**DEPARTMENT OF INFORMATION TECHNOLOGY**

**COURSE NAME:** Machine Learning Laboratory **COURSE CODE:** DJS22L602

**CLASS:** Third Year B. Tech **SEM:** VI

**Name :Anish Sharma**

**EXPERIMENT NO. 6**

**CO Measured:**

**CO3 Apply various machine learning techniques**

**TITLE:** To implement ensemble methods by performing comparative analysis on bagging and boosting techniques used for prediction.

**AIM / OBJECTIVE:**

Perform ensemble methods using python libraries for the following methods over the suitable selected dataset and compare results.

* Simple Ensemble Techniques o Max Voting o Averaging
* Advanced Ensemble techniques o Stacking o Blending o Bagging

o Boosting

**DESCRIPTION OF EXPERIMENT:**

Explain the Basic ensemble methods?

1. Averaging method: It is mainly used for regression problems. The method consists of building multiple models independently and returning the average of the prediction of all the models. In general, the combined output is better than an individual output because variance is reduced.

1. Max voting: It is mainly used for classification problems. The method consists of building multiple models independently and getting their individual output called ‘vote’. The class with maximum votes is returned as output.

Explain the Advanced ensemble methods?

1. Stacking: It is an ensemble method that combines multiple models (classification or regression) via meta-model (meta-classifier or meta-regression). The base models are trained on the complete dataset, then the meta-model is trained on features returned (as output) from base models. The base models in stacking are typically different. The meta-model helps to find the features from base models to achieve the best accuracy.

1. Blending: It is similar to the stacking method explained above, but rather than using the whole dataset for training the base-models, a validation dataset is kept separate to make predictions.

1. Bagging: It is also known as a bootstrapping method. Base models are run on bags to get a fair distribution of the whole dataset. A bag is a subset of the dataset along with a replacement to make the size of the bag the same as the whole dataset. The final output is formed after combining the output of all base models.

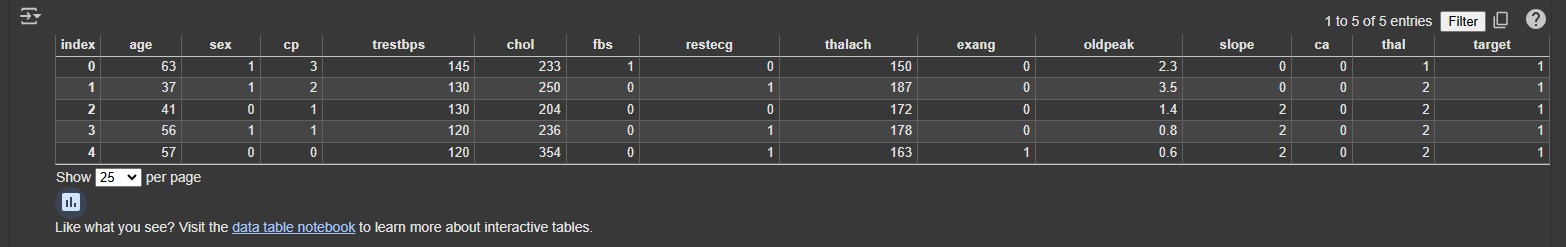
1. Boosting: Boosting is a sequential method–it aims to prevent a wrong base model from affecting the final output. Instead of combining the base models, the method focuses on building a new model that is dependent on the previous one. A new model tries to remove the errors made by its previous one. Each of these models is called weak learners. The final model (aka strong learner) is formed by getting the weighted mean of all the weak learners.

**PROCEDURE:**

Code :

import pandas as pd

heart\_data = pd.read\_csv('heart.csv') heart\_data.head()



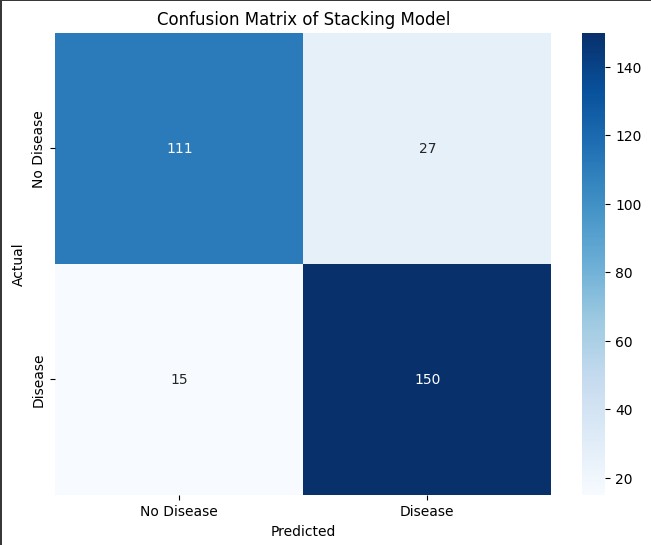
Steps to perform Stacking:

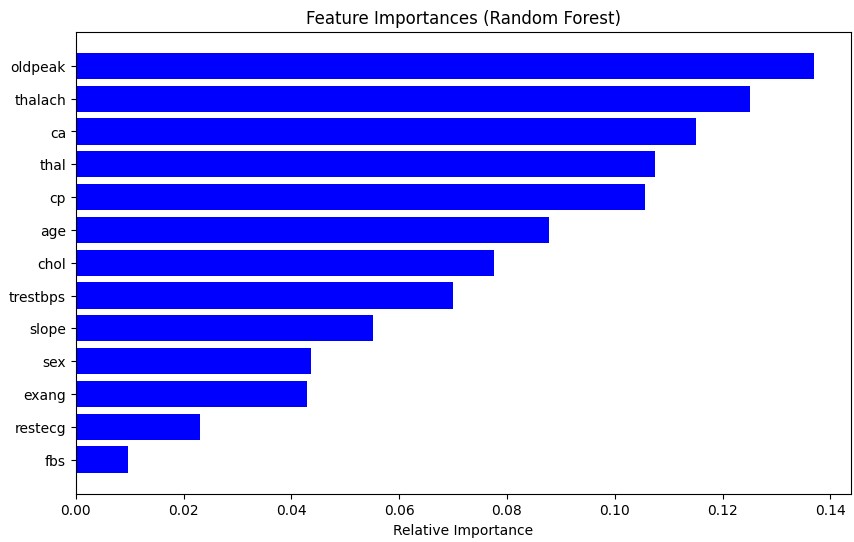
* 1. Split the train dataset into n parts
  2. A base model (say linear regression) is fitted on n-1 parts and predictions are made for the nth part. This is done for each one of the n part of the train set.
  3. The base model is then fitted on the whole train dataset.
  4. This model is used to predict the test dataset.
  5. The Steps 2 to 4 are repeated for another base model which results in another set of predictions for the train and test dataset.
  6. The predictions on train data set are used as a feature to build the new model.
  7. This final model is used to make the predictions on test dataset

|  |
| --- |
| 1. !pip install scikit-learn matplotlib seaborn 2. from sklearn.preprocessing import StandardScaler 3. from sklearn.linear\_model import LogisticRegression 4. from sklearn.model\_selection import KFold 5. from sklearn.metrics import accuracy\_score, confusion\_matrix 6. from sklearn.ensemble import RandomForestClassifier 7. import numpy as np 8. import pandas as pd 9. import matplotlib.pyplot as plt 10. import seaborn as sns   18.   1. # Load the dataset 2. heart\_data = pd.read\_csv('heart.csv')   21.   1. # Prepare data 2. X = heart\_data.drop('target', axis=1) 3. y = heart\_data['target']   25.   1. # Scale the data 2. scaler = StandardScaler() 3. X\_scaled = scaler.fit\_transform(X)   29.   1. # Split data into training and testing 2. kf = KFold(n\_splits=5, shuffle=True, random\_state=42)   32.   1. # Base models 2. base\_model\_1 = LogisticRegression(max\_iter=1000) 3. base\_model\_2 = RandomForestClassifier()   36.   1. # Meta-model (stacking model) 2. meta\_model = LogisticRegression(max\_iter=1000)   39.   1. # Store base model predictions and targets 2. train\_predictions = [] 3. train\_targets = []   43.   1. # Perform cross-validation for stacking 2. for train\_index, test\_index in kf.split(X\_scaled): 3. X\_train, X\_test = X\_scaled[train\_index], X\_scaled[test\_index] 47. y\_train, y\_test = y.iloc[train\_index], y.iloc[test\_index]   48.   1. base\_model\_1.fit(X\_train, y\_train) 2. base\_model\_2.fit(X\_train, y\_train) |

|  |
| --- |
| 51.   1. base\_pred\_1 = base\_model\_1.predict(X\_test) 2. base\_pred\_2 = base\_model\_2.predict(X\_test)   54.   1. train\_predictions.extend(np.column\_stack([base\_pred\_1, base\_pred\_2])) 2. train\_targets.extend(y\_test)   57.   1. # Combine predictions for training the meta-model 2. train\_meta\_features = np.array(train\_predictions).reshape(-1, 2) 60. meta\_model.fit(train\_meta\_features, train\_targets)   61.   1. # Get predictions for the entire dataset 2. base\_pred\_1\_final = base\_model\_1.predict(X\_scaled) 3. base\_pred\_2\_final = base\_model\_2.predict(X\_scaled) 4. test\_meta\_features = np.column\_stack([base\_pred\_1\_final, base\_pred\_2\_final]) 5. final\_predictions = meta\_model.predict(test\_meta\_features)   67.   1. # Evaluate 2. stacking\_accuracy = accuracy\_score(y, final\_predictions) 3. print(f"Stacking Model Accuracy: {stacking\_accuracy}")   71.   1. # Visualization 1: Confusion Matrix 2. cm = confusion\_matrix(y, final\_predictions) 3. plt.figure(figsize=(8, 6)) 4. sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", 5. xticklabels=['No Disease', 'Disease'], 6. yticklabels=['No Disease', 'Disease']) 7. plt.title('Confusion Matrix of Stacking Model') 8. plt.xlabel('Predicted') 9. plt.ylabel('Actual') 10. plt.show()   82.   1. # Visualization 2: Feature Importance of Base Model 2 (Random Forest) 2. if hasattr(base\_model\_2, 'feature\_importances\_'): 3. importances = base\_model\_2.feature\_importances\_ 4. features = X.columns 5. indices = np.argsort(importances)   88.   1. plt.figure(figsize=(10, 6)) 2. plt.title('Feature Importances (Random Forest)') |

1. plt.barh(range(len(indices)), importances[indices], color='b', align='center')
2. plt.yticks(range(len(indices)), [features[i] for i in indices])
3. plt.xlabel('Relative Importance') 94. plt.show()
4. else:
5. print("Base model 2 does not have feature\_importances\_ attribute.")





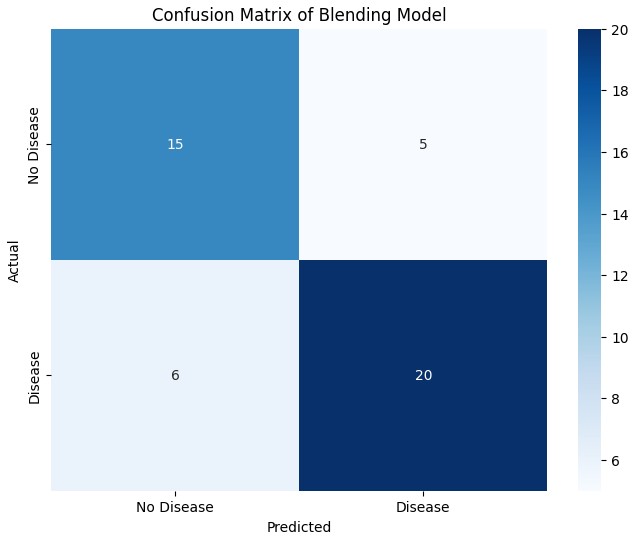
Steps to perform Blending:

* 1. Split the training dataset into train, test and validation dataset.
  2. Fit all the base models using train dataset.
  3. Make predictions on validation and test dataset.
  4. These predictions are used as features to build a second level model
  5. This model is used to make predictions on test and meta-features
  6. !pip install scikit-learn matplotlib seaborn --upgrade
  7. import pandas as pd
  8. from sklearn.model\_selection import train\_test\_split
  9. from sklearn.linear\_model import LogisticRegression
  10. from sklearn.ensemble import RandomForestClassifier
  11. from sklearn.metrics import accuracy\_score
  12. import matplotlib.pyplot as plt 13. import seaborn as sns

14.

* 1. # 1. Load and split the dataset
  2. heart\_data = pd.read\_csv('heart.csv') # Assuming 'heart.csv' is your dataset
  3. X = heart\_data.drop('target', axis=1)
  4. y = heart\_data['target']

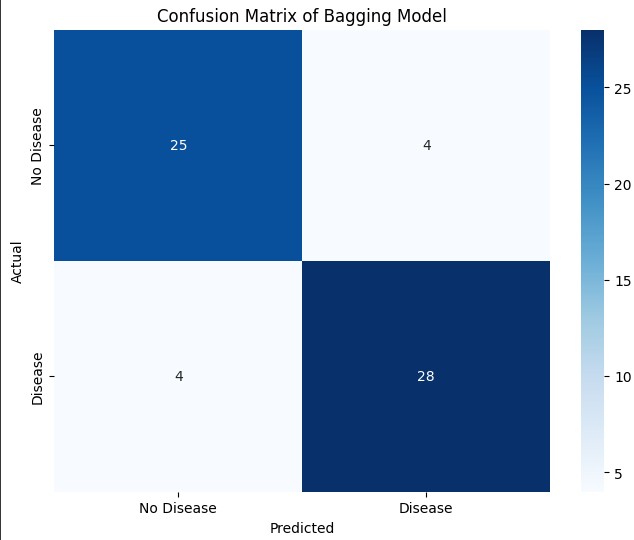
|  |  |
| --- | --- |
| 1. X\_train, X\_temp, y\_train, y\_temp = train\_test\_split(X, y, test\_size=0.3, random\_state=42) 2. X\_val, X\_test, y\_val, y\_test = train\_test\_split(X\_temp, y\_temp, test\_size=0.5, random\_state=42)   21.   1. # 2. Fit base models 2. base\_model\_1 = LogisticRegression(max\_iter=1000) 3. base\_model\_2 = RandomForestClassifier() 4. base\_model\_1.fit(X\_train, y\_train) 5. base\_model\_2.fit(X\_train, y\_train)   27.   1. # 3. Make predictions 2. val\_pred\_1 = base\_model\_1.predict(X\_val) 3. val\_pred\_2 = base\_model\_2.predict(X\_val) 4. test\_pred\_1 = base\_model\_1.predict(X\_test) 5. test\_pred\_2 = base\_model\_2.predict(X\_test)   33.   1. # 4. Create meta-features 2. val\_meta\_features = pd.DataFrame({'pred\_1': val\_pred\_1, 'pred\_2': val\_pred\_2}) 3. test\_meta\_features = pd.DataFrame({'pred\_1': test\_pred\_1, 'pred\_2':   test\_pred\_2})  37.   1. # 5. Build and train meta-model (blender) 2. meta\_model = LogisticRegression(max\_iter=1000) 3. meta\_model.fit(val\_meta\_features, y\_val)   41.   1. # 6. Make final predictions and evaluate 2. final\_predictions = meta\_model.predict(test\_meta\_features) 3. blending\_accuracy = accuracy\_score(y\_test, final\_predictions) 45. print(f"Blending Model Accuracy: {blending\_accuracy}")   46.   1. # Visualizations 2. from sklearn.metrics import confusion\_matrix   49.   1. # Confusion Matrix 2. cm = confusion\_matrix(y\_test, final\_predictions) 3. plt.figure(figsize=(8, 6)) 4. sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", 5. xticklabels=['No Disease', 'Disease'], 6. yticklabels=['No Disease', 'Disease']) 7. plt.title('Confusion Matrix of Blending Model') 8. plt.xlabel('Predicted') | |
| 58. | plt.ylabel('Actual') |
| 59. | plt.show() |



Steps to perform Bagging:

* 1. Create multiple datasets from the train dataset by selecting observations with replacements
  2. Run a base model on each of the created datasets independently
  3. Combine the predictions of all the base models to each the final output
  4. !pip install scikit-learn matplotlib seaborn --upgrade
  5. import pandas as pd
  6. from sklearn.model\_selection import train\_test\_split
  7. from sklearn.ensemble import BaggingClassifier, RandomForestClassifier
  8. from sklearn.metrics import accuracy\_score
  9. import matplotlib.pyplot as plt
  10. import seaborn as sns

|  |
| --- |
| 11.   1. # Load the dataset 2. heart\_data = pd.read\_csv('heart.csv')   14.   1. # Prepare data 2. X = heart\_data.drop('target', axis=1) 3. y = heart\_data['target']   18.   1. # Split data into train and test sets 2. X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)   21.   1. # Base model (Random Forest) 2. base\_model = RandomForestClassifier()   24.   1. # Bagging Classifier (using 'estimator') 2. bagging\_model = BaggingClassifier(estimator=base\_model, n\_estimators=10, random\_state=42)   27.   1. # Train the bagging model 2. bagging\_model.fit(X\_train, y\_train)   30.   1. # Make predictions on test set 2. predictions = bagging\_model.predict(X\_test)   33.   1. # Evaluate 2. bagging\_accuracy = accuracy\_score(y\_test, predictions) 3. print(f"Bagging Model Accuracy: {bagging\_accuracy}")   37.   1. # Visualization: Confusion Matrix 2. cm = confusion\_matrix(y\_test, predictions) 3. plt.figure(figsize=(8, 6)) 4. sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", 5. xticklabels=['No Disease', 'Disease'], 6. yticklabels=['No Disease', 'Disease']) 7. plt.title('Confusion Matrix of Bagging Model') 8. plt.xlabel('Predicted') 9. plt.ylabel('Actual') 10. plt.show() |



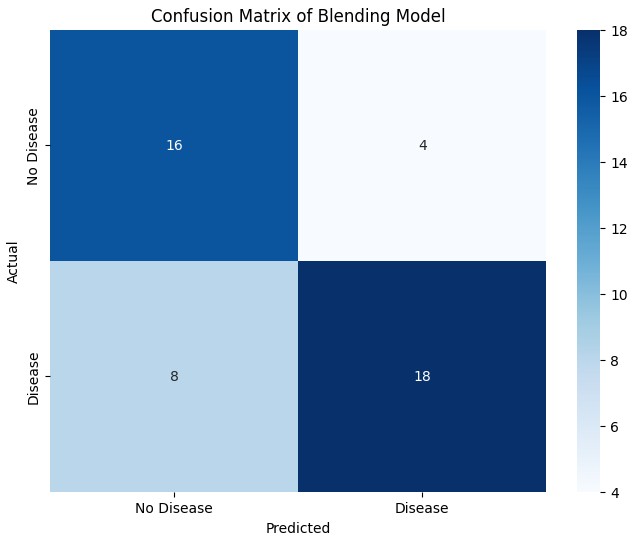
Steps to perform Boosting:

* 1. Take a subset of the train dataset.
  2. Train a base model on that dataset.
  3. Use third model to make predictions on the whole dataset.
  4. Calculate errors using the predicted values and actual values.
  5. Initialize all data points with same weight.
  6. Assign higher weight to incorrectly predicted data points.
  7. Make another model, make predictions using the new model in such a way that errors made by the previous model are mitigated/corrected.
  8. Similarly, create multiple models–each successive model correcting the errors of the previous model.
  9. The final model (strong learner) is the weighted mean of all the previous models (weak learners)

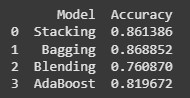
|  |  |
| --- | --- |
| 10. | !pip install scikit-learn matplotlib seaborn |
| 11. | import pandas as pd |

|  |
| --- |
| 1. from sklearn.model\_selection import train\_test\_split 2. from sklearn.linear\_model import LogisticRegression 3. from sklearn.ensemble import RandomForestClassifier 4. from sklearn.metrics import accuracy\_score 5. import matplotlib.pyplot as plt 6. import seaborn as sns   18.   1. # Load the dataset 2. heart\_data = pd.read\_csv('heart.csv')   21.   1. # Prepare data 2. X = heart\_data.drop('target', axis=1) 3. y = heart\_data['target']   25.   1. # Split data into train, validation, and test sets 2. X\_train, X\_temp, y\_train, y\_temp = train\_test\_split(X, y, test\_size=0.3, random\_state=42) 3. X\_val, X\_test, y\_val, y\_test = train\_test\_split(X\_temp, y\_temp, test\_size=0.5, random\_state=42)   29.   1. # Base models 2. base\_model\_1 = LogisticRegression(max\_iter=1000) 3. base\_model\_2 = RandomForestClassifier()   33.   1. # Train base models on the training set 2. base\_model\_1.fit(X\_train, y\_train) 3. base\_model\_2.fit(X\_train, y\_train)   37.   1. # Make predictions on validation and test sets 2. val\_pred\_1 = base\_model\_1.predict(X\_val) 3. val\_pred\_2 = base\_model\_2.predict(X\_val) 4. test\_pred\_1 = base\_model\_1.predict(X\_test) 5. test\_pred\_2 = base\_model\_2.predict(X\_test)   43.   1. # Create meta-features for validation and test sets 2. val\_meta\_features = pd.DataFrame({'pred\_1': val\_pred\_1, 'pred\_2': val\_pred\_2}) 3. test\_meta\_features = pd.DataFrame({'pred\_1': test\_pred\_1, 'pred\_2':   test\_pred\_2})  47.   1. # Meta-model (blender) 2. meta\_model = LogisticRegression(max\_iter=1000)   50. |

|  |  |
| --- | --- |
| 51. | # Train meta-model on validation set |
| 52. | meta\_model.fit(val\_meta\_features, y\_val) |
| 53.  54. | # Make final predictions on test set |
| 55. | final\_predictions = meta\_model.predict(test\_meta\_features) |
| 56.  57. | # Evaluate |
| 58. | blending\_accuracy = accuracy\_score(y\_test, final\_predictions) |
| 59. | print(f"Blending Model Accuracy: {blending\_accuracy}") |
| 60.  61. | # Visualization: Confusion Matrix |
| 62. | cm = confusion\_matrix(y\_test, final\_predictions) |
| 63. | plt.figure(figsize=(8, 6)) |
| 64. | sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", |
| 65. | xticklabels=['No Disease', 'Disease'], |
| 66. | yticklabels=['No Disease', 'Disease']) |
| 67. | plt.title('Confusion Matrix of Blending Model') |
| 68. | plt.xlabel('Predicted') |
| 69. | plt.ylabel('Actual') |
| 70. | plt.show() |



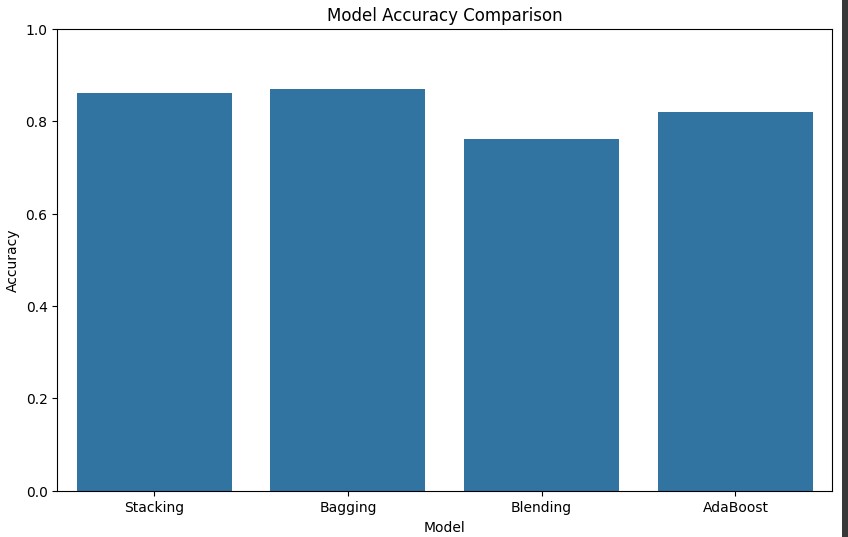
|  |
| --- |
| import pandas as pd  data = {'Model': ['Stacking', 'Bagging', 'Blending', 'AdaBoost'],  'Accuracy': [stacking\_accuracy, bagging\_accuracy, blending\_accuracy, accuracy]}    comparison\_table = pd.DataFrame(data) print(comparison\_table) |



import matplotlib.pyplot as plt

import seaborn as sns

plt.figure(figsize=(10, 6)) sns.barplot(x='Model', y='Accuracy', data=comparison\_table) plt.title('Model Accuracy Comparison') plt.xlabel('Model') plt.ylabel('Accuracy') plt.ylim(0, 1) # Set y-axis limits for better visualization plt.show()



Dataset:

* 1. Dataset from the UCI repository: Alcohol QCM Sensor Donated on 7/21/2019 <https://archive.ics.uci.edu/dataset/496/alcohol+qcm+sensor+dataset>
  2. Dataset from Kaggle: Cardiac features of patients from the "heart.csv" dataset:

<https://www.kaggle.com/datasets/arezaei81/heartcsv>

**OBSERVATIONS / DISCUSSION OF RESULT:**

1. Compare the results of Basic ensemble methods and the Advanced ensemble methods?

**CONCLUSION:**

Based on the results, discuss the conclusions; describe the meaning of the experiment and the implications of your results.

**REFERENCES:**

**(List the references as per format given below and citations to be included the document)**

1. Ethem Alpaydın, “Introduction to Machine Learning”, 4th Edition, The MIT Press, 2020.
2. Peter Harrington, “Machine Learning in Action”, 1st Edition, Dreamtech Press, 2012.
3. Tom Mitchell, “Machine Learning”, 1st Edition, McGraw Hill, 2017.
4. Andreas C, Müller and Sarah Guido, “Introduction to Machine Learning with Python: A Guide for Data Scientists”, 1st Edition, O'reilly, 2016. 5. Kevin P. Murphy, “Machine Learning: A Probabilistic Perspective”, 1st Edition, MIT Press, 2012.

**Website References:**

Ensemble Learning Techniques Tutorial. Available Online: Ensemble Learning Techniques Tutorial, https://www.kaggle.com/code/pavansanagapati/ensemble-learning-techniques-tutorial

Ensembles: Gradient boosting, random forests, bagging, voting, stacking. Available Online:

<https://scikit-learn.org/stable/modules/ensemble.html>