Ensemble Techniques Interview Questions Practical

(Practice Project)

PW SKILLS

Easy Questions

1.You are given a dataset. Write code to load the dataset into your notebook. Print the size of the dataset and also frame a business problem on your own for classification tasks.

(**Hint**: This dataset contains various features that leads to employee attrition in an organization you need to decide the target variable such that you can predict employee attrition based on the given relevant information)

Dataset link:

https://drive.google.com/file/d/1RTEG-DQVey-GRWfgCE76BsmneTRn2ZDi/view?usp=drive_link

Ans: Code to load my dataset into notebook we need to import pandas and then under read_csv method give the path in case we are using collab we will first mount our drive and copy the path in the read_csv module we give an r prefix upfront so it tells Python to treat the backslashes as literal characters, so \n will not be interpreted as a newline character.

Code block:

```
import pandas as pd

# Load the dataset
df = pd.read_csv(r"/content/drive/MyDrive/Employee Attrition
DATA/HR-Employee-Attrition.csv")

# Print the size of the dataset
print(f"Dataset Size: {df.shape[0]} rows and {df.shape[1]} columns")
```

```
Dataset Size: 1470 rows and 35 columns
```

Business problem:
Our goal in this classification problem is to predict, given a variety of parameters like work-life balance, pay, performance reviews, and job satisfaction, whether an employee will leave the company (attrition).

2.When will you use ensemble techniques for the given dataset?. Write one line code for train test split and demonstrate the importance of this split and its relationship with the term data leakage.

Ans: When we wish to use the strengths of numerous models to increase our model's predictive performance, ensemble techniques come in handy. For this dataset, we ought to think about applying ensemble techniques if:

Model Performance: We wish to increase accuracy, decrease variance, or strengthen predictions from the initial models (such as Decision Trees and Logistic Regression), which perform somewhat well.
Overfitting: By averaging or combining predictions, ensemble techniques like Random Forest or Gradient Boosting can assist in decreasing overfitting when individual models overfit the training set.
Complex Data: When a dataset contains non-linear correlations or complex patterns that are difficult for single models to explain.

Train-Test Split Code:

Here's a one-liner code to perform the train-test split using scikit-learn:

```
from sklearn.model_selection import train_test_split

# Assuming 'df' is your DataFrame and 'Attrition' is the target
variable
X_train, X_test, y_train, y_test =
train_test_split(df.drop('Attrition', axis=1), df['Attrition'],
test_size=0.2, random_state=42)
```

Train-Test Split and Data Leakage: How Important Are They?
Train-test split is important because it lets us assess how well our model performs on unknown data, simulating real-world situations. We may prevent overfitting and obtain a more accurate measure of model performance by dividing your data into training and testing sets so that the model is trained on one part and assessed on another.

When data from outside the training dataset is used to build the model, it is known as data leakage and results in too optimistic performance during training but poor generalization to new data. By keeping training and testing datasets apart, a correct train-test split helps prevent data leaking by stopping the model from unintentionally learning from information it shouldn't have access to during training.

3. **Do you think encoding is necessary for any variable in this dataset? If yes, give me the code for data encoding.**

Ans: Indeed, encoding is required for the dataset's categorical variables, which indicate discrete categories as opposed to continuous numerical values. Categorical variables need to be translated into a numerical representation since machine learning algorithms usually need numerical input.

Typical Encoding Methods

- **Label Encoding:** Gives every category a distinct number (for example, "Male" → 0, "Female" → 1).
- By using One-Hot Encoding, binary columns are created for every category, such as "Department" → "Department_Sales", "Department_HR", and so on.

Code for Data Encoding:

```
from sklearn.preprocessing import LabelEncoder, OneHotEncoder

# Example: Label Encoding for binary categorical variables
label_encoder = LabelEncoder()

# Encode binary variables (e.g., 'Attrition' and 'Gender')
df['Attrition'] = label_encoder.fit_transform(df['Attrition'])  #
Yes=1, No=0
df['Gender'] = label_encoder.fit_transform(df['Gender'])  # Male=1,
Female=0

# Example: One-Hot Encoding for multi-class categorical variables
(e.g., 'Department' and 'JobRole')
df = pd.get_dummies(df, columns=['Department', 'JobRole'],
drop_first=True)
```

4.What are the steps that you have implemented in the data preprocessing. Write code to create a pipeline of the same

Ans: The steps that I have implemented is shown in the pipeline below:

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, OneHotEncoder, StandardScaler
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline

# Identify categorical and numerical columns
categorical_cols = ['Gender', 'Department', 'JobRole']  # example categorical
columns
numerical_cols = df.select_dtypes(include=['int64',
'float64']).columns.tolist()

# Remove the target column 'Attrition' from numerical_cols if present
if 'Attrition' in numerical_cols:
    numerical_cols.remove('Attrition')

# Preprocessing for categorical data
categorical_transformer = Pipeline(steps=[
    ('onehot', OneHotEncoder(drop='first'))  # One-Hot Encoding and drop first
to avoid multicollinearity
])

# Preprocessing for numerical data (if any normalization or scaling is needed)
numerical_transformer = Pipeline(steps=[
    ('scaler', StandardScaler())  # StandardScaler to normalize numerical
features
])

# Combine preprocessing steps for both categorical and numerical data
preprocessor = ColumnTransformer(
    transformers=[
        ('num', numerical_transformer, numerical_cols),
        ('cat', categorical_transformer, categorical_cols)
    ])

# Define the full pipeline
pipeline = Pipeline(steps=[
    ('preprocessor', preprocessor)
])

# Target variable encoding
label_encoder = LabelEncoder()
df['Attrition'] = label_encoder.fit_transform(df['Attrition'])  # Encode the
target variable (Attrition)

# Split the data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(df.drop('Attrition',
axis=1), df['Attrition'], test_size=0.2, random_state=42)

# Fit and transform the training data using the pipeline
X_train_transformed = pipeline.fit_transform(X_train)

# Transform the test data using the pipeline
X_test_transformed = pipeline.transform(X_test)

# Display the transformed training data shape
print(f"Transformed X_train shape: {X_train_transformed.shape}")
```

```
Transformed X_train shape: (1176, 37)
```

categorical_cols: Recognises the columns that have category information in them.

numerical_cols: Identifies columns (apart from the target variable) that hold numerical data.
Preparing Categorical Data:

We use One-Hot Encoding on categorical variables. Dropping the first category using the drop='first' argument avoids multicollinearity.

Numerical features are normalized using StandardScaler, guaranteeing that each feature contributes equally to the model.

Combining the Steps in Preprocessing:Several preprocessing methods can be simultaneously applied to numerical and category columns using ColumnTransformer.

Pipeline Establishment: The categorical and numerical transformations are part of the preprocessing stage that creates the entire pipeline.

Encoding the target: Since the target variable (attrition) is a binary classification issue, it is label-encoded.

Train-Test Division: An 80-20 split of the dataset is made into training and testing sets.
The pipeline is fit on the training data and then used to transform both the training and test data.

5.What is a package? Write code to import any ensemble technique from scikit-learn package.

Ans: Packages, especially for Python, are groups of modules with specialized functions arranged in directories. Using packages makes it easier to handle larger projects by allowing us to easily organize and reuse code. They frequently come with pre-made tools, classes, and methods that make difficult jobs like web development, machine learning, and data manipulation easier.

Python packages can be imported into your code using the import statement and are typically installed using a package manager such as pip.

Example of Importing an Ensemble Technique from Scikit-learn:

```
from sklearn.ensemble import RandomForestClassifier

# Initialize the model
model = RandomForestClassifier(n_estimators=100, random_state=42)

# You can now use the 'model' variable to fit and predict on your dataset
```

6.Write code to define x and y such that you can have your x_train and x_test. Write code to view the first 5 rows of x_train and last 5 rows of y_test.

Ans:

```
from sklearn.model_selection import train_test_split


# Define X (features) and y (target)
X = df.drop('Attrition', axis=1)  # Drop the target column from the
features
y = df['Attrition']  # Target variable

# Split the data into training and testing sets (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# View the first 5 rows of X_train
print("First 5 rows of X_train:")
print(X_train.head())

# View the last 5 rows of y_test
print("\nLast 5 rows of y_test:")
print(y_test.tail())
```

7.In doing train_test_split why do we introduce the random_state parameter. Change the random_state from 42 to 0 and explain the significance of this parameter.

Ans: The data is shuffled before being divided into training and testing sets using the random_state argument in train_test_split. Stated differently, it guarantees the reproducibility of the split. Using the same random_state for multiple runs ensures that you always obtain the same train-test split, which is essential for testing, debugging, and comparing models.

```
from sklearn.model_selection import train_test_split

# Split the data with random_state=42
X_train_42, X_test_42, y_train_42, y_test_42 = train_test_split(X, y,
test_size=0.2, random_state=42)

# Split the data with random_state=0
X_train_0, X_test_0, y_train_0, y_test_0 = train_test_split(X, y,
test_size=0.2, random_state=0)

# Print the first 5 rows of X_train for both random_state=42 and
random_state=0
print("First 5 rows of X_train with random_state=42:")
print(X_train_42.head())

print("\nFirst 5 rows of X_train with random_state=0:")
print(X_train_0.head())

# Print the first 5 rows of y_train for both random_state=42 and
random_state=0
print("\nFirst 5 rows of y_train with random_state=42:")
print(y_train_42.head())

print("\nFirst 5 rows of y_train with random_state=0:")
print(y_train_0.head())
```

Two Different Splits: The code splits the dataset twice, once with random_state=42 and once with random_state=0.

Comparing Results: By printing the first 5 rows of X_train and y_train for both splits, we can see that the data in these sets is different, demonstrating how changing random_state alters the train-test split.
Significance:

Reproducibility: We make sure that the dataset is split uniformly each time the function is executed by defining a random_state. This is crucial for maintaining consistency, particularly when contrasting the output of various models or variants of the same model.

Fair Comparison: Having the same training and testing data is crucial when working with various machine learning methods. The random state makes sure that varied data splits don't skew the fairness of the model-to-model comparison.

Debugging: Using a fixed random_state makes it easier to recreate and debug problems with your model in the event that they arise.

Medium Questions:

8.Tell me when to choose bagging , boosting and stacking and here what is the bagging algorithm that you can implement to get better results. Write code to import and create an instance of the bagging technique that you have chosen.

Ans: Bagging:for example Random Forest: It is applied when one wants to reduce the variance and overfitting of high-variance models, such as decision trees.

Boosting: for example, XGBoost, AdaBoost; this would be applied when one needs to reduce bias and improve accuracy because it corrects the mistakes of weak learners.

Stacking: This will be used where one wants to combine several models' predictions to get a good performance.

Bagging Algorithm: Random Forest instance creation

```
from sklearn.ensemble import RandomForestClassifier
# Create an instance of the RandomForestClassifier
rf_classifier = RandomForestClassifier(n_estimators=100,
random_state=42)
```

9.Suppose you and your friend have decided to participate in a hackathon and the problem statement and dataset is the one that you have framed in Question 1. Now your friend is using a decision tree and you are using a random forest. Tell me whose model will have a higher chance of winning the competition and why. And also show the parameters that you have passed in your random forest model.

Ans: When comparing a Decision Tree model and a Random Forest model, the Random Forest model generally has a higher chance of performing better and thus winning the competition. This is because:

4. **Ensemble Learning:** Random Forest is an ensemble method that builds multiple decision trees (each trained on different subsets of the data) and averages their predictions. This reduces the variance and prevents overfitting, which is a common issue with standalone Decision Trees.
5. **Generalization:** Random Forests tend to generalize better on unseen data because they reduce the impact of noisy or outlier data by averaging the predictions of multiple trees. Decision Trees, on the other hand, can easily overfit, especially if the tree is deep.
6. **Robustness:** Random Forests are more robust to overfitting compared to Decision Trees, especially in complex datasets. Given that hackathons usually involve test datasets that we haven't seen before, the Random Forest model is likely to perform better.

Parameters Passed in Random Forest Model:
from sklearn.ensemble import RandomForestClassifier

 # Initialize the RandomForestClassifier

```
rf_classifier = RandomForestClassifier(
    n_estimators=100,          # Number of trees in the forest
    max_depth=None,            # Maximum depth of the tree (None means
nodes are expanded until all leaves are pure)
    min_samples_split=2,       # Minimum number of samples required to
split an internal node
    min_samples_leaf=1,        # Minimum number of samples required to be
at a leaf node
    max_features='auto',       # Number of features to consider when
looking for the best split
    bootstrap=True,            # Whether bootstrap samples are used when
building trees
    random_state=42,           # Seed used by the random number generator
    n_jobs=-1                  # Use all available cores for computation)
```

10. **Is it always advisable to go for random forest like you did in your hackathon. Tell me the major drawback of random forest over decision trees. Write your code which you wrote for training your random forest on training data. Also write the code for predicting on test data.**

Ans: While Random Forest is an excellent and widely deployed model, it's not the best fit in every situation. There are some disadvantages of Random Forests compared to decision tree :

Complexity: The Random Forest models are more complex than Decision Trees and, therefore, much heavier in computation. This means taking longer training and prediction times, especially when great volumes of data are used or the number of trees combined is large.

Interpretability - Decision Trees are more interpretable and visual. It is straightforward to tell the rules for which the tree is considering to make any prediction. In contrast, Random Forests- as an ensemble of lots of trees- are hard to interpret.

Slower Prediction: Since Random Forest has to make the prediction over several trees, this might be slower; whereas a single Decision Tree could give quicker predictions. This can also be a disadvantage if there is a need for real-time predictive power in certain applications. Memory Usage: This is more in Random Forests compared to Decision Trees, as it keeps a number of trees in memory at a given time. This tends to be a weakness with large datasets more often.

Code for training the random forest:

```
# Train the model
 rf_classifier.fit(X_train, y_train)
```

Code for predicting on Test data:

```
# Make predictions on the test set
y_pred = rf_classifier.predict(X_test)
```

11. **What is the difference between estimator and n_estimator parameters? Write code to set n_estimators to 100. How do you decide this number?**

Ans: Difference between estimator vs n_estimators:

estimator: This is the parameter that in ensemble methods like Bagging or Stacking, one can specify what kind of base model-DecisionTreeClassifier or LogisticRegression for example-will be used for the individual learners. It defines the modeling type which shall be used in the ensemble.

n_estimators: This would be a parameter for the ensemble models, which include Random Forest, Bagging, and Boosting. This option allows one to define how many base learners are to be generated in the ensemble. In a Random Forest, this parameter would show the size of the forest, or how many trees in the forest.

Code to Set n_estimators to 100:

```
from sklearn.ensemble import RandomForestClassifier

# Create an instance of RandomForestClassifier with n_estimators set
to 100
rf_classifier = RandomForestClassifier(n_estimators=100,
random_state=42)

# Train the model (assuming X_train and y_train are already defined)
rf_classifier.fit(X_train, y_train)
```

How do we decide the number of estimators (n_estimators)?

1. **Performance Consideration:** Increasing the number of estimators generally improves the model's performance by reducing variance and overfitting. However, after a certain point, the performance gains diminish, and we may experience diminishing returns.

2. **Computational Resources:** More estimators require more computational resources (memory and time). We need to balance the number of estimators with the available computational resources. A common choice is 100 estimators, but this can be adjusted based on the hardware and time constraints.

3. **Cross-Validation:** Use cross-validation to experiment with different values of n_estimators and select the one that provides the best performance on the validation set.

4. **Problem Complexity:** For more complex problems or larger datasets, we might need more estimators to capture the underlying patterns.

12. **In your code you have kept your criterion param as entropy and write code to set it to gini. What is the possible difference and why did you go with entropy instead of gini? What is the default param?**

Ans: In a Random Forest or Decision Tree model, you can set the criterion parameter to gini for using the Gini impurity measure:

```
from sklearn.ensemble import RandomForestClassifier

# Create an instance of RandomForestClassifier with criterion set to
'gini'
rf_classifier = RandomForestClassifier(n_estimators=100,
criterion='gini', random_state=42)

# Train the model (assuming X_train and y_train are already defined)
rf_classifier.fit(X_train, y_train)
```

Differences

Entropy: As a result of the logarithmic calculations, entropy is more computationally expensive than Gini.

When to use: If we are looking for more "information" in terms of how our tree splits, due to Information Gain, you want to use entropy.

Gini: Gini is computationally faster than entropy because it does not involve any logarithmic computations. When to use: If we are more concerned with speed and performance and less concerned with the theoretical implications of entropy, Gini is generally a good choice.

Why Entropy and not Gini?

I've chosen entropy initially, as in most cases, it is more interpretable in terms of Information Gain - the measure is useful for explaining how much information a feature contributes to the prediction of the target. It might be pretty useful for understanding how splits in a decision tree contribute to reducing uncertainty in the classification tasks.

Gini is preferred because of its ease of computation and sometimes results in very similar performance.
The default criterion for RandomForestClassifier and DecisionTreeClassifier is gini.

13. **Write code to print train accuracy and test accuracy. Can ensemble techniques ever give high variance in train and test accuracies. If no, tell me why or if yes then when?**

Ans: Here's a Python code snippet using RandomForestClassifier to print the training and testing accuracies:

```
# Train the model
 rf_classifier.fit(X_train, y_train)
# Predict on training and testing data
train_predictions = rf_classifier.predict(X_train)
test_predictions = rf_classifier.predict(X_test)
# Calculate and print train and test accuracy
train_accuracy = accuracy_score(y_train, train_predictions)
test_accuracy = accuracy_score(y_test, test_predictions)

print(f"Training Accuracy: {train_accuracy * 100:.2f}%")
print(f"Testing Accuracy: {test_accuracy * 100:.2f}%")
```

```
Training Accuracy: 100.00%
Testing Accuracy: 86.62%
```

Yes, sometimes ensemble techniques will show a high variance between the train and test accuracies, though less likely compared to single models like decision trees.

Reason:
The overall intuition with ensemble methods, such as Random Forests or Boosting Techniques, is to reduce the variance and overfitting by averaging multiple model predictions. However, there are specific scenarios where high variance in train and test accuracies still exists:

Overfitting through Model Complexity:
Boosting algorithms, such as Gradient Boosting, will overfit if the model is too complex - for example, too many boosting iterations, or a very high learning rate or if the base learners are too strong - such as deep trees. In such cases, the accuracy on the training set may be very high, but the model does not generalize, hence yielding poor test accuracy.

Insufficient Regularization:
Also, if regularization is absent in the ensemble method-that is, no limit to a maximum depth and no minimum number of samples per leaf-it may overfit the training data, hence record high training accuracy versus lower test accuracy.

Poor quality of data:
With non-representative training data to the test data, including things like different distributions and high noise, even an ensemble method could overfit on the training data, resulting in high variance over train and test accuracies. Imbalanced Datasets This could mean that in highly imbalanced data, if the model is not properly adjusted-think class weights or SMOTE-a high score on the ensemble might mean good performance on the majority class during training with poor generalization on that small minority class, hence a poor test accuracy.

Hard Questions:

14. **What is data drift ? Can your employee attrition data ever go through this scenario if yes tell me when?. Write code for importing accuracy score from sklearn and also demonstrate accuracy score without the help of sklearn generating a synthetic dataset.**

Ans: Data drift is the change in the statistical properties of data over time, thus leading to degradation in model performance. In other words, the relationship between input data and target variables changes; hence, predictions by that model are no longer accurate. Such drift might be due to changes in the underlying distribution of data, business processes, or external factors, or maybe even the way data is collected.

Yes, employee attrition data could undergo data drift. It might occur under several conditions:

Change in Company Policies: The factors contributing to employee attrition changed in the case of policy changes and characteristics of data would drift away from the original training data.

Economic or Market Changes: Contraction and changes in the economy and industries are external factors that may cause a change in employee behavior or their intention or propensity to leave a company, thus producing data drift.

Organizational Changes: Changes due to merger and acquisition or a change in leadership is also one of the reasons for a change in employees' sentiment, hence causing drift in the data distribution.

Technological Changes: Adoption of any new technology or process that may have an effect on the job role and, subsequently, an impact on job satisfaction and, finally, the attrition pattern.

Code for Importing Accuracy Score from sklearn:

```
from sklearn.metrics import accuracy_score
```

Code to Demonstrate Accuracy Calculation Without sklearn:

```
import numpy as np

# Generating a synthetic dataset
# Let's assume we have 100 predictions and actual labels
np.random.seed(42)
y_true = np.random.randint(0, 2, 100)  # Actual labels (0 or 1)
y_pred = np.random.randint(0, 2, 100)  # Predicted labels (0 or 1)

# Calculate accuracy without sklearn
def calculate_accuracy(y_true, y_pred):
    correct_predictions = np.sum(y_true == y_pred)
    accuracy = correct_predictions / len(y_true)
    return accuracy

# Using the custom function to calculate accuracy
manual_accuracy = calculate_accuracy(y_true, y_pred)
print(f"Manual Accuracy: {manual_accuracy * 100:.2f}%")

# For comparison, using sklearn's accuracy_score
sklearn_accuracy = accuracy_score(y_true, y_pred)
print(f"Sklearn Accuracy: {sklearn_accuracy * 100:.2f}%")
```

```
Manual Accuracy: 50.00%
Sklearn Accuracy: 50.00%
```

15. When do we use F1 score , precision and recall. Write code to implement them. Tell me the importance of domain knowledge in solving any business problem.

Ans: Precision refers to the ratio of true positive predictions versus all the positive predictions of a model. It is used when there is a high cost associated with a false positive. For example, spam detection in emails involves precision because you would not want your important email to be marked as spam by mistake.

Recall: The ratio of true positive predictions to the overall actual number of positives. It's applied when the cost of a false negative is very high. In disease diagnosis, its recall is also very crucial since failure to identify a positive case of a disease will lead to dire consequences.

F1-score is the harmonic mean of precision and recall. It is used in situations where a real balance between precision and recall exists or when one wants a single metric to reflect both false positives and false negatives. This can be highly useful in imbalanced datasets where one class predominates highly over another.

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import precision_score, recall_score, f1_score,
accuracy_score

# Initialize the Random Forest classifier
rf_classifier = RandomForestClassifier(random_state=42)

# Train the model on the transformed training data
rf_classifier.fit(X_train_transformed, y_train)
```

```
# Make predictions on the transformed test data
y_pred = rf_classifier.predict(X_test_transformed)

# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy * 100:.2f}%")

# Calculate precision
precision = precision_score(y_test, y_pred)
print(f"Precision: {precision * 100:.2f}%")

# Calculate recall
recall = recall_score(y_test, y_pred)
print(f"Recall: {recall * 100:.2f}%")

# Calculate F1 score
f1 = f1_score(y_test, y_pred)
print(f"F1 Score: {f1 * 100:.2f}%")
```

```
Accuracy: 87.07%
Precision: 66.67%
Recall: 5.13%
F1 Score: 9.52%
```

Domain knowledge helps in precise problem identification, selection of relevant features, and selection of appropriate models to solve business problems. It enhances data quality and interpretation, with assurance that technical solutions meet the targets of the business. Besides, it allows for better communication with stakeholders, rallying practical and innovative solutions.

16. **Write code to get your test accuracy score in two decimal points and also convert it into percentage. Suppose your accuracy score is 100 percent. What can be the reason behind it. What steps will you take to mitigate this or will you push the model into production.**

Ans:

```
from sklearn.metrics import accuracy_score

# Assuming y_test and test_predictions are already defined
accuracy = accuracy_score(y_test, test_predictions)

# Convert accuracy to percentage and round to two decimal places
accuracy_percentage = round(accuracy * 100, 2)

print(f"Test Accuracy: {accuracy_percentage}%")
```

If the score for accuracy is 100%:

Potential Causes:

Overfitting: The model may not be generalizing adequately to new, unknown data because it is overfit to the training set, capturing noise or patterns unique to the training set.

Data Leakage: Data leakage is the possibility that during training, information from the test set or target variable was unintentionally used.

Simple challenge: A perfect score could indicate that the challenge is too simple or that the dataset is too clear-cut and easy.

Actions to Reduce:

Cross-Checking: To see if the model performs consistently across several data splits, use cross-validation.

Regularization: To avoid overfitting, use regularization strategies (such as L1, L2).

Verify for Data Leaks: Make sure that no training has utilized the test data, and that no information from the target variable has leaked into the training process.

Decision on Production: If overfitting or data leakage is suspected, do not push the model into production. First, address the issues and re-evaluate the model.If the problem is genuinely simple and after thorough validation, the accuracy remains high, the model can be considered for production, but continued monitoring is necessary.

17.What is a decision stump? Do you prefer a decision stump in bagging or boosting more? Implement decision stump in your model and print accuracy. Which parameter have you introduced and what is the use of this parameter?

Ans: A decision tree with only one level is the basis of a decision stump, a basic machine learning model. Stated differently, it represents a tree with a single decision node that divides the data into two leaves. As weak learners in ensemble techniques like bagging and boosting, decision stumps are frequently employed.

Choosing between bagging and boosting:

Boosting: Since decision stumps are frequently weak by nature, they are frequently preferred in boosting approaches (e.g., AdaBoost). This is because boosting focusses on repairing the errors made by weak learners. Numerous decision stumps can be used with boosting to produce a powerful overall model.
Decision stumps can be used in bagging, but because bagging depends more on stronger base models, they are typically more useful in boosting.

```
# Create a decision stump model
decision_stump = DecisionTreeClassifier(max_depth=1, random_state=42)
# Fit the model
decision_stump.fit(X_train, y_train)
# Predict on training and testing data
train_predictions = decision_stump.predict(X_train)
test_predictions = decision_stump.predict(X_test)
 # Calculate accuracy
train_accuracy = accuracy_score(y_train, train_predictions)
test_accuracy = accuracy_score(y_test, test_predictions)
print(f"Training Accuracy: {train_accuracy * 100:.2f}%")
print(f"Testing Accuracy: {test_accuracy * 100:.2f}%")
```

```
Training Accuracy: 83.33%
Testing Accuracy: 85.37%
```

Parameter Introduced: max_depth

- **max_depth=1:** This parameter limits the depth of the decision tree, ensuring that the model only creates a single split, thus behaving as a decision stump.

Use of max_depth Parameter:

- The max_depth parameter is crucial in controlling the complexity of the decision tree. By setting max_depth=1, you enforce the tree to only have one split, making it a decision stump. This simplicity can be useful in boosting, where the goal is to correct the errors of weak models iteratively.

18. When do you use the class_weight parameter ? In your model do you think this param is required. Demonstrate the use of this parameter.

Ans: When working with imbalanced datasets, models employ the class_weight parameter. The model may become biased in favor of the dominant class if one class substantially outnumbers the other or classes, which would result in subpar performance on the minority class. By giving the minority class a higher weight than the other classes, the class_weight parameter aids in this process.

How and When to Use class_weight

Imbalanced datasets: when the representation of one class in relation to others is low.
When misclassification comes at a different cost: For instance, it may be more expensive to incorrectly identify a fraudulent transaction as legitimate in fraud detection than the opposite.

Increasing minority class recall: This parameter assists in balancing the model's attention if you are more interested in discovering the minority class (e.g., disease prediction).

19. Suppose you want to use all processors, what is the param you will introduce and what is the value you will set. Showcase its use and mention its significance.

Ans:

Name of Parameter: n_jobs

To Set Value: -1

The algorithm is instructed to employ all processors available for parallel computing when n_jobs = -1 is set.
Significance:

Speed: Making use of every processor can cut down on computation time considerably, particularly for big datasets or models with a lot of estimators.

Efficiency: By dividing the burden among all cores, parallel processing improves the efficiency of the model training process.

Scalability: It makes handling larger datasets or more complicated models possible by assisting in the training process's acceleration when more CPU cores are used.

```
# Initialize Random Forest Classifier with n_jobs=-1 to use all processors
rf_model = RandomForestClassifier(n_estimators=100, n_jobs=-1,
random_state=42)
```

20.Write code to set bootstrap and oob params to False and True respectively and explain to me what will be the impact now on your model.

Ans: Code to set bootstrap and oob_score parameters:

```
# Initialize Random Forest Classifier with bootstrap=False and
oob_score=True rf_model = RandomForestClassifier(n_estimators=100,
bootstrap=False, oob_score=True, random_state=42)
```

Explanation of the parameters:

bootstrap=False:

Impact: Turns off bootstrapping, such that every tree is trained on the entire dataset using all samples.

Impact:

Less Diversity: Trees are less diverse, reducing model robustness.

Higher Risk of Overfitting: The trees can easily become overly similar, hence increasing the risk of overfitting of the model.

Smaller Variance Reduction: The usual benefit of variance reduction due to averaging is sacrificed.

oob_score=True:

Impact: This option allows you to get the Out-of-Bag score via estimates of model performance using all data that were not included in the bootstrap sample.

Impact:

Irrelevant OOB Score: Since bootstrap = False, no OOB samples are available, rendering the OOB score irrelevant.

Deceptive Performance Estimate: OOB score will not be able to give any valid estimate about generalization error.

Impact on the Model: Without Bootstrapping (bootstrap=False): It increases the risk of overfitting and makes the model less robust since the diversity of trees will be lower. OOB Scoring with oob_score=True and No Bootstrapping In such cases, the OOB score can no longer serve effectively for the assessment of performance.