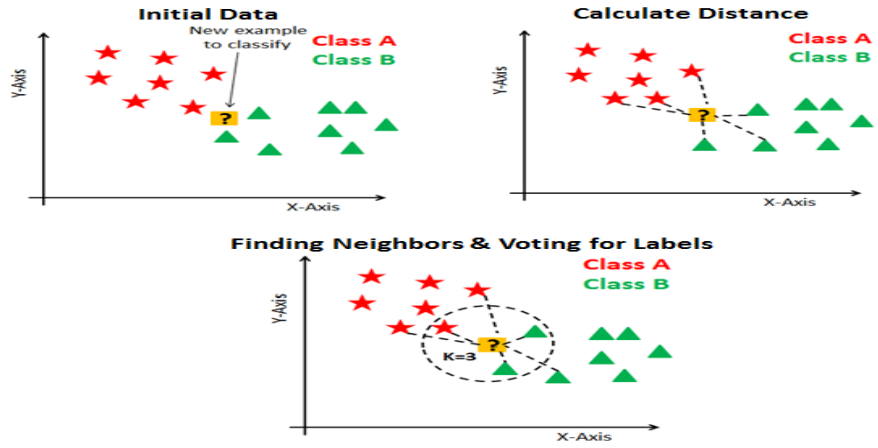
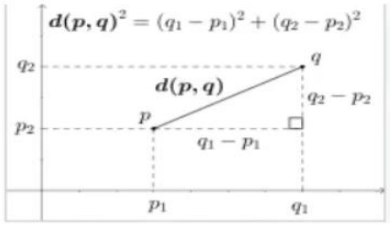


Image Credit: https://rekhavsrh.medium.com/all-about-knn-algorithm-6b35a18c2b15

Distance Formula:

Euclidean Distance.



Euclidean distance basically works on principal of **Pythagorean** theorem,

We are finding distance between point p and q.

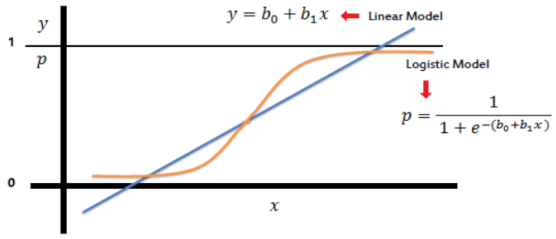
$$d(\mathbf{p}, \mathbf{q}) = \sqrt{(q_1 - p_1)^2 + (q_2 - p_2)^2}.$$

Image Credit: https://rekhavsrh.medium.com/all-about-knn-algorithm-6b35a18c2b15

Bagging with Logistic Regression:

We selected Logistic Regression as a second base model for our Bagging ensemble method. Logistic Regression is great at guessing if something is a yes or no. It's perfect for deciding between two choices. It's also easy to understand, Logistic Regression's skill in deciding chances and choices. This teamwork makes our ensemble even better at predicting capability.

On the other hand, a logistic regression produces a logistic curve, which is limited to values between 0 and 1. Logistic regression is similar to a linear regression, but the curve is constructed using the natural logarithm of the "odds" of the target variable, rather than the probability. Moreover, the predictors do not have to be normally distributed or have equal variance in each group.



In the logistic regression the constant (b_0) moves the curve left and right and the slope (b_1) defines the steepness of the curve. By simple transformation, the logistic regression equation can be written in terms of an odds ratio.

Image Credit: https://saedsayad.com/logistic_regression.htm

Synergy in Ensemble:
The combination of k-NN or Logistic Regression within the Bagging ensemble framework increases the strength of both models. k-NN sees complicated patterns, and Logistic Regression is easy to understand and predicts simply. The selection of k-Nearest Neighbors and Logistic Regression as base models for our Bagging ensemble is rooted in their respective strengths and compatibility.

3.1.2 Selecting base models for AdaBoost Ensemble method

We wanted to make our predictions better and create a strong model, so we thought a lot about which basic models to use for our AdaBoost. As we have discussed earlier logistic regression have good ability to choose between yes or no, so we decided to keep that one for Adaboost as well. And as a second base model we have chosen Decision tree classifier as it has ability to intricate relationships. As we wanted to use models strengths together, so we decided to include both the Decision Tree Classifier and Logistic Regression in our AdaBoost method.

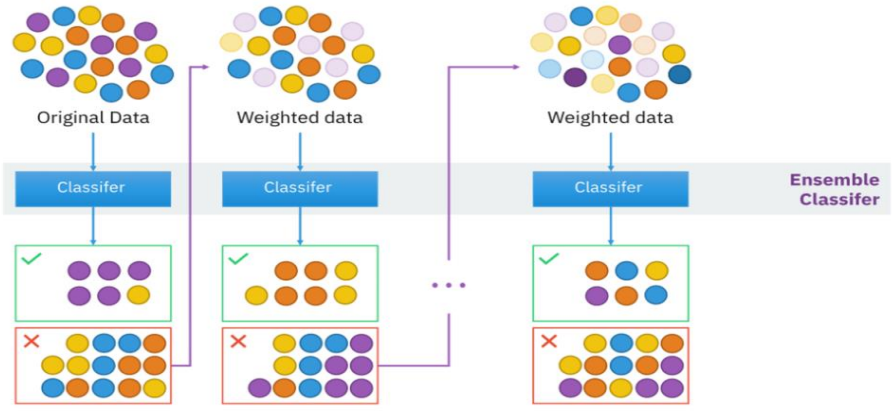


Image Credit: https://www.almabetter.com/bytes/tutorials/data-science/adaboost-algorithm

AdaBoost with Logistic Regression:
Our first choice for the ensemble base model is the Logistic Regression in our AdaBoost method. Logistic Regression perform better at guessing chances and choosing between two things, which matches well with what AdaBoost wants to do. By including Logistic Regression, we want to make the ensemble more clear and understandable, while getting benefits from its effective classification capabilities.

AdaBoost with Decision Tree Classifier:
Our second choice for the ensemble base model is the Decision Tree Classifier, integrated with the AdaBoost method. The Decision Tree's ability to capture intricate relationships within the data goes well with AdaBoost's iterative process. AdaBoost focuses on improving the accuracy of weak learners, and the Decision Tree's skill in dealing with complicated data patterns makes it a perfect partner.

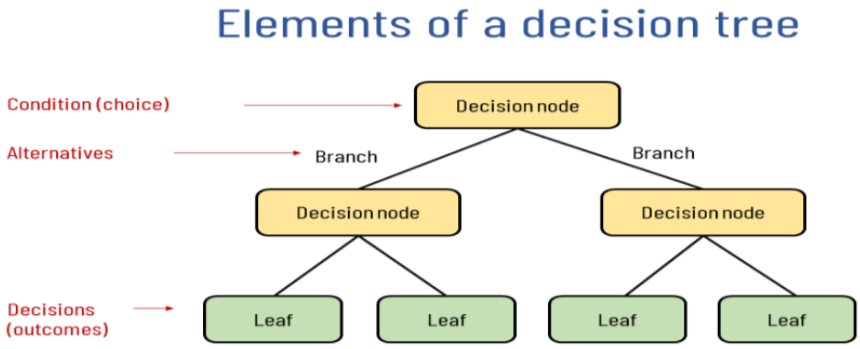


Image Credit: https://why-change.com/2021/11/13/how-to-create-decision-trees-for-business-rules-analysis/

Synergy of Models in AdaBoost:
Bringing the Decision Tree Classifier or Logistic Regression in AdaBoost shows that we're trying to use the best of the model. The Decision Tree is good at understanding small details, while Logistic Regression is good at giving simple answers. This mix is expected to make our ensemble method work really well with complicated data and give us better predictions.

3.2 Prepare Data

3.2.1 Data Cleaning:
Before using dataset for our main models we need to check that if there are any columns with empty values or there are columns which are exact similar with other columns in dataset. We looked closely at the dataset we picked and checked if we need to make any changes in it. After carefully checking, we found that everything was already good and nothing important to change but still we followed basic data cleaning process like dropna() for removing rows with missing values and drop_duplicates() for removing duplicates in our dataset. This makes me more confident that when we train our models and make predictions, the results will be accurate, dependable and non-biased.

Removing the rows with missing values and duplicates

```
In [3]: df = df.dropna()

In [4]: df = df.drop_duplicates()
```

Information about dataset and checking for null values

```
In [5]: print(df.columns)

Index(['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin',
       'BMI', 'DiabetesPedigreeFunction', 'Age', 'Outcome'],
      dtype='object')

In [6]: df.isnull().sum()

Out[6]: Pregnancies                 0
        Glucose                     0
        BloodPressure               0
        SkinThickness               0
        Insulin                     0
        BMI                         0
        DiabetesPedigreeFunction    0
        Age                         0
        Outcome                     0
        dtype: int64

In [7]: df.describe()

Out[7]:
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
count	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000
mean	3.845052	120.894531	69.105469	20.536458	79.799479	31.992578	0.471876	33.240885	0.348958
std	3.369578	31.972618	19.355807	15.952218	115.244002	7.884160	0.331329	11.760232	0.476951

3.2.2 Handling Imbalanced Data:

Our dataset contain only two labels 0 and 1 which represents either patient do have diabetes or don't. We checked the balance of our dataset with help of counting both labels, and we found that it's well distributed. This means we have about the good amount of data for each category we're interested in.

For "Pima Indians Diabetes" Dataset:

```
In [8]: class_distribution = df['Outcome'].value_counts()
        print(class_distribution)

0    500
1    268
Name: Outcome, dtype: int64
```

For "Heart Disease Detection" Dataset:

```
In [10]: class_distribution = df['target'].value_counts()
         print(class_distribution)

1    164
0    138
Name: target, dtype: int64
```

We checked the balance with help of value_counts(), we counted the number of 0's and 1's in "Outcome" column which are labels in our dataset. They are distributed in nearly 1:2 ratio, and provides non-biased and accurate results.

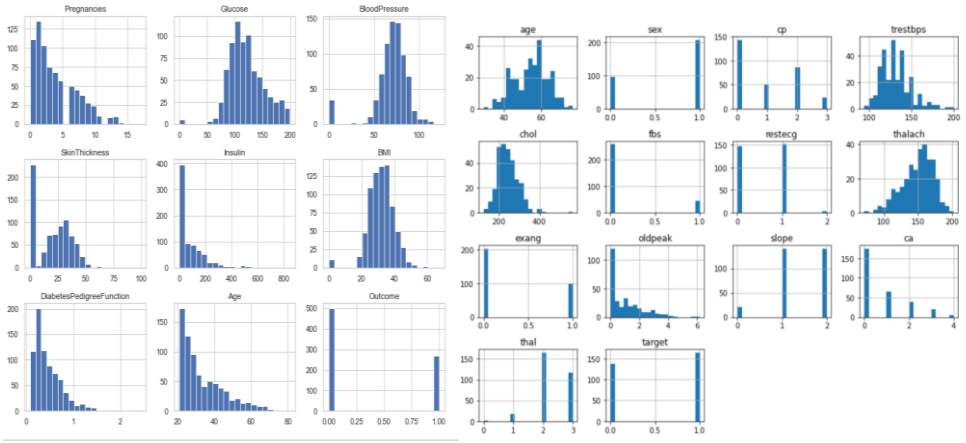
3.2.3 Data Visualization:

Dataset data visualization is very important part in the whole prediction process. Data visualization helps in following ways:

- Data understanding
- Feature selection
- Communicating insights
- Data quality assessment
- Supporting decision making
- Pattern recognition, and many more important ways.

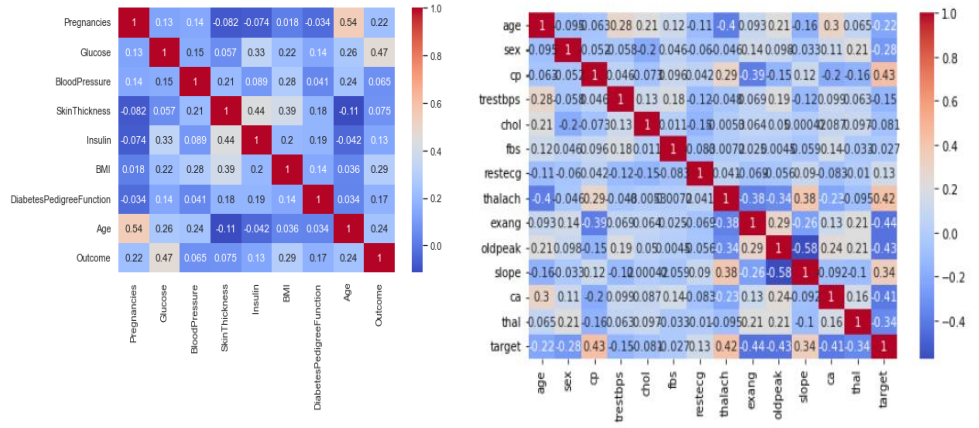
All visualizations libraries are imported from https://matplotlib.org

Histogram & Distribution of both datasets



"Pima Indians Diabetes" and "Heart Disease Detection"

Correlation matrix

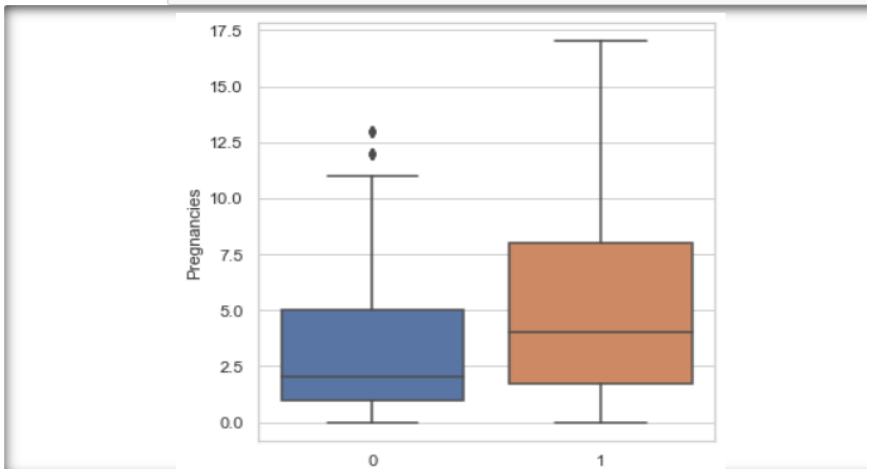


"Pima Indians Diabetes" and "Heart Disease Detection"

Box plot

Same plots for all features in both the datasets we have used.

```
In [20]: data1 = df.drop("Outcome",axis="columns")
         y1 = df.Outcome
         for i in data1.columns:
             plt.figure(figsize=(5, 5))
             sns.boxplot(x=y1, y=i, data=data1)
             plt.tight_layout()
             plt.show()
```



3.2.4 Scaling/Normalization:

We standardized/scaled/normalized our data using the StandardScaler() method [35]. This helped us to put all the data on the same scale so that our models could work better and predict better results. This is really important because it helps us trust our data more as all feature comes to same scale. Before scaling we are separating the outcome column from dataset so that our labels don't change and we perform scaling on remaining part of dataset features.

```
In [13]: from sklearn.preprocessing import StandardScaler #standerdization process

         scaler = StandardScaler()
         X_scaled = scaler.fit_transform(X)
         X_scaled[:3]

Out[13]: array([[ 0.63994726,  0.84832379,  0.14964075,  0.90726993, -0.69289057,
                   0.20401277,  0.46849198,  1.4259954 ],
                 [-0.84488505, -1.12339636, -0.16054575,  0.53090156, -0.69289057,
                  -0.68442195, -0.36506078, -0.19067191],
                 [ 1.23388019,  1.94372388, -0.26394125, -1.28821221, -0.69289057,
                  -1.10325546,  0.60439732, -0.10558415]])
```

Data Scaling Methodology:

For standardizing the data, we used the StandardScaler method. This way, the numbers in the data have mean of zero and a standard deviation of one, which helps us compare them better.

Formula for Standardization = $(X - \mu) / \sigma$

Where, observation value (X), the mean (μ) and the standard deviation (σ)

$$\sigma = \sqrt{\frac{\Sigma(X - \mu)^2}{N}}$$

$$\mu = \frac{\Sigma X}{N}$$

X - The Value in the data distribution
μ - The population Mean
N - Total Number of Observations

Image Credit: https://www.cuemath.com/data/standard-deviation/

Benefits of Data Scaling:
Scaling the data through the StandardScaler method enhances the efficiency of our models. It prevents any feature from dominating the prediction process by itself due to its scale, it ensures the fair consideration of all features. Additionally, scaled data often leads to faster convergence during model training as it avoids large calculations.

3.2.5 Splitting Data:
We divided our dataset into two parts using the `train_test_split` function from the `sklearn.model_selection` library [35]. This helps us have separate sets for training and testing our models.

Dataset Splitting Using 'train_test_split()' Function
To achieve this separation, we used the `train_test_split` function provided by the `sklearn.model_selection` library. This method divides the dataset into training and testing subsets, following a pre-determined proportion, with randomness involved.

```
In [14]: from sklearn.model_selection import train_test_split  #splitting dataset
         X_train1, X_test1, y_train, y_test = train_test_split(X_scaled, y, random_state=10)
```

Dividing the Dataset
Using the `train_test_split` function, we separated our dataset into two parts: a training set, which is used to train our models, and a testing set, which is used to observe how accurate our models prediction is. For splitting features into train and test we are using scaled data.

Benefits of Dataset Splitting
Splitting the dataset is essential for estimating how accurately our models will perform on new, real-world data. It helps to test the models' capacity to apply what they've learned to new/real-world data beyond their initial training.

3.2.6 Feature Selection/Extraction:

We performed feature selection using the `SelectKBest` method from the `sklearn.feature_selection` library, along with the `f_classif` function [35]. This approach allowed us to choose the most relevant features for our analysis.

```
In [15]: from sklearn.feature_selection import SelectKBest, f_classif
         num_features_to_select = 5
         selector = SelectKBest(score_func=f_classif, k=num_features_to_select)
         X_train = selector.fit_transform(X_train1, y_train)
         X_test = selector.transform(X_test1)
```

Feature Selection Using SelectKBest and f_classif:

To make our analysis better, we used this technique to choose the most important features from our dataset. This helps us pay attention to the things that really matter for predicting in our models.

Feature Selection Methodology:

For this purpose, we used the `SelectKBest` method available in the `sklearn.feature_selection` library. This technique picks out 'n' most important features that have a big contribution with what we're trying to achieve in prediction. We used the `f_classif` function to figure out how strong this contribution is.

Benefits of Feature Selection:

By using the `SelectKBest` method with the `f_classif` function, we made our dataset simpler but kept the most important features. This makes our models work better, fast and prevents them from learning too much from the less important data, which could lead to mistakes or confusion.

3.3 Create Variations

Train each base model on different parts of the data.

3.3.1 Bagging with k-NN and Logistic Regression

In our study, we utilized the Bagging ensemble strategy to boost the performance of the k-Nearest Neighbors (k-NN) base model and Logistic Regression. To introduce verity, we trained k-NN on multiple versions of the dataset and for did same with logistic regression on separate bootstrap samples randomly extracted from our dataset. Separating dataset randomly is called bootstrapping. Bootstrapping helps to create subsets of data from main dataset.

```
for _ in range(n_classifiers):
    # Bootstrap sample of the training data
    index = np.random.choice(len(X_train), len(X_train), replace=True)
    X_train_btstr = X_train[index]
    y_train_btstr = y_train.iloc[index]
```

For each k-NN and logistic regression model version. For every subset we store the predictions it generated on these subsets for the test set. So after completion this

process with all subsets we have created, we take maximum votes from all predicted sets. This ensemble method of predictions to generate a final prediction that is more reliable and accurate. By training on different parts and combining their predictions, this method adds variety that helps us avoid mistakes and gives us better predictions.

3.3.2 AdaBoost with Decision Tree and Logistic Regression

In our study, we also used an ensemble method called AdaBoost to make the Logistic Regression and Decision Tree model work even better. We trained the Logistic Regression and Decision tree models in different ways using AdaBoost. AdaBoost helps weak models get better, and here, the Logistic Regression and Decision tree are the base models we used as weak models. Each time we trained the models, we used different parts of our data in every iteration, especially focusing on the data that the previous models couldn't get right. We put all these different guesses together to make a stronger and more accurate final guess. By training the Logistic Regression and Decision tree in different ways and adding up the predictions, our method brings in lots of differences that help us avoid mistakes and get better results.

3.4 Aggregate

3.4.1 Bagging:

We aggregate the different predictions for all our bags made by the k-Nearest Neighbors (k-NN) if we choose k-NN as base model or Logistic Regression models if we choose Logistic regression as our base model. Each of these model learned from different parts of the data that we picked randomly from original scaled dataset. Then, in the Bagging function, we used a voting system to make our last prediction. This means we looked at all the predictions we got and chose the one that models predicted the most. This way, we're sure about our final prediction because it's backed up by what majority models predicted. This approach makes our prediction stronger, accurate and more reliable.

```
    # Aggregating predictions using majority voting
    bagged_predictions = []
    for i in range(len(predictions[0])):
        labels = [predictions[j][i] for j in range(n_bags)]
        unique_labels, label_counts = np.unique(labels, return_counts=True)
        major_label = unique_labels[np.argmax(label_counts)]
        bagged_predictions.append(major_label)
    return bagged_predictions
```

To evaluate, we stored the separate predictions generated by the k-NN and Logistic Regression models. By applying a majority vote, we collectively consider all these predictions and select the one that is predicted by the majority of times. By combining the predictions and taking a majority voting strategy, our Bagging ensemble approach shows its capacity to enhance predictive accuracy and model stability.

3.4.2 Boosting:

Boosting is a method that helps models, become better by working as a team. In Boosting with Logistic Regression, first we start with a simple Logistic Regression model. Then, we with help of AdaBoost, looked at where it makes mistakes and focus on those areas in the next training round. This makes each new model try to fix the errors the previous one made. We do this many times, making a group of Logistic Regression models, each model is good at solving different problems. When we want to make a prediction, we ask each model for its predictions. We pay more attention to the models that were good before. This teamwork makes our predictions more accurate and dependable, especially when the data is complicated.

3.5 Predict

3.5.1 Bagging

Predicting with Bagging and taking base model as Logistic Regression or k-NN, involves using the strengths of both methods to make better predictions. In this process, we first create multiple subsets of original scaled data and passed every subset to Logistic Regression models and stored predictions made by model every time, did same with k-NN model as if we chose it as base model. These models work together in a team with bagging function, contributing their opinions to the final prediction. Instead of relying on just one model, Bagging with Logistic Regression or k-NN lets us benefit from the combined wisdom of the models with bagging function. This approach helps us make more accurate and reliable predictions, particularly when dealing with complex data patterns. At the end with help of maximum voting we decide the final prediction and pass it get compared with test labels. It's like getting advice from multiple experts to make a smarter decision.

3.5.2 Boosting

Predicting with AdaBoost and taking base model as Logistic Regression or Decision Tree involves using a teamwork approach to improve predictions in every iteration. In this method, we start with a basic Logistic Regression model or Decision tree model as per choice of base model and identify where it struggles with different part of dataset. Then, we create a new model that focuses on those challenges and tries to fix the errors made by the previous iteration. This process is repeated many times, building a team of Logistic Regression models or Decision tree models with AdaBoost, each specialized in handling specific difficulties. When we want to make a prediction, we gather opinions from all these models and give more importance to the ones that have performed well before. This effort from model helps us make more accurate predictions, especially when dealing with complex data patterns or uncertainties. It's like learning from our mistakes and getting better with each new attempt.

3.6 Optimize

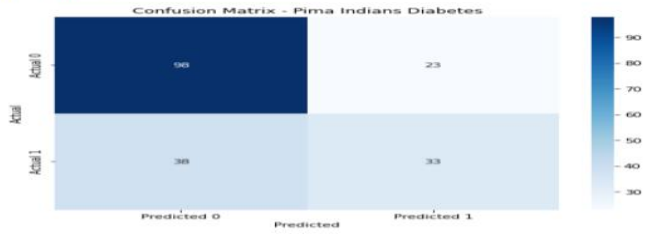
Adjust the way models are combined and their settings for the best results.

3.6.1 K-NN (k-Nearest Keighbors):

In k-NN we can optimize our results or predictions by changing value of k. As value of k represents how many nearest neighbors data we are going to consider for test data point.

For k=3

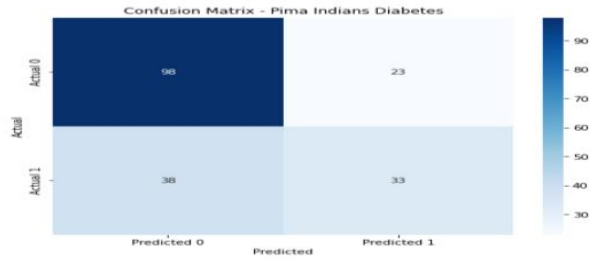
```
 Choose estimater
1. KNN
2. LOGESTIC
3. Decision Tree

 Enter your choice: 1
Enter K: 3
```




```
Traditional k-NN Accuracy:  68.22916666666666 %
```

For k=5

```
 Choose estimater
1. KNN
2. LOGESTIC
3. Decision Tree

 Enter your choice: 1
Enter K: 5
```

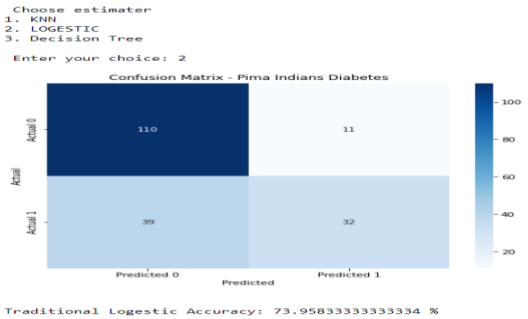



```
Traditional k-NN Accuracy:  68.22916666666666 %
```

As we can see in above two scenarios we can't see any change in accuracy this may be because our dataset doesn't have lot of data. But if we try with some bigger datasets we can see the difference in accuracy.

3.6.2 Logistic Regression:

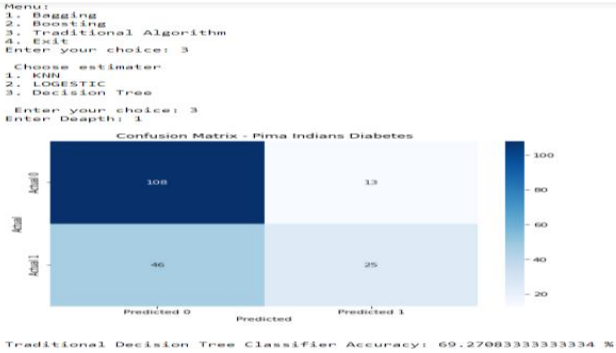
As we are importing this model for our study there is as such changes we can do with its parameters but we can change the random state values. As now we randomly choose 42 as random state.



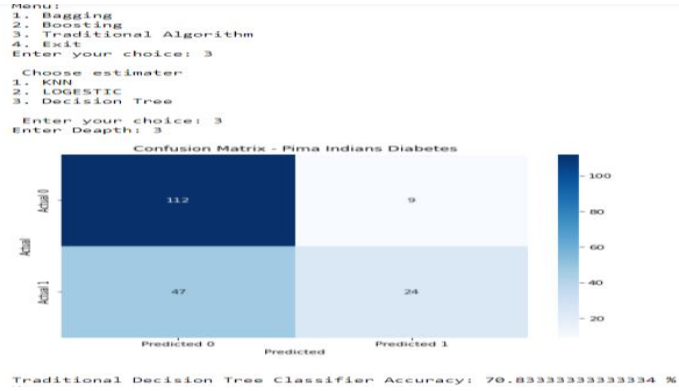
3.6.3 Decision Tree Classifier:

We can optimise the accuracy of decision tree by changing the depth of tree.

Depth = 1



Depth = 3



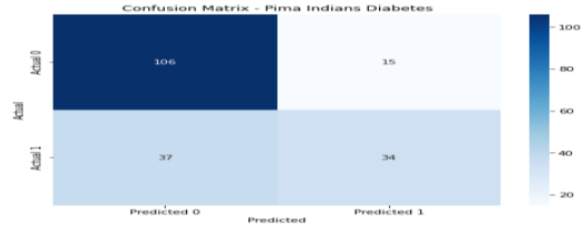
As we can see in above results accuracy is optimised after changing depth of tree.

3.6.4 Bagging with k-NN:

As we already discussed about k-NN, we can change the number of nearest neighbors to manipulate the accuracy. So now when we are using k-NN as a base estimator for our bagging ensemble method we can control the number of bags or bootstrapped datasets we want to create from original scaled dataset. So this factor can help us to achieve better accuracy in predictions. We can optimize our accuracy by making combinations of number of bags with k values.

Number of bags = 10 & value of k = 3.

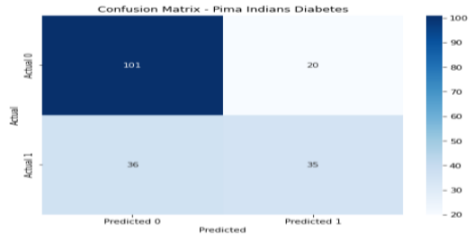
```
Menu:
1. Bagging
2. Boosting
3. Traditional Algorithm
4. Exit
Enter your choice: 1

Choose base estimater
1. KNN
2. LOGESTIC
Enter your choice: 1
Enter number of bags: 10
Enter K: 3
```




```
Bagging k-NN Accuracy: 72.91666666666666 %
```

Number of bags = 15 & value of k = 5.

```
Choose base estimater
1. KNN
2. LOGESTIC
Enter your choice: 1
Enter number of bags: 15
Enter K: 5
```



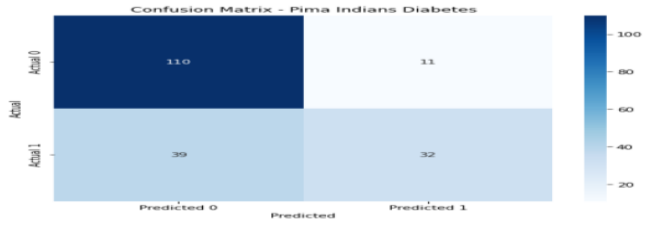

```
Bagging k-NN Accuracy: 70.83333333333334 %
```

As we can see from above experiment we can optimise the accuracy of model.

3.6.5 Bagging with logistic regression:

Just like bagging with k-NN, in bagging with logistic regression we can also optimise the accuracy of model by changing the number of bags.

```
Menu:
1. Bagging
2. Boosting
3. Traditional Algorithm
4. Exit
Enter your choice: 1

Choose base estimater
1. KNN
2. LOGESTIC
Enter your choice: 2
Enter number of bags: 50
```




```
Bagging Logestic Accuracy: 73.95833333333334 %
```

3.6.6 AdaBoost with logistic regression:

We can optmise Adaboost for base model ir contaion with the help of the number of estimaters or we can say number of iterations in which model gona train.

```
Menu:
1. Bagging
2. Boosting
3. Traditional Algorithm
4. Exit
Enter your choice: 2

Choose base estimater
1. Decision Tree
2. LOGESTIC
Enter your choice: 2
Enter number of estimators: 50

Boosting with Logestic Regression Accuracy: 73.95833333333334 %
Menu:
1. Bagging
2. Boosting
3. Traditional Algorithm
4. Exit
Enter your choice: 2

Choose base estimater
1. Decision Tree
2. LOGESTIC
Enter your choice: 2
Enter number of estimators: 5

Boosting with Logestic Regression Accuracy: 73.4375 %
```

As we can see in above two results we changed the estimators of AdaBoost as 50 in first and 5 in second, so we can see slight difference in their predictions.

3.6.7 AdaBoost with decision tree:

While using AdaBoost with decision tree as base model, we can optimise the accuracy by changing parameters of AdaBoost estimators and also the depth of decision tree.

```
Menu:
1. Bagging
2. Boosting
3. Traditional Algorithm
4. Exit
Enter your choice: 2

Choose base estimater
1. Decision Tree
2. LOGESTIC
Enter your choice: 1
Enter number of estimators: 50
Enter Deapth: 2

Boosting with Decision Tree Classifier Accuracy: 69.27083333333334 %
Menu:
1. Bagging
2. Boosting
3. Traditional Algorithm
4. Exit
Enter your choice: 2

Choose base estimater
1. Decision Tree
2. LOGESTIC
Enter your choice: 1
Enter number of estimators: 5
Enter Deapth: 1

Boosting with Decision Tree Classifier Accuracy: 71.35416666666666 %
```

As we can see the accuracy is increased after changing the parameters of model as we discussed earlier.

4. DATASET

4.1 Pima Indians Diabetes Dataset

The "Pima Indians Diabetes" dataset is a well-known dataset used in machine learning and data analysis due to its clean and noiseless data. Dataset contains information about group of Pima Indian women, specifically focusing on whether women have diabetes or they do not. The dataset is often used for building predictive models. Those predictive model is to determine the similarity or likelihood of an individual having diabetes based on various features introduced in dataset. Below is general description of the dataset:
Dataset: https://www.kaggle.com/gargmanas/pima-indians-diabetes.

	A	B	C	D	E	F	G	H	I
1	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
2	6	148	72	35	0	33.6	0.627	50	1
3	1	85	66	29	0	26.6	0.351	31	0
4	8	183	64	0	0	23.3	0.672	32	1
5	1	89	66	23	94	28.1	0.167	21	0
6	0	137	40	35	168	43.1	2.288	33	1
7	5	116	74	0	0	25.6	0.201	30	0
8	3	78	50	32	88	31	0.248	26	1
9	10	115	0	0	0	35.3	0.134	29	0
10	2	197	70	45	543	30.5	0.158	53	1
11	8	125	96	0	0	0	0.232	54	1
12	4	110	92	0	0	37.6	0.191	30	0
13	10	168	74	0	0	38	0.537	34	1
14	10	139	80	0	0	27.1	1.441	57	0
15	1	189	60	23	846	30.1	0.398	59	1
16	5	166	72	19	175	25.8	0.587	51	1
17	7	100	0	0	0	30	0.484	32	1
18	0	118	84	47	230	45.8	0.551	31	1
19	7	107	74	0	0	29.6	0.254	31	1
20	1	103	30	38	83	43.3	0.183	33	0
21	1	115	70	30	96	34.6	0.529	32	1
22	3	126	88	41	235	39.3	0.704	27	0
23	8	99	84	0	0	35.4	0.388	50	0
24	7	196	90	0	0	39.8	0.451	41	1
25	9	119	80	35	0	29	0.263	29	1
26	11	143	94	33	146	36.6	0.254	51	1
27	10	125	70	26	115	31.1	0.205	41	1
28	7	147	76	0	0	39.4	0.257	43	1
29	1	97	66	15	140	23.2	0.487	22	0
30	13	145	82	19	110	22.2	0.245	57	0

diabetes (1)

Number of Instances: The dataset typically contains 768 instances, where each instance represents a different Pima Indian woman. From 768 instances 500 instances are for Pima Indian women don't have diabetes and 268 instances for having diabetes.

Number of Attributes: Each instance in the dataset has 8 attributes or features, which are used to predict the target variable (whether the woman has diabetes or not). After observation we found the data inside the dataset is clean and don't contain any noise like dataset doesn't contain any duplicate rows, any empty or null values. Still we have performed necessary data pre-processing just for confidence.

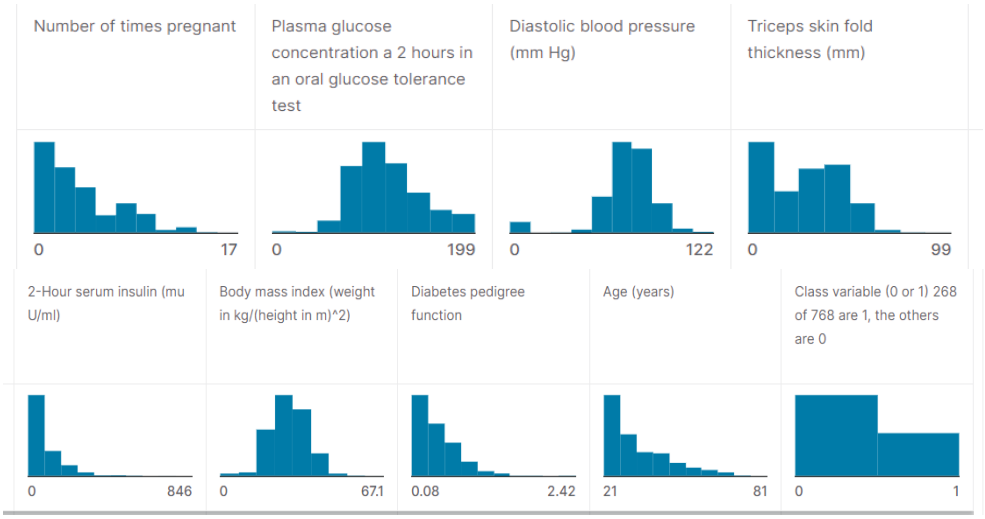


Image credit: https://www.kaggle.com/gargmanas/pima-indians-diabetes

Here are the attributes or features present in the dataset:

1. Pregnancies: Number of times the woman has been pregnant.
2. Glucose: Plasma glucose concentration after a 2-hour oral glucose tolerance test.
3. Blood Pressure: Diastolic blood pressure (mm Hg).
4. Skin Thickness: Triceps skinfold thickness (mm).
5. Insulin: 2-Hour serum insulin (mu U/ml).
6. BMI: Body mass index, which is a measure of body fat based on height and weight.
7. Diabetes Pedigree Function: A function that represents the likelihood of diabetes based on family history.
8. Age: Age of the individual in years.

Outcome (Target Variable): The binary target variable (0 and 1) represents whether the individual has diabetes or not. This is the target variable that machine learning models aim to predict.

This dataset is quite more used with classification algorithms, as the goal is to predict output based on other features and its outcomes are categorical.

4.2 Heart Disease Dataset

A "Heart Disease Dataset" typically contains data related to individuals and their cardiovascular health. The dataset include various features and information that can be used to predict or diagnose heart disease. This dataset mainly focuses on predicting heart disease for individual. Here are common attributes you might find in such a dataset:

Dataset: https://www.kaggle.com/datasets/zeeshanmulla/heart-disease-dataset

age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
63	1	3	145	233	1	0	150	0	2.3	0	0	1	1
37	1	2	130	250	0	1	187	0	3.5	0	0	2	1
41	0	1	130	204	0	0	172	0	1.4	2	0	2	1
56	1	1	120	236	0	1	178	0	0.8	2	0	2	1
57	0	0	120	354	0	1	163	1	0.6	2	0	2	1
57	1	0	140	192	0	1	148	0	0.4	1	0	1	1
56	0	1	140	294	0	0	153	0	1.3	1	0	2	1
44	1	1	120	263	0	1	173	0	0	2	0	3	1
52	1	2	172	199	1	1	162	0	0.5	2	0	3	1
57	1	2	150	168	0	1	174	0	1.6	2	0	2	1
54	1	0	140	239	0	1	160	0	1.2	2	0	2	1
48	0	2	130	275	0	1	139	0	0.2	2	0	2	1
49	1	1	130	266	0	1	171	0	0.6	2	0	2	1
64	1	3	110	211	0	0	144	1	1.8	1	0	2	1
58	0	3	150	283	1	0	162	0	1	2	0	2	1
50	0	2	120	219	0	1	158	0	1.6	1	0	2	1
58	0	2	120	340	0	1	172	0	0	2	0	2	1
66	0	3	150	226	0	1	114	0	2.6	0	0	2	1
43	1	0	150	247	0	1	171	0	1.5	2	0	2	1
69	0	3	140	239	0	1	151	0	1.8	2	2	2	1
59	1	0	135	234	0	1	161	0	0.5	1	0	3	1
44	1	2	130	233	0	1	179	1	0.4	2	0	2	1
42	1	0	140	226	0	1	178	0	0	2	0	2	1
61	1	2	150	243	1	1	137	1	1	1	0	2	1
40	1	3	140	199	0	1	178	1	1.4	2	0	3	1
71	0	1	160	302	0	1	162	0	0.4	2	2	2	1
59	1	2	150	212	1	1	157	0	1.6	2	0	2	1
51	1	2	110	175	0	1	123	0	0.6	2	0	2	1
65	0	2	140	417	1	0	157	0	0.8	2	1	2	1

Number of Instances: The dataset typically contains 303 instances, where each instance represents a different individual. From 303 instances 138 instances are for individual don't have heart disease and 164 instances for having heart disease.

Number of Attributes: Each instance in the dataset has 13 attributes or features, which are used to predict the target variable (whether the individual having heart disease or not). After observation for this dataset we found the data inside the dataset is clean and don't contain any noise like dataset doesn't contain any duplicate rows, any empty or null values as our previous dataset does. Still we have performed necessary data pre-processing just for confidence.

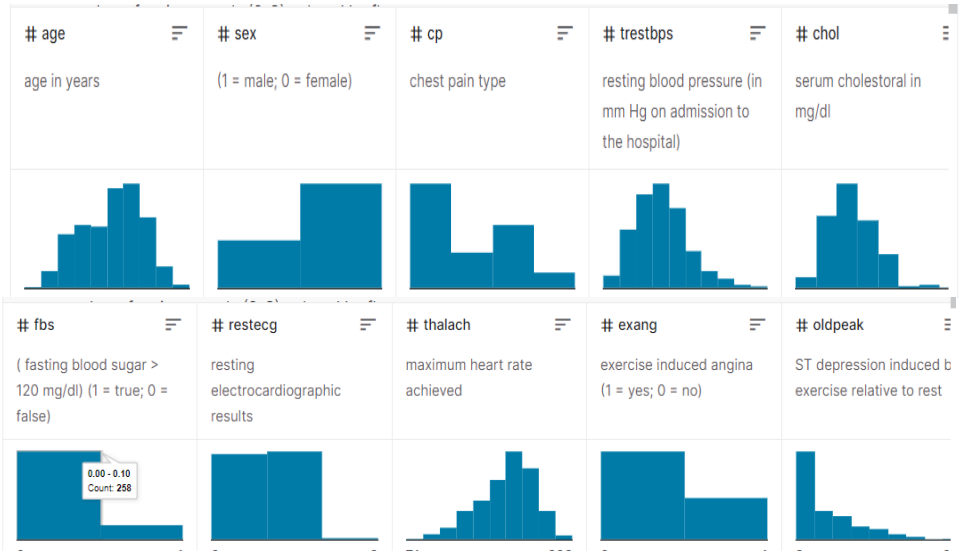


Image Credit: https://www.kaggle.com/datasets/zeeshanmulla/heart-disease-dataset

Here are the features in dataset:

1. age : age in years for individual
2. sex : gender information (1 = male; 0 = female)
3. cp : chest pain type
4. trestbps : resting blood pressure (in mm Hg on admission to the hospital)
5. chol : serum cholestoral in mg/dl
6. fbs : (fasting blood sugar > 120 mg/dl) (1 = true; 0 = false)
7. restecg : resting electrocardiographic results
8. thalach : maximum heart rate achieved
9. exang : exercise induced angina (1 = yes; 0 = no)
10. oldpeak : ST depression induced by exercise relative to rest
11. slope : the slope of the peak exercise ST segment
12. ca : number of major vessels (0-3) coloured by fluoroscopy
13. thal3 : normal; 6 = fixed defect; 7 = reversible defect

Outcome (Target Variable): The binary target variable (0 and 1) represents whether the individual has heart disease or not. This is the target variable that our ensemble models aim to predict.

5. RESULT AND ANALYSIS

In this section we are going to describe about the results we came across by performing above methodology with above described dataset. So with help of following points we will discuss our result and analysis:

5.1 Performance matrices:

It is very important to evaluate the models we are creating and take close look on the results they are providing. To evaluate and compare there are varies of performance matrices. We are using Accuracy as a performance matrix for our model evaluation. Accuracy represents the correctly identified instances from the overall instances. So higher the accuracy, better the model performed.

"Pima Indians Diabetes" dataset

MODEL	PARAMETERS	ACCURACY (%)
k-NN	k = 3	68.315666
	k = 5	68.229166
	k = 7	68.219666
	k = 9	68.229167
Logistic Regression		73.958333
Decision Tree	Max Depth = 1	69.270833
	Max Depth = 3	70.833333
	Max Depth = 5	73.437566
	Max Depth = 7	69.270833
Bagging with k-NN	Bags = 10 & k = 3	73.437500
	Bags = 15 & k = 5	72.395833
	Bags = 20 & k = 7	71.875000
	Bags = 50 & k = 9	72.395833
Bagging with logistic regression	Bags = 30	73.958333
	Bags = 50	73.958333
	Bags = 100	73.958333
	Bags = 200	73.958333
AdaBoost with Decision Tree	Estimators = 30 & Max Depth = 1	75.000000
	Estimators = 50 & Max Depth = 3	72.395833
	Estimators = 70 & Max Depth = 5	75.000000
	Estimators = 100 & Max Depth = 7	75.000000
AdaBoost with Logistic Regression	Estimators = 30	74.479166
	Estimators = 50	73.958333
	Estimators = 100	73.958333
	Estimators = 200	73.958333

"Heart Disease Detection" Dataset

MODEL	PARAMETERS	ACCURACY (%)
k-NN	k = 3	71.052632
	k = 5	71.052632
	k = 7	67.105263
	k = 9	65.789474
Logistic Regression		75.000000
Decision Tree	Max Depth = 1	75.000000
	Max Depth = 3	75.000000
	Max Depth = 5	76.312578
	Max Depth = 7	75.000000
Bagging with k-NN	Bags = 10 & k = 3	73.684211
	Bags = 15 & k = 5	76.315789
	Bags = 20 & k = 7	68.421053
	Bags = 50 & k = 9	69.736842
Bagging with logistic regression	Bags = 30	75.000000
	Bags = 50	75.000000
	Bags = 100	75.000000
	Bags = 200	75.000000
AdaBoost with Decision Tree	Estimators = 30 & Max Depth = 1	76.315789
	Estimators = 50 & Max Depth = 3	73.684211
	Estimators = 70 & Max Depth = 5	72.368421
	Estimators = 100 & Max Depth = 7	75.000000
AdaBoost with Logistic Regression	Estimators = 30	75.000000
	Estimators = 50	75.000000
	Estimators = 100	75.000000
	Estimators = 200	75.000000

As we can observe in above table with help of parameters change we can increase accuracy for the model. This will also help us to choose the best model among the all.

5.2 Model Comparison:

As per the results we have got in our performance matrix which is Accuracy in our project, we can compare the models with help of this. So we can decide which model out-performs all and provide us with better results.

"Pima Indian Diabetes" Dataset

Traditional vs Ensemble methods:

As we can observe from performance matrix we can say for every k value k-NN with bagging provides a better results. But for logistic regression provides same results irrespective of the models.

Boosting help to increase logistic regression accuracy as we can observe that in performance matrix. Same with decision tree it gives best accuracy with it for every value of depth

"Heart Disease Detection" Dataset

Traditional vs Ensemble methods:

Bagging

For this dataset as well ensemble methods performed very well, and they are able to perform better than the traditional models as we can compare that with help of above performance matrix.

We can see that for every value of k we are able to get better performance in bagging ensemble method.

In case of logistic regression we can observe that accuracy is same over all models.

Boosting

As we can observe from performance matrix we can say for every depth we can observe we are getting near about same or better predictions in decision trees with AdaBoost.

For logistic regression we can say we observed same results all over the models.

5.3 Model Selection

Now with the help of performance matrix and model compression we can decide which model to select for our further studies. So with help of model comparison we can select the best performer models those are

For Bagging ensemble method we can select Logistic Regression as base model as it provides better results than k-NN as base model.

For AdaBoost ensemble method we can select Decision Trees as it provide better results as compare to Logistic Regression model. Also show some improvement in results.

6. PROFESSIONAL ISSUES

As we studied and experimented ensemble methods in overall project we get to know that they are very powerful techniques. As they combine the predictions of multiple base model to give overall better predictions. However like other methodologies there are certain professional issues and considerations to keep in mind while using ensemble methods. Some of them are faced by us and some of them came in our studies:

Model complexity: Ensemble methods can lead to complex model especially when we provide them with large number of base models to work with or number of estimators are too large. Complex models are hard to interpret, debug and maintain as well. We can see this in performance metrics.

Training time: Majorly when we are training ensemble methods in which we require training multiple model sequentially or parallel and in some ensemble methods if we provide with some large number of bags then it will take long period of time to execute. This process can be computationally intensive and might require more time compared to single model.

Over-fitting: Especially ensemble methods reduce over-fitting as compared to individual models, but still there is risk if base model is highly complex.

Bias: Ensemble methods can sometime inherit biases from the base models. If the base model have some errors or biases, these could be reflected in ensemble methods.

Choice of base model: The effectiveness and quality of ensemble methods is highly depends on diversity and quality of base models. Choosing appropriate base model is very important.

Generalization: As we observed in our project that ensemble methods perform exceptionally well on the training data but sometimes struggle to generalise to new or unseen data. Careful separation of test data is very important.

Data Imbalance: If dataset has imbalanced classes, then ensemble methods might need some additional considerations to ensure balanced predictions. So to avoid this we have to balance the classes in dataset.

Scalability: Some ensemble methods may not scale or perform well with very large datasets due to memory and computational constraints.

7. CONCLUSION

Ensemble methods are very powerful techniques in machine learning which involve combing with base model and improve the predictive performance and generalization. Through their ability to reduce bias, variance, over-fitting ensemble methods offer a number of advantages that can be leveraged for many type of problems.

7.1 Satisfied Objectives

Data pre-processing

As the dataset we have chosen for our project is very stable and have good quality of data. Still we have performed data cleaning, data transformation with help of Standard scalar, Feature selection, checked for data imbalance. Along with that we split the data into training and testing sets. To gain insights and relationships in the data we performed data visualization.

Understand Ensemble techniques

Literature review we studied and information we got from scikit-learn website, helped us to get good understanding about ensemble methods how they work.

Ensemble methods using various machine learning algorithms:

As we discussed earlier in methodology section we have performed and experimented ensemble methods with different machine learning algorithms. And results after the experiments are discussed in Result and Analysis section 5.

Comparing various Machine learning models

We have experimented with combining various machine learning models with ensemble methods and their comparisons we have performed in Model comparison section 5.2.

Improve predictive accuracy

To increase the predictive accuracy of model we tried changing their parameters and observed their behaviour in performance matrix section 5.1.
So with help of changing parameters we can increase predictive accuracy of model and make it more powerful.

Provide Valuable insights for the decision makers like stakeholders, clients

By offering enhanced predictive performance and deeper understanding of the relationship and patterns in the data, ensemble methods make it easy for decision-makers to make their decisions.

7.2 Findings:

7.2.1 Advantages:

Enhanced Performance:
As we observed in our experiments in this project ensemble methods outperforms individual models by combining their strengths and balancing for their weaknesses.

Reduced Over-fitting:
By averaging or combining with models, ensemble methods can reduce risk of over-fitting, resulting in better generalization of new or unseen data.

Stability:
Ensemble methods are less sensitive to noise and variations in the data, so this quality make them more stable for noisy datasets.

Improved Robustness:
Ensemble methods are less likely to be get affected by outliers or anomalies in the data.

Flexibility:
Ensemble methods can work with a variety of base models and algorithms and also allow customization based on the problem we are solving.

Applicability:
Ensemble methods can be applied to both classification and regression problems, and also to other tasks like ranking and recommendation systems.

7.2.2 Considerations

Complexity:
Ensemble methods can result in complex models if we use a large number of base models, this will potentially making them harder to interpret and maintain.

Computation Time:
Some ensemble methods require training multiple base models, which will lead them to need more computational power. Just like in bagging if we make large number of bags then it will demand more computational power.

Bias from Base Models:
If base models are biased, those biases can carry over to the ensemble methods predictions.

Parameter Tuning:
Ensemble methods require parameters that need tuning to achieve optimal performance.

Interpretability:
While individual models like linear regression or decision trees might have straightforward decision rules that are easy to interpret on other hand, ensemble methods involve a combination of models. This combination can leads to more complex decision boundaries, making it harder to explain how the ensemble arrives at its final predictions.

Data Dependencies:
There is high possibility that ensemble methods may not perform well if base models are highly dependent on similar data patterns.

7.3 Future Scope:

Combining different types of ensemble methods like bagging and boosting together to make hybrid ensemble methods that will lead to leverage strengths of different ensemble techniques. We can create graphical user interface for our project, so that it will be easy for end user to use the system we have created.

Techniques for automatically selecting base models within ensemble methods to improve efficiency and performance could see further development. Integrating ensemble methods with reasonable AI techniques could enhance the understanding ability of ensemble predictions and decision-making. Developing ensemble methods that can adapt to changing data distributions and evolving patterns in real-time scenarios can be a significant future direction.

7.4 Final Remark:

When implementing ensemble methods, the selection of an appropriate approach depends on several key factors: the nature of the problem, the characteristics of the available data, and the trade-offs between model complexity, interpretability, and predictive performance. Among the list of ensemble techniques available, such as Bagging and Boosting, each offers distinct strategies for integrating different models into a combined predictive entity.

However, the strategic selection of an ensemble method is only the beginning. Thorough and accurate evaluation, along with accurate analysis of the ensemble's performance, is important. This careful analysis ensures that the ensemble's behaviour bring into line with the future objectives and is a component of the decision-making process.

To further optimize the ensemble's effectiveness, experimenting with various configurations and using different base models is an invaluable exercise. This experimental effort empowers practitioners to reveal the most effective combination, striking the right balance between predictive precision, interpretability, and overall performance.

8. USER MANUAL

Table of Contents

1. Introduction
Welcome to the Ensemble Methods User Manual. This manual provides step-by-step instructions on how to use the provided code to implement ensemble methods for binary classification using the Pima Indians Diabetes dataset. The code utilizes libraries such as pandas, scikit-learn, matplotlib, and seaborn to perform data analysis and modeling.

2. Prerequisites
Before you begin, make sure you have the required libraries installed. You will need:

- pandas
- numpy
- scikit-learn
- matplotlib
- seaborn

3. Getting Started
Follow these steps to get started:
1. Download the Pima Indians Diabetes dataset from the provided link.
2. Save the dataset as "diabetes.csv" in the same directory as the code.
3. Open the Python environment and run the provided code.

4. Understanding the Dataset

- The code loads the dataset, removes rows with missing values, and drops duplicates.
- Basic information about the dataset, including data types and non-null counts, is displayed.
- The first few rows of the dataset and column names are printed.
- The code checks for missing values and displays the count of missing values for each column.
- Descriptive statistics of the dataset are shown.

5. Data Visualization

- Histograms and distribution plots are generated to visualize the distribution of each feature.
- A correlation matrix heat-map is plotted to show the correlations between features.
- Box plot is plotted for understanding the distribution of each feature.

6. Data Pre-processing

- The dataset is split into features (X) and target (y).
- Features are scaled using StandardScaler to standardize data.
- Data is split into training and testing sets using train_test_split.
- Feature selection is performed using SelectKBest and f_classif.

7. Methods

Three types of methods are implemented:

1. Bagging: Combines predictions from multiple models.
2. Boosting: Enhances the performance of a weak base model.
3. Traditional Algorithms: K-NN, Logistic Regression, and Decision Tree classifiers.

8. Using the Code

- The code provides a menu with options to choose the ensemble method or exit.
- For bagging, choose the base estimator (KNN or Logistic Regression) and the number of bags and for k-NN we can choose number of nearest neighbors.
- For Boosting, select the base estimator (Decision Tree or Logistic Regression) and the number of estimators and for decision tree we can decide depth of tree.
- For Traditional Algorithms, choose the model (KNN, Logistic Regression, or Decision Tree) and tune their parameters accordingly.

9. Interpreting Results

- After executing each ensemble method or traditional algorithm, a confusion matrix heat-map is displayed.
- The confusion matrix helps visualize the accuracy and misclassifications.
- The accuracy of the model is also printed as a percentage.

10. Conclusion

You have successfully used the provided code to implement ensemble methods and traditional algorithms. The code allows you to explore different ensemble techniques and compare their performance to traditional algorithms.

Feel free to experiment with different settings, such as the number of bags, estimators, and hyper-parameters, to understand how they affect the results.

For any further assistance or questions, please refer to the documentation of the libraries used.

9. REFERENCES

1. Breiman, L. (1996). Bagging predictors. Machine Learning, 24(2), 123-140.
2. Caruana, R., Niculescu-Mizil, A., Crew, G., & Ksikes, A. (2004). Ensemble selection from libraries of models. Proceedings of the 21st International Conference on Machine Learning, 18, 18.
3. Freund, Y., & Schapire, R. E. (1999). A short introduction to boosting. Journal-Japanese Society For Artificial Intelligence, 14(5), 771-780.
4. Gao, Y., Pedrycz, W., & Guo, S. (2017). An enhanced ensemble clustering approach for disease prediction. Expert Systems with Applications, 71, 273-280.
5. Brown, G., & Kuncheva, L. I. (2007). Diversity-based hybrid model for credit scoring. Expert Systems with Applications, 32(4), 946-956.
6. Drucker, H., Wu, D., & Vapnik, V. (1999). Support vector machines for spam categorization. IEEE Transactions on Neural Networks, 10(5), 1048-1054.
7. Viola, P., & Jones, M. J. (2001). Rapid object detection using a boosted cascade of simple features. Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 1, I-511.
8. Wu, T. T. (2004). Ensemble methods in bioinformatics. In Data mining in bioinformatics (pp. 123-153). Springer.
9. Pang, B., & Lee, L. (2008). Opinion mining and sentiment analysis. Foundations and Trends® in Information Retrieval, 2(1–2), 1-135.
10. Hastie, T., Tibshirani, R., & Friedman, J. (2001). The Elements of Statistical Learning: Data Mining, Inference, and Prediction. Springer.
11. Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). ImageNet Classification with Deep Convolutional Neural Networks. Advances in Neural Information Processing Systems, 25, 1097-1105.
12. Chandola, V., Banerjee, A., & Kumar, V. (2009). Anomaly detection: A survey. ACM Computing Surveys, 41(3), 1-58.
13. Hosmer Jr, D. W., & Lemeshow, S. (2004). Applied Logistic Regression. John Wiley & Sons.
14. Peduzzi, P., Concato, J., Kemper, E., Holford, T. R., & Feinstein, A. R. (1996). A simulation study of the number of events per variable in logistic regression analysis. Journal of Clinical Epidemiology, 49(12), 1373-1379.
15. Moro, S., Cortez, P., & Rita, P. (2014). A data-driven approach to predict the success of bank telemarketing. Decision Support Systems, 62, 22-31.
16. Baesens, B., Roesch, D., Scheule, H., & Satchell, S. (2017). Credit Risk Analytics: Measurement Techniques, Applications, and Examples in SAS. John Wiley & Sons.
17. Cover, T., & Hart, P. (1967). Nearest neighbor pattern classification. IEEE Transactions on Information Theory, 13(1), 21-27.
18. Li, Y., Cai, Z., Zhou, X., & Song, Q. (2013). Ensemble of KNN model based on bagging method. Proceedings of the 2013 International Conference on Computer Sciences and Applications, 1-4.
19. Zou, Y., & He, H. (2019). Improved Method for Data Classification Based on Ensemble of KNN Models. Proceedings of the 2019 International Conference on Computer Information and Big Data Applications, 224-228.

20. Das, S., & Dey, D. (2017). Hybridizing k-nearest neighbor and ensemble bagging for anomaly detection. International Journal of Computer Applications, 176(19).
21. Wang, Z., Zhao, D., & Cao, L. (2018). Combining Bagging with Logistic Regression for Classification. Proceedings of the 2018 International Conference on Big Data and Machine Learning, 185-190.
22. Tang, Y., Hong, Y., & Nguyen, H. T. (2015). A novel bagging ensemble classifier based on logistic regression. Applied Soft Computing, 28, 156-164.
23. Santos, M. S., Rodrigues, P. P., & Santos, J. M. (2019). An improved ensemble model using bagging and logistic regression for classification problems. Expert Systems with Applications, 131, 373-384.
24. Mavroforakis, C., & Tsoumakas, G. (2019). Bagging Ensemble for Imbalanced Data. Proceedings of the 2019 IEEE International Conference on Data Science and Advanced Analytics, 424-433.
25. Breiman, L., Friedman, J., Stone, C., & Olshen, R. (1984). Classification and Regression Trees. CRC press.
26. Zeng, X., & Song, D. (2005). Customer churn prediction using improved balanced random forests. Proceedings of the 2005 IEEE International Conference on Data Mining, 389-396.
27. Wolpert, D. H. (1992). Stacked generalization. Neural Networks, 5(2), 241-
28. Zhu, S., & Rosset, S. (2009). Adaptive and self-confident multi-class boosting. Journal of Machine Learning Research, 10(Jan), 1009-1041.
29. Attias, H. (1999). Inferring parameters and structure of latent variable models by variational Bayes. In Proceedings of the Fifteenth Conference on Uncertainty in Artificial Intelligence (pp. 21-30).
30. Phua, C., Lee, V. C., Smith-Miles, K., & Gayler, R. W. (2005). A comprehensive survey of data mining-based fraud detection research. ArXiv preprint cs/0502013.
31. Vapnik, V. (1998). Statistical learning theory. John Wiley & Sons.
32. Verbeke, W., Martens, D., & Baesens, B. (2012). Improving customer relationship management using combined predictive modeling techniques. Journal of Marketing Research, 49(3), 359-377.
33. Huang, H., & Zhang, Y. (2018). Combining k-NN with Bagging to Predict Diabetes. Proceedings of the 2018 International Conference on Computer, Information and Telecommunication Systems, 146-149.
34. https://youtu.be/RtrBtAKwcxQ Youtube video for bagging with k-NN.
35. https://scikit-learn.org/stable/ to understand meaning of imported libraries.

APPENDIX

Code for "Pima Indian Diabetes" Detection:

<h1 align='center'>DIABETES PREDICTION WITH ENSEMBLE METHODS</h1>

Important Libraries

Pandas: https://pandas.pydata.org/docs/user_guide/index.html

Numpy: https://numpy.org/devdocs/user/quickstart.html

sklearn: https://scikit-learn.org

Collections: https://www.guru99.com/python-counter-collections-example.html

Seaborn:
https://seaborn.pydata.org/tutorial/introduction.html#:~:text=Seaborn%20is%20a%20library%20for,explore%20and%20understand%20your%20data.

```
import pandas as pd #data structures and functions
import numpy as np #support for arrays and mathematical functions

from sklearn.linear_model import LogisticRegression #model for binary classification and regression
from sklearn.tree import DecisionTreeClassifier #decision trees for classification
from sklearn.ensemble import AdaBoostClassifier #boosting model
from sklearn.metrics import confusion_matrix #function to compute a confusion matrix
from collections import Counter #class for counting

import matplotlib.pyplot as plt #plotting library for visualizations
import seaborn as sns #data visualization library based on Matplotlib
```

Load and Understand the Data

Dataset: https://www.kaggle.com/gargmanas/pima-indians-diabetes

```
df = pd.read_csv("diabetes1.csv")

df = df.dropna()
len(df)

df = df.drop_duplicates()
len(df)
```

Display basic info about the dataset

```
print(df.info())
```

```
print(df.head())

print(df.columns)

df.isnull().sum()

df.describe()
```

classes are balanced or imbalanced

```
class_distribution = df['target'].value_counts()
print(class_distribution)
```

Data Visualization

Histograms and Distributions: https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.hist.html

```
df.hist(figsize=(10, 8), bins=20)
plt.tight_layout()
plt.show()
```

Correlation Matrix: https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.corr.html

```
correlation_matrix = df.corr()
plt.figure(figsize=(8, 5))
sns.heatmap(correlation_matrix, annot=True, cmap="coolwarm")
plt.show()
```

Box Plots: https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.boxplot.html

```
data1 = df.drop("target",axis="columns")
y1 = df.target
for i in data1.columns:
    plt.figure(figsize=(5, 5))
    sns.boxplot(x=y1, y=i, data=data1)
    plt.tight_layout()
    plt.show()
```

There is slight imbalance in our dataset but since it is not major we will not worry about it!

<h3>Train test split</h3>

```
X = df.drop("target",axis="columns")
y = df.target
```

Scaling data using standard scalar: https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html

```
from sklearn.preprocessing import StandardScaler #standerdization process

scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
X_scaled[:3]
```

Splitting Data for training and testing: https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html

```
from sklearn.model_selection import train_test_split  #splitting dataset
X_train1, X_test1, y_train, y_test = train_test_split(X_scaled, y, random_state=10)
```

Feature Selection: https://scikit-learn.org/stable/modules/feature_selection.html

```
from sklearn.feature_selection import SelectKBest, f_classif #for selecting more important features
num_features_to_select = 5
selector = SelectKBest(score_func=f_classif, k=num_features_to_select)
X_train = selector.fit_transform(X_train1, y_train)
X_test = selector.transform(X_test1)

# K-NN classifier function

def euclidean_distance(p1, p2):  #Euclidean Distance function
    return sum((a - b) ** 2 for a, b in zip(p1, p2)) ** 0.5 #square root of squared distance

def knn1(train_data, labels, test_data, k): #KNN predictor function
    distances = [(euclidean_distance(test_data, data_point), label) for data_point, label in zip(train_data, labels)]
    sorted_distances = sorted(distances, key=lambda x: x[0]) #Sort the list of distances
    k_near_neighbors = sorted_distances[:k]  # K nearest points
    label_cnts = {}
    label_cnts = Counter(k_near_neighbors)
    return label_cnts.most_common(1)[0][0]

# Bagging Function
```

Ensemble methods Bagging: https://scikit-learn.org/stable/modules/ensemble.html

```
def bagging_FUNC(X_train, X_test, y_train, estimater, n_bags):
```

```
    predictions = []
    for _ in range(n_bags):
        # Bootstrap sample of the training data
        index = np.random.choice(len(X_train), len(X_train), replace=True)
        X_train_btstr = X_train[index]
        y_train_btstr = y_train.iloc[index]

        # KNN classifier on the bootstrap sample
        if estimater>0:
            #predicted_labels = knnsrc(X_train_btstr, y_train_btstr, X_test,
estimater)
            predicted_labels = [knn1(X_train_btstr, y_train_btstr, X_t, estimater) for
X_t in X_test]
        # LogisticRegression on the bootstrap sample
        if estimater==0:
            model = LogisticRegression(random_state=42)
            # Fit the model to the training data
            model.fit(X_train, y_train)
            # Make predictions on the test set
            predicted_labels = model.predict(X_test)

        predictions.append(predicted_labels)

    # Aggregating predictions using majority voting
    bagged_predictions = []
    for i in range(len(predictions[0])):
        labels = [predictions[j][i] for j in range(n_bags)]
        unique_labels, label_counts = np.unique(labels, return_counts=True)
        major_label = unique_labels[np.argmax(label_counts)]
        bagged_predictions.append(major_label)
    return bagged_predictions

# MAIN

References:
confusion matrix: https://scikit-
learn.org/stable/modules/generated/sklearn.metrics.confusion_matrix.html

AdaBoost: https://scikit-learn.org/stable/modules/ensemble.html#adaboost

while 1:
    print("\nMenu:")
    print("1. Bagging")
    print("2. Boosting")
    print("3. Traditional Algorithm")
    print("4. Comparision")
    print("5. Exit")
    choice = input("Enter your choice: ")
#-----------------------------------------------------------------------------------------------
-------------
```

```
#For Bagging
   if choice == "1":
      print("\nChoose base estimater")
      print("1. KNN")
      print("2. LOGESTIC")
      choice1 = input("Enter your choice: ")
      num_bags = int(input("Enter number of bags: "))
      if choice1 == "1":
         KN = int(input("Enter K: "))
         bagged_predictions = bagging_FUNC(X_train, X_test, y_train, KN,
num_bags)
         bagged_predictions = np.array(bagged_predictions)
         bagged_predictions = bagged_predictions.astype(int)
         # Plot the confusion matrix as a heatmap
         cm = confusion_matrix(y_test, bagged_predictions)
         plt.figure(figsize=(8, 6))
         sns.heatmap(cm, annot=True, fmt="d", cmap="Blues",
                  xticklabels=["Predicted 0", "Predicted 1"],
                  yticklabels=["Actual 0", "Actual 1"])
         plt.xlabel('Predicted')
         plt.ylabel('Actual')
         plt.title('Confusion Matrix - Bagging with k-NN')
         plt.show()
         accuracy = format((np.mean(bagged_predictions == y_test)*100),".6f")
         print("\nBagging k-NN Accuracy:", accuracy, "%")
      elif choice1 == "2":
         bagged_predictions = bagging_FUNC(X_train, X_test, y_train, 0,
num_bags)
         # Plot the confusion matrix as a heatmap
         cm = confusion_matrix(y_test, bagged_predictions)
         plt.figure(figsize=(8, 6))
         sns.heatmap(cm, annot=True, fmt="d", cmap="Blues",
                  xticklabels=["Predicted 0", "Predicted 1"],
                  yticklabels=["Actual 0", "Actual 1"])
         plt.xlabel('Predicted')
         plt.ylabel('Actual')
         plt.title('Confusion Matrix - Bagging with Logestic Regression')
         plt.show()
         accuracy = format((np.mean(bagged_predictions == y_test)*100),".6f")
         print("\nBagging Logestic Accuracy:", accuracy, "%")
      else:
         print("\nInvalid choice. Please try again.")

#-----------------------------------------------------------------------------------------------------
-------------
#For Boosting
   elif choice == "2":
      print("\nChoose base estimater")
      print("1. Decision Tree")
      print("2. LOGESTIC")
```

```
        choice1 = input("Enter your choice: ")
        estimators_boost = int(input("Enter number of estimators: "))
        if choice1 == "1":
            KN = int(input("Enter Deapth: "))

            dt_classifier = DecisionTreeClassifier(max_depth=KN)

            adaboost_classifier = AdaBoostClassifier(base_estimator=dt_classifier,
n_estimators=estimators_boost)

            # Train the AdaBoost classifier
            adaboost_classifier.fit(X_train, y_train)

            # Evaluate the classifier on the test data
            accuracy = format((adaboost_classifier.score(X_test, y_test)*100),".6f")

            print("\nBoosting with Decision Tree Classifier Accuracy:", accuracy,
"%")
        elif choice1 == "2":
            # Create a base Logistic Regression
            LgRegression = LogisticRegression()

            # Create an AdaBoost classifier with KNN as base learner
            adaboost_classifier                                                =
AdaBoostClassifier(base_estimator=LgRegression,
n_estimators=estimators_boost)

            # Train the AdaBoost classifier
            adaboost_classifier.fit(X_train, y_train)

            # Evaluate the classifier on the test data
            accuracy = format((adaboost_classifier.score(X_test, y_test)*100),".6f")
            print("\nBoosting with Logestic Regression Accuracy:", accuracy, "%")
        else:
            print("\nInvalid choice. Please try again.")

#-----------------------------------------------------------------------------------------------------
-------------
#Traditional Algorithms
    elif choice == "3":
        print("\n Choose estimater")
        print("1. KNN")
        print("2. LOGESTIC")
        print("3. Decision Tree")
        choice2 = input("\n Enter your choice: ")
        if choice2 == "1":
            KKN = int(input("Enter K: "))
            predicted_labels = [knn1(X_train, y_train, d, KKN) for d in X_test]
            predicted_labels1      =          [predicted_labels[i][1]      for      i      in
range(len(predicted_labels))]
```

```
    # Plot the confusion matrix as a heatmap
    cm = confusion_matrix(y_test, predicted_labels1)
    plt.figure(figsize=(8, 6))
    sns.heatmap(cm, annot=True, fmt="d", cmap="Blues",
                xticklabels=["Predicted 0", "Predicted 1"],
                yticklabels=["Actual 0", "Actual 1"])
    plt.xlabel('Predicted')
    plt.ylabel('Actual')
    plt.title('Confusion Matrix - k-NN')
    plt.show()

    accuracy = format((np.mean(predicted_labels1 == y_test)*100),".6f")
    print("\nTraditional k-NN Accuracy:", accuracy, "%")
elif choice2 == "2":
    model = LogisticRegression(random_state=42)
    # Fit the model to the training data
    model.fit(X_train, y_train)
    # Make predictions on the test set
    predicted_labels = model.predict(X_test)
    cm = confusion_matrix(y_test, predicted_labels)
    plt.figure(figsize=(8, 6))
    sns.heatmap(cm, annot=True, fmt="d", cmap="Blues",
                xticklabels=["Predicted 0", "Predicted 1"],
                yticklabels=["Actual 0", "Actual 1"])
    plt.xlabel('Predicted')
    plt.ylabel('Actual')
    plt.title('Confusion Matrix - Logestic Rreegression')
    plt.show()
    accuracy = format((np.mean(predicted_labels == y_test)*100),".6f")
    print("\nTraditional Logestic Accuracy:", accuracy, "%")
elif choice2 == "3":
    KN = int(input("Enter Deapth: "))

    dt_classifier = DecisionTreeClassifier(max_depth=KN)
    # Fit the model to the training data
    dt_classifier.fit(X_train, y_train)
    # Make predictions on the test set
    predicted_labels = dt_classifier.predict(X_test)
    cm = confusion_matrix(y_test, predicted_labels)
    plt.figure(figsize=(8, 6))
    sns.heatmap(cm, annot=True, fmt="d", cmap="Blues",
                xticklabels=["Predicted 0", "Predicted 1"],
                yticklabels=["Actual 0", "Actual 1"])
    plt.xlabel('Predicted')
    plt.ylabel('Actual')
    plt.title('Confusion Matrix - Decision Tree')
    plt.show()
    accuracy = format((np.mean(predicted_labels == y_test)*100),".6f")
    print("\nTraditional Decision Tree Classifier Accuracy:", accuracy, "%")
else:
```

```
            print("\nInvalid choice. Please try again.")
#-------------------------------------------------------------------------------------------------------------
------------------------
#Comarision
    elif choice == "4":
        print("\n Choose compair")
        print("1. Bagging")
        print("2. Boosting")
        choice2 = input("\n Enter your choice: ")
        if choice2 == "1":
            baggingc=[]
            KKN = int(input("Enter K: "))
            nbag= int(input("Enter num of bags: "))
            predicted_labels = [knn1(X_train, y_train, d, KKN) for d in X_test]
            predicted_labels1 = [predicted_labels[i][1] for i in
range(len(predicted_labels))]
            baggingc.append((np.mean(predicted_labels1 == y_test)*100))

            bagged_predictions = bagging_FUNC(X_train, X_test, y_train, KKN,
nbag)
            bagged_predictions = np.array(bagged_predictions)
            bagged_predictions = bagged_predictions.astype(int)
            baggingc.append(np.mean(bagged_predictions == y_test)*100)

            model = LogisticRegression(random_state=42)
            model.fit(X_train, y_train)
            predicted_labelsl = model.predict(X_test)
            baggingc.append(np.mean(predicted_labelsl == y_test)*100)

            bagged_predictionsl = bagging_FUNC(X_train, X_test, y_train, 0, nbag)
            baggingc.append(np.mean(bagged_predictionsl == y_test)*100)

            labels = ['KNN', 'Bagging KNN', 'Logistic Regression', 'Bagging Logistic
Regression']
            colors = ['purple', 'purple', 'green', 'green']
            plt.figure(figsize=(10, 6))
            # Create an array of indices for the x-axis
            x = np.arange(len(labels))
            # Create the bar graph
            plt.bar(x, baggingc, width=0.4, align='center', tick_label=labels,
color=colors)
            # Set the y-axis label
            plt.ylabel('Accuracy (%)')
            # Set a title for the graph
            plt.title('Comparison of Different Models with Bagging')
            # Show the graph
            plt.show()

        elif choice2 == "2":
            boostc=[]
```

```
            estimators_boost = int(input("Enter number of Estimators: "))
            KN = int(input("Enter Deapth for Decision Tree: "))
            dt_classifier = DecisionTreeClassifier(max_depth=KN)
            dt_classifier.fit(X_train, y_train)
            predicted_labels = dt_classifier.predict(X_test)
            boostc.append(np.mean(predicted_labels == y_test)*100)

            adaboost_classifier = AdaBoostClassifier(base_estimator=dt_classifier,
n_estimators=estimators_boost)
            adaboost_classifier.fit(X_train, y_train)
            boostc.append(adaboost_classifier.score(X_test, y_test)*100)

            model = LogisticRegression(random_state=42)
            model.fit(X_train, y_train)
            predicted_labelsl = model.predict(X_test)
            boostc.append(np.mean(predicted_labelsl == y_test)*100)

            adaboost_classifierl = AdaBoostClassifier(base_estimator=model,
n_estimators=estimators_boost)
            adaboost_classifierl.fit(X_train, y_train)
            boostc.append(adaboost_classifierl.score(X_test, y_test)*100)

            labels = ['Decision Tree', 'AdaBoost Decision Tree', 'Logistic Regression',
'AdaBoost Logistic Regression']
            colors = ['purple', 'purple', 'green', 'green']
            # Create an array of indices for the x-axis

            x = np.arange(len(labels))
            plt.figure(figsize=(10, 6))
            # Create the bar graph
            plt.bar(x, boostc, width=0.4, align='center', tick_label=labels,
color=colors)

            # Set the y-axis label

            plt.ylabel('Accuracy (%)')
            # Set a title for the graph
            plt.title('Comparison of Different Models with AdaBoost')
            # Show the graph
            plt.show()
        else:
            print("\nInvalid choice. Please try again.")
#-----------------------------------------------------------------------------------------------
------------------------
#Exit
    elif choice == "5":
        print("\nThank You.")
        break
#-----------------------------------------------------------------------------------------------
------------------------
```

```
#Invalid choice
   else:
      print("\nInvalid choice. Please try again.")

#=====================================================================#
```

Code for "Heart Disease Prediction" Dataset

```
<h1 align='center'>HEART DISEASE PREDICTION ENSEMBLE METHODS</h1>

# Important Libraries

Pandas: https://pandas.pydata.org/docs/user_guide/index.html

Numpy: https://numpy.org/devdocs/user/quickstart.html

sklearn: https://scikit-learn.org

Collections: https://www.guru99.com/python-counter-collections-example.html

Seaborn:
https://seaborn.pydata.org/tutorial/introduction.html#:~:text=Seaborn%20is%20a%
20library%20for,explore%20and%20understand%20your%20data.

import pandas as pd #data structures and functions
import numpy as np #support for arrays and mathematical functions

from sklearn.linear_model import LogisticRegression #model for binary
classification and regression
from sklearn.tree import DecisionTreeClassifier #decision trees for classification
from sklearn.ensemble import AdaBoostClassifier #boosting model
from sklearn.metrics import confusion_matrix #function to compute a confusion
matrix
from collections import Counter #class for counting

import matplotlib.pyplot as plt #plotting library for visualizations
import seaborn as sns #data visualization library based on Matplotlib

# Load and Understand the Data

Dataset: https://www.kaggle.com/datasets/zeeshanmulla/heart-disease-dataset

df = pd.read_csv("Heart_data.csv")

df = df.dropna()
len(df)
```

```
df = df.drop_duplicates()
len(df)
```

Display basic info about the dataset

```
print(df.info())
```

```
print(df.head())
```

```
print(df.columns)
```

```
df.isnull().sum()
```

```
df.describe()
```

classes are balanced or imbalanced

```
class_distribution = df['target'].value_counts()
print(class_distribution)
```

Data Visualization

Histograms and Distributions: https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.hist.html

```
df.hist(figsize=(10, 8), bins=20)
plt.tight_layout()
plt.show()
```

Correlation Matrix: https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.corr.html

```
correlation_matrix = df.corr()
plt.figure(figsize=(8, 5))
sns.heatmap(correlation_matrix, annot=True, cmap="coolwarm")
plt.show()
```

Box Plots: https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.boxplot.html

```
data1 = df.drop("target",axis="columns")
y1 = df.target
for i in data1.columns:
   plt.figure(figsize=(5, 5))
   sns.boxplot(x=y1, y=i, data=data1)
   plt.tight_layout()
```

```
   plt.show()
```

There is slight imbalance in our dataset but since it is not major we will not worry about it!

```
<h3>Train test split</h3>

X = df.drop("target",axis="columns")
y = df.target
```

Scaling data using standard scalar: https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html

```
from sklearn.preprocessing import StandardScaler #standerdization process

scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
X_scaled[:3]
```

Splitting Data for training and testing: https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html

```
from sklearn.model_selection import train_test_split  #splitting dataset
X_train1, X_test1, y_train, y_test = train_test_split(X_scaled, y, random_state=10)
```

Feature Selection: https://scikit-learn.org/stable/modules/feature_selection.html

```
from sklearn.feature_selection import SelectKBest, f_classif #for selecting more important features
num_features_to_select = 5
selector = SelectKBest(score_func=f_classif, k=num_features_to_select)
X_train = selector.fit_transform(X_train1, y_train)
X_test = selector.transform(X_test1)

# K-NN classifier function

def euclidean_distance(p1, p2):  #Euclidean Distance function
   return sum((a - b) ** 2 for a, b in zip(p1, p2)) ** 0.5 #square root of squared distance

def knn1(train_data, labels, test_data, k): #KNN predictor function
   distances = [(euclidean_distance(test_data, data_point), label) for data_point, label in zip(train_data, labels)]
   sorted_distances = sorted(distances, key=lambda x: x[0]) #Sort the list of distances
   k_near_neighbors = sorted_distances[:k]  # K nearest points
   label_cnts = {}
```

```
    label_cnts = Counter(k_near_neighbors)
    return label_cnts.most_common(1)[0][0]

# Bagging Function

def bagging_FUNC(X_train, X_test, y_train, estimater, n_bags):

    predictions = []
    for _ in range(n_bags):
        # Bootstrap sample of the training data
        index = np.random.choice(len(X_train), len(X_train), replace=True)
        X_train_btstr = X_train[index]
        y_train_btstr = y_train.iloc[index]

        # KNN classifier on the bootstrap sample
        if estimater>0:
            #predicted_labels = knnsrc(X_train_btstr, y_train_btstr, X_test, estimater)
            predicted_labels = [knn1(X_train_btstr, y_train_btstr, X_t, estimater) for X_t
in X_test]
        # LogisticRegression on the bootstrap sample
        if estimater==0:
            model = LogisticRegression(random_state=42)
            # Fit the model to the training data
            model.fit(X_train, y_train)
            # Make predictions on the test set
            predicted_labels = model.predict(X_test)

        predictions.append(predicted_labels) #collecting all predictions togather

    # Aggregating predictions using majority voting
    bagged_predictions = []
    for i in range(len(predictions[0])):
        labels = [predictions[j][i] for j in range(n_bags)]
        unique_labels, label_counts = np.unique(labels, return_counts=True)
        major_label = unique_labels[np.argmax(label_counts)]
        bagged_predictions.append(major_label)
    return bagged_predictions

# MAIN
```

References:
confusion matrix: https://scikit-learn.org/stable/modules/generated/sklearn.metrics.confusion_matrix.html

Ensemble methods Bagging: https://scikit-learn.org/stable/modules/ensemble.html

AdaBoost: https://scikit-learn.org/stable/modules/ensemble.html#adaboost

```
while 1:
   print("\nMenu:")
   print("1. Bagging")
   print("2. Boosting")
   print("3. Traditional Algorithm")
   print("4. Comparision")
   print("5. Exit")
   choice = input("Enter your choice: ")
#-----------------------------------------------------------------------------------------------------------
-------
#For Bagging
   if choice == "1":
      print("\nChoose base estimater")
      print("1. KNN")
      print("2. LOGESTIC")
      choice1 = input("Enter your choice: ")
      num_bags = int(input("Enter number of bags: "))
      if choice1 == "1":
         KN = int(input("Enter K: "))
         bagged_predictions = bagging_FUNC(X_train, X_test, y_train, KN,
num_bags)
         bagged_predictions = np.array(bagged_predictions)
         bagged_predictions = bagged_predictions.astype(int)
         # Plot the confusion matrix as a heatmap
         cm = confusion_matrix(y_test, bagged_predictions)
         plt.figure(figsize=(8, 6))
         sns.heatmap(cm, annot=True, fmt="d", cmap="Blues",
                  xticklabels=["Predicted 0", "Predicted 1"],
                  yticklabels=["Actual 0", "Actual 1"])
         plt.xlabel('Predicted')
         plt.ylabel('Actual')
         plt.title('Confusion Matrix - Bagging with k-NN')
         plt.show()
         accuracy = format((np.mean(bagged_predictions == y_test)*100),".6f")
         print("\nBagging k-NN Accuracy:", accuracy, "%")
      elif choice1 == "2":
         bagged_predictions = bagging_FUNC(X_train, X_test, y_train, 0, num_bags)
         # Plot the confusion matrix as a heatmap
         cm = confusion_matrix(y_test, bagged_predictions)
         plt.figure(figsize=(8, 6))
         sns.heatmap(cm, annot=True, fmt="d", cmap="Blues",
                  xticklabels=["Predicted 0", "Predicted 1"],
```

```
                yticklabels=["Actual 0", "Actual 1"])
        plt.xlabel('Predicted')
        plt.ylabel('Actual')
        plt.title('Confusion Matrix - Bagging with Logestic Regression')
        plt.show()
        accuracy = format((np.mean(bagged_predictions == y_test)*100),".6f")
        print("\nBagging Logestic Accuracy:", accuracy, "%")
    else:
        print("\nInvalid choice. Please try again.")

#-------------------------------------------------------------------------------------------------------
-------
#For Boosting
  elif choice == "2":
    print("\nChoose base estimater")
    print("1. Decision Tree")
    print("2. LOGESTIC")
    choice1 = input("Enter your choice: ")
    estimators_boost = int(input("Enter number of estimators: "))
    if choice1 == "1":
        KN = int(input("Enter Deapth: "))

        dt_classifier = DecisionTreeClassifier(max_depth=KN)

        adaboost_classifier = AdaBoostClassifier(base_estimator=dt_classifier,
n_estimators=estimators_boost)

        # Train the AdaBoost classifier
        adaboost_classifier.fit(X_train, y_train)

        # Evaluate the classifier on the test data
        accuracy = format((adaboost_classifier.score(X_test, y_test)*100),".6f")

        print("\nBoosting with Decision Tree Classifier Accuracy:", accuracy, "%")
    elif choice1 == "2":
        # Create a base Logistic Regression
        LgRegression = LogisticRegression()

        # Create an AdaBoost classifier with KNN as base learner
        adaboost_classifier = AdaBoostClassifier(base_estimator=LgRegression,
n_estimators=estimators_boost)

        # Train the AdaBoost classifier
        adaboost_classifier.fit(X_train, y_train)
```

```
        # Evaluate the classifier on the test data
        accuracy = format((adaboost_classifier.score(X_test, y_test)*100),".6f")
        print("\nBoosting with Logestic Regression Accuracy:", accuracy, "%")
    else:
        print("\nInvalid choice. Please try again.")

#-----------------------------------------------------------------------------------------------------------
-------
#Traditional Algorithms
  elif choice == "3":
    print("\n Choose estimater")
    print("1. KNN")
    print("2. LOGESTIC")
    print("3. Decision Tree")
    choice2 = input("\n Enter your choice: ")
    if choice2 == "1":
        KKN = int(input("Enter K: "))
        predicted_labels = [knn1(X_train, y_train, d, KKN) for d in X_test]
        predicted_labels1 = [predicted_labels[i][1] for i in
range(len(predicted_labels))]
        # Plot the confusion matrix as a heatmap
        cm = confusion_matrix(y_test, predicted_labels1)
        plt.figure(figsize=(8, 6))
        sns.heatmap(cm, annot=True, fmt="d", cmap="Blues",
                    xticklabels=["Predicted 0", "Predicted 1"],
                    yticklabels=["Actual 0", "Actual 1"])
        plt.xlabel('Predicted')
        plt.ylabel('Actual')
        plt.title('Confusion Matrix - k-NN')
        plt.show()

        accuracy = format((np.mean(predicted_labels1 == y_test)*100),".6f")
        print("\nTraditional k-NN Accuracy:", accuracy, "%")
    elif choice2 == "2":
        model = LogisticRegression(random_state=42)
        # Fit the model to the training data
        model.fit(X_train, y_train)
        # Make predictions on the test set
        predicted_labels = model.predict(X_test)
        cm = confusion_matrix(y_test, predicted_labels)
        plt.figure(figsize=(8, 6))
        sns.heatmap(cm, annot=True, fmt="d", cmap="Blues",
                    xticklabels=["Predicted 0", "Predicted 1"],
                    yticklabels=["Actual 0", "Actual 1"])
        plt.xlabel('Predicted')
```

```
            plt.ylabel('Actual')
            plt.title('Confusion Matrix - Logestic Rreegression')
            plt.show()
            accuracy = format((np.mean(predicted_labels == y_test)*100),".6f")
            print("\nTraditional Logestic Accuracy:", accuracy, "%")
        elif choice2 == "3":
            KN = int(input("Enter Deapth: "))

            dt_classifier = DecisionTreeClassifier(max_depth=KN)
            # Fit the model to the training data
            dt_classifier.fit(X_train, y_train)
            # Make predictions on the test set
            predicted_labels = dt_classifier.predict(X_test)
            cm = confusion_matrix(y_test, predicted_labels)
            plt.figure(figsize=(8, 6))
            sns.heatmap(cm, annot=True, fmt="d", cmap="Blues",
                        xticklabels=["Predicted 0", "Predicted 1"],
                        yticklabels=["Actual 0", "Actual 1"])
            plt.xlabel('Predicted')
            plt.ylabel('Actual')
            plt.title('Confusion Matrix - Decision Tree')
            plt.show()
            accuracy = format((np.mean(predicted_labels == y_test)*100),".6f")
            print("\nTraditional Decision Tree Classifier Accuracy:", accuracy, "%")
        else:
            print("\nInvalid choice. Please try again.")
#-------------------------------------------------------------------------------------------------------------
------------------
#Comarision
    elif choice == "4":
        print("\n Choose compair")
        print("1. Bagging")
        print("2. Boosting")
        choice2 = input("\n Enter your choice: ")
        if choice2 == "1":
            baggingc=[]
            KKN = int(input("Enter K: "))
            nbag= int(input("Enter num of bags: "))
            predicted_labels = [knn1(X_train, y_train, d, KKN) for d in X_test]
            predicted_labels1 = [predicted_labels[i][1] for i in
range(len(predicted_labels))]
            baggingc.append((np.mean(predicted_labels1 == y_test)*100))

            bagged_predictions = bagging_FUNC(X_train, X_test, y_train, KKN, nbag)
            bagged_predictions = np.array(bagged_predictions)
```

```
        bagged_predictions = bagged_predictions.astype(int)
        baggingc.append(np.mean(bagged_predictions == y_test)*100)

        model = LogisticRegression(random_state=42)
        model.fit(X_train, y_train)
        predicted_labelsl = model.predict(X_test)
        baggingc.append(np.mean(predicted_labelsl == y_test)*100)

        bagged_predictionsl = bagging_FUNC(X_train, X_test, y_train, 0, nbag)
        baggingc.append(np.mean(bagged_predictionsl == y_test)*100)

        labels = ['KNN', 'Bagging KNN', 'Logistic Regression', 'Bagging Logistic
Regression']
        colors = ['purple', 'purple', 'green', 'green']
        plt.figure(figsize=(10, 6))
        # Create an array of indices for the x-axis
        x = np.arange(len(labels))
        # Create the bar graph
        plt.bar(x, baggingc, width=0.4, align='center', tick_label=labels, color=colors)
        # Set the y-axis label
        plt.ylabel('Accuracy (%)')
        # Set a title for the graph
        plt.title('Comparison of Different Models with Bagging')
        # Show the graph
        plt.show()

      elif choice2 == "2":
        boostc=[]
        estimators_boost = int(input("Enter number of Estimators: "))
        KN = int(input("Enter Deapth for Decision Tree: "))
        dt_classifier = DecisionTreeClassifier(max_depth=KN)
        dt_classifier.fit(X_train, y_train)
        predicted_labels = dt_classifier.predict(X_test)
        boostc.append(np.mean(predicted_labels == y_test)*100)

        adaboost_classifier = AdaBoostClassifier(base_estimator=dt_classifier,
n_estimators=estimators_boost)
        adaboost_classifier.fit(X_train, y_train)
        boostc.append(adaboost_classifier.score(X_test, y_test)*100)

        model = LogisticRegression(random_state=42)
        model.fit(X_train, y_train)
        predicted_labelsl = model.predict(X_test)
        boostc.append(np.mean(predicted_labelsl == y_test)*100)
```

```
        adaboost_classifierl = AdaBoostClassifier(base_estimator=model,
n_estimators=estimators_boost)
        adaboost_classifierl.fit(X_train, y_train)
        boostc.append(adaboost_classifierl.score(X_test, y_test)*100)

        labels = ['Decision Tree', 'AdaBoost Decision Tree', 'Logistic Regression',
'AdaBoost Logistic Regression']
        colors = ['purple', 'purple', 'green', 'green']
        # Create an array of indices for the x-axis

        x = np.arange(len(labels))
        plt.figure(figsize=(10, 6))
        # Create the bar graph
        plt.bar(x, boostc, width=0.4, align='center', tick_label=labels, color=colors)

        # Set the y-axis label

        plt.ylabel('Accuracy (%)')
        # Set a title for the graph
        plt.title('Comparison of Different Models with AdaBoost')
        # Show the graph
        plt.show()
      else:
        print("\nInvalid choice. Please try again.")
#-------------------------------------------------------------------------------------------------------
------------------
#Exit
  elif choice == "5":
    print("\nThank You.")
    break
#-------------------------------------------------------------------------------------------------------
--------------------
#Invalid choice
  else:
    print("\nInvalid choice. Please try again.")
```