Data Mining

Project Report

SE421

Mushroom Dataset Classification

DR. Abdelrahman Hedar

Team members:

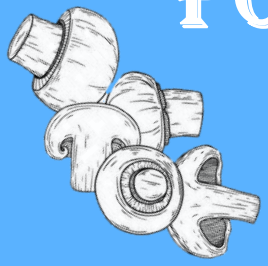
-Nourhan Amr Mohammed

-Nada Salah El-Deen

-Eman Deyaa El-Deen

Department:

Software Engineering



POISONOUS

OR

EDIBLE

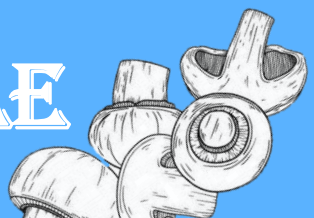


Table of contents

1.Introduction

Classification is a fundamental task in data analysis and machine learning. It involves categorizing data items into predefined classes or categories based on their features or characteristics.

A model is typically trained on a dataset where the classes or labels are already known, and then applied to make predictions on new data.

There are various classification algorithms, such as Logistic Regression, Decision Trees, Random Forests, Support Vector Machines, and Neural Networks, each with its own strengths and weaknesses.

1.1 Define Task

The task in our dataset is to find out whether the type of mushroom is poisonous or not.

By analyzing the input data, the output would be 1 -which means “poisonous”- or zero - which means “edible”-.

We are going to use different type of classification algorithms to find which would provide the best results.

1.2 Dataset description

Dataset name: Mushroom classification dataset.

Source: Kaggle.

Dataset size: the dataset contains 9 attributes and 54035 records, its memory usage is 3.7MB.

Attributes: the 9 columns in the dataset describe various characteristics of the mushrooms as they are:

- cap diameter

-cap shape

-gill attachment

-gill color

-stem height

-stem width

-stem color

-season

-poisonous

There are 2 float columns and 7 int columns.

Target Variable: The target variable is the "poisonous" column, which indicates whether the mushroom is "edible" (45.08% of the dataset) or "poisonous" (54.92% of the dataset).

1.3 Algorithms

Logistic Regression:

- Logistic Regression is a linear model used for binary classification problems, so The output is a probability value between 0 and 1.
- It learns a logistic function to predict the probability of an instance belonging to a particular class.

Decision Trees:

- Decision Trees create a tree-like model of decisions and their possible consequences.
- They recursively split the attribute space based on the attribute that provides the most information gain.
- Decision Trees can handle both numerical and categorical features, and are interpretable.

Random Forests:

- Random Forests are an ensemble learning method that combines multiple Decision Trees and train them on different subsets of the data and features.
- The final prediction is made by averaging the predictions of the individual Decision Trees.
- Random Forests are robust to overfitting and can handle a wide range of feature types.

Support Vector Machines (SVM):

- SVMs are a class of linear models that find the optimal hyperplane to separate different classes.
- They work well with high-dimensional data and can handle non-linear decision boundaries using kernel tricks.
- SVMs are effective for both binary and multi-class classification problems.

k-Nearest Neighbors (k-NN):

- k-NN is a non-parametric, instance-based learning algorithm.
- It classifies a new instance based on the majority vote of its k nearest neighbors in the feature space.

- k-NN is simple to implement and can capture complex decision boundaries, but is sensitive to the choice of k and feature scaling.

Naive Bayes:

- Naive Bayes is a probabilistic classifier based on the Bayes' theorem.
- It makes predictions by assuming that the features are independent given the class.
- Naive Bayes is particularly effective for high-dimensional datasets and is computationally efficient.

Bagging (Bootstrap Aggregating):

- Bagging is an ensemble learning technique that combines multiple base classifiers (e.g., Decision Trees) to improve the overall performance.
- It creates multiple models by training each base classifier on a random subset of the training data (with replacement).
- The final prediction is made by aggregating the predictions of the individual base classifiers, typically through majority voting for classification tasks.

Boosting:

- Boosting is also ensemble learning method that sequentially trains weak learners (e.g., Decision Stumps) to create a strong learner.
- The key idea is to focus on the instances that were misclassified by the previous weak learners, giving them higher weights in the subsequent training rounds.
- Popular boosting algorithms include AdaBoost, Gradient Boosting, and XGBoost.
- Boosting is effective in improving the performance of weak classifiers and can handle a wide range of classification problems.

Neural Networks:

- Neural Networks are a family of machine learning models inspired by the structure and function of the human brain.
- They consist of interconnected nodes (neurons) organized in layers, and can learn complex non-linear relationships.
- Neural Networks, especially deep learning models, have shown impressive performance in various classification tasks, such as image recognition and natural language processing.

2.Methodology

We are going to use 5 different algorithms to apply classification on the dataset, but before it there are two important processes which are data preprocessing and data splitting.

2.1 Algorithms used

1. Decision Tree:
   - They are inherently interpretable, which can provide valuable insights into the most important features for classifying mushrooms as edible or poisonous.
   - The recursive feature splitting in Decision Trees aligns well with the structured nature of the mushroom dataset, where different characteristics (e.g., cap shape, gill color) can be used to distinguish between the classes.
2. Logistic Regression:
   - It is a suitable choice for the binary classification problem of distinguishing between edible and poisonous mushrooms.

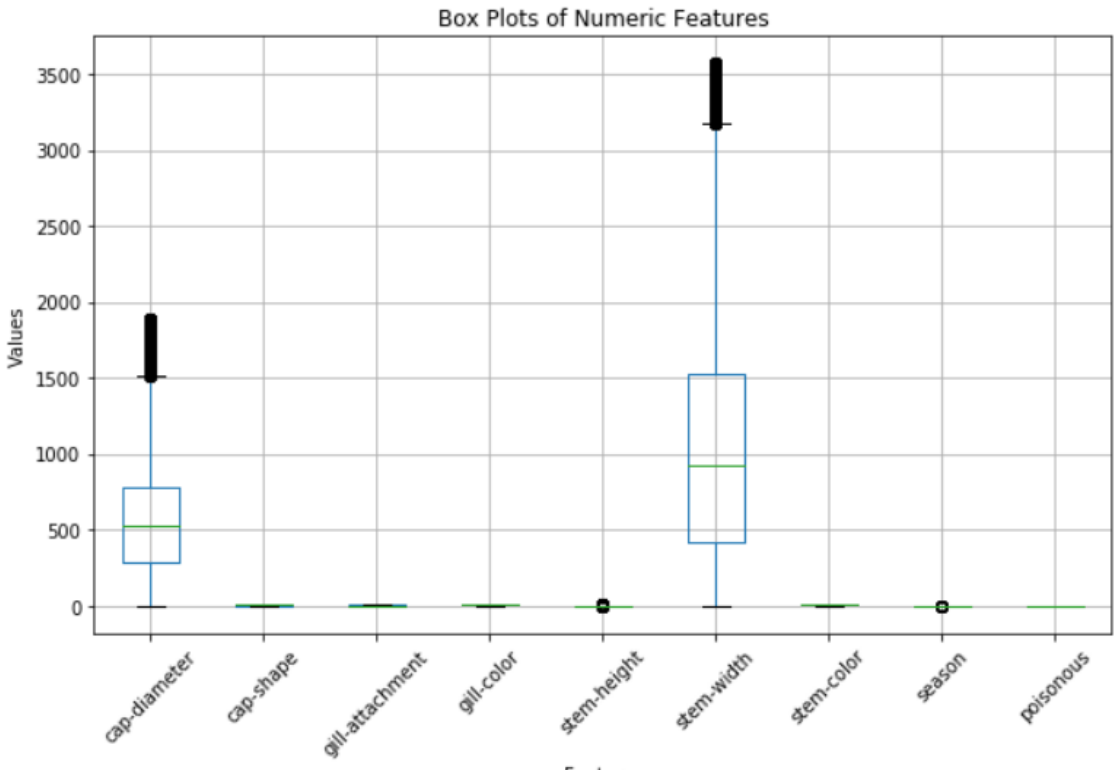
- The linear nature of Logistic Regression may capture the underlying relationships between the mushroom features and the target class effectively.

3. Random forest:
   - The ensemble nature of Random Forests can help capture complex patterns and non-linear relationships in the data, which may not be easily captured by a single Decision Tree.
   - Random Forests are known for their robustness to overfitting, which is important for the mushroom dataset, as it may contain some noisy or irrelevant features.
4. Bagging:
   - Bagging can help reduce the variance of the base classifiers (e.g., Decision Trees) and make the overall model more robust to overfitting.
5. Boosting:
   - Boosting, especially algorithms like AdaBoost or Gradient Boosting, can effectively handle the mushroom dataset by sequentially training weak learners and focusing on the misclassified instances.
   - It can capture complex non-linear relationships in the data, which may not be easily captured by a single Decision Tree or Logistic Regression model.

2.2 Data preprocessing

Problems in the dataset and its solutions:

1. Problem: there are some outliers specially in “Cap-diameter” and “stem-width”.



Solution: on applying Z-score way, it detected the outliers, so we dropped them.
It was a total of “3827” outlier.

2. Problem: there are some duplicates in the dataset.
   Solution: after detecting the duplicates, we dropped them from the data.
   It was a total of “250” duplicate.

2.3 Data splitting

We used "train-test-split" from "Sklearn.model selection" to split the data, "y" for "poisonous" class and "x" for the rest attributes.

we applied "pd.crosstab" to examine the relationship between attributes with respect to "poisonous" class, it shows that its possible to predict if the mushroom is poisonous from the rest of the attributes.

3.Experimental Simulation

3.1 Environment

- Language: Python

- Environment: Jupyter Notebook

- Notebook details and steps: we started by uploading the dataset then loading it inside the notebook.

- Data Exploration: 1.Summary Statistics 2.Data Visualization
- data preprocessing: 1.Missing values 2.Outliers 3.Duplicates
- Classification: 1.Data splitting 2.Applying algorithms

3.2 Methods

3.2.1 Logistic regression (Baseline model)

By using SKlearn logistic regression lib. We applied the model on the dataset.

First we applied the default parameters, the results is:

- the mean is 0.3779993353273513, which means that the Logistic Regression model is making errors with an average squared difference of 0.3767 between the actual and predicted values.

- the accuracy score is 0.6220006646726487, which means that the logistic regression model correctly predicted the class label for 62.33% of the instances in the test set.

- the confusion matrix shows that the model made 3678 true positive predictions, 3195 true negative predictions, 2492 false positive predictions, and 5680 false negative predictions.

- Precision for the class labeled as "0" is 0.60. This means that 60% of the instances the model predicted as class "0" were actually class "0".

Precision for the class labeled as "1" is 0.64. This means that 64% of the instances the model predicted as class "1" were actually class "1".

- Recall for the class "0" is 0.54. This means the model correctly identified 54% of the actual class "0" instances.

Recall for the class "1" is 0.70. This means the model correctly identified 70% of the actual class "1" instances.

- F1-Score for the class "0" is 0.56, which is the harmonic mean of precision and recall.

F1-Score for the class "1" is 0.67, which is the harmonic mean of precision and recall.

We applied hyperparameter tuning by the params:

'C': [0.1, 1, 10],

'penalty': ['l1', 'l2'],

'solver': ['liblinear', 'newton-cg', 'sag'],

'max_iter': [100, 200, 300]

The Best hyperparameters: 'C': 0.1, 'max_iter': 100, 'penalty': 'l2', 'solver': 'newton-cg'

After hyperparameter tuning:

- Test Accuracy: 0.6493851777999335

- Mean Squared Error: 0.35061482220006646

- the confusion matrix: 3842 true positive predictions, 3031 true negative predictions, 2244 false positive predictions, and 5928 false negative predictions.

3.2.2 Decision tree

We used entropy criterion with maxdepth = 3 (default) to create the decision tree model.

First we applied the default parameters, the results are:

- the mean is 0.3524759056164839, which means that, on average, the predicted target values are off by a squared difference of 0.352 from the actual target values.

- the accuracy is 0.6475240943835161, which means that the Decision Tree Classifier correctly predicted the target variable 64.75% of the time.

-The confusion matrix shows the number of true positive (5145), false positive (3575), false negative (1728), and true negative (4597) predictions made by the Decision Tree Classifier.

-Precision for the class labeled as "0" is 0.59. This means that 59% of the instances the model predicted as class "0" were actually class "0".

Precision for the class labeled as "1" is 0.73. This means that 73% of the instances the model predicted as class "1" were actually class "1".

- Recall for the class "0" is 0.75. This means the model correctly identified 75% of the actual class "0" instances.

Recall for the class "1" 0.56. This means the model correctly identified 56% of the actual class "1" instances.

- F1-Score for the class "0" is 0.66, which is the harmonic mean of precision and recall.

F1-Score for the class "1" is 0.63, which is the harmonic mean of precision and recall.

We applied hyperparameter tuning by the params:

'criterion': ['gini', 'entropy'],

'max_depth': [2, 3, 4, 5, 6, 7, 8, 9, 10, 12],

'min_samples_split': [2, 5, 10, 12],

'min_samples_leaf': [1, 2, 4, 6],

'max_features': ['auto', 'sqrt', 'log2']

The Best hyperparameters: 'criterion': 'gini', 'max_depth': 12, 'max_features': 'log2', 'min_samples_leaf': 6, 'min_samples_split': 10

After hyperparameter tuning:

- Test Accuracy: 0.9054170820870722

- Mean Squared Error: 0.09458291791292789

- the confusion matrix: 6508 true positive predictions, 365 true negative predictions, 1058 false positive predictions, and 7114 false negative predictions.

3.2.3 Random forest

We used SKLearn.ensemble Random forest lib., using grid search.

First we applied this prarms as our default:

n estimators: [50, 100, 200] Number of trees in the forest

Max depth: [None, 10, 20, 30] Maximum depth of the trees

Min samples split: [2, 5, 10] Minimum number of samples required to split a node

min samples leaf: [1, 2, 4] Minimum number of samples required at each leaf node

bootstrap: [True, False]

the results are:

- the mean is 0.011365902293120638, which indicates that the Random Forest model is performing well on the dataset.

- the accuracy score is 0.9886340977068794, which means that the Random Forest model correctly predicted the class label for 98.88% of the instances in the test set.

- The resulting confusion matrix shows the number of true positives (6788), false positives (86), false negatives (85), and true negatives (8086).

-Precision for the class labeled as "0" is 0.99. This means that 99% of the instances the model predicted as class "0" were actually class "0".

Precision for the class labeled as "1" is also 0.99. This means that 99% of the instances the model predicted as class "1" were actually class "1".

- Recall for the class "0" is 0.99. This means the model correctly identified 99% of the actual class "0" instances.

Recall for the class "1" is also 0.99. This means the model correctly identified 99% of the actual class "1" instances.

- F1-Score for both the class "0" and class "1" is 0.99, which is the harmonic mean of precision and recall.

We applied hyperparameter tuning by the params:

n_estimators: [50, 100, 200, 300]

max depth: [None, 10, 20, 30, 35, 40]

min samples split: [2, 5, 10, 15]

min samples leaf: [1, 2, 4, 6]

bootstrap: [True, False]

- Best hyperparameters: 'bootstrap': False, 'max_depth': None, 'min_samples_leaf': 1, 'min_samples_split': 5, 'n_estimators': 200

After hyperparameter tuning:

-Test Accuracy: 0.9885676304420074

- Mean Squared Error: 0.011432369557992688

- the confusion matrix: 6788 true positive predictions, 85 true negative predictions, 87 false positive predictions, and 8085 false negative predictions.

3.2.4 Bagging

We used SKLearn ensemble bagging classifier lib with maximum depth 10 and n-estimates 500.

First we applied the default parameters, the results are:

- The MSE is 0.04971751412429379, which indicates that the model is performing reasonably well.

- the accuracy score is 0.9502824858757062, which indicates that the model is performing well.

- The resulting confusion matrix shows the number of true positives (6650), false positives (223), false negatives (525), and true negatives (7647).

- Precision for the class labeled as "0" is 0.93. This means that 93% of the instances the model predicted as class "0" were actually class "0".

Precision for the class labeled as "1" is 0.97. This means that 97% of the instances the model predicted as class "1" were actually class "1".

- Recall for the class "0", the recall is 0.97. This means the model correctly identified 97% of the actual class "0" instances.

Recall for the class "1" is 0.94. This means the model correctly identified 94% of the actual class "1" instances.

- F1-Score for both the class "0" and class "1" is 0.95, which is the harmonic mean of precision and recall.

We applied hyperparameter tuning by the params:

Base estimator max depth: [5, 10, 15,20,25,30]

N_estimators: [100, 200, 350,400, 500,600]

bootstrap: [True, False]

Best hyperparameters: 'base_estimator__max_depth': 20, 'bootstrap': True, 'n_estimators': 200

After hyperparameter tuning:

- Test Accuracy: 0.9851777999335327

- Mean Squared Error: 0.014822200066467265

- The confusion matrix: 6755 true positive predictions, 118 true negative predictions, 105 false positive predictions, and 8067 false negative predictions.

3.2.5 Boosting

We used SKLearn ensemble bagging classifier lib with maximum depth 10 and n-estimates 500.

First we applied the default parameters, the results are:

- the MSE is 0.012495845795945497, which is a relatively low value, indicating that the Boosting model has performed well in predicting the target variable for the test set.

- the accuracy score is 0.9875041542040545, which is a relatively high value, indicating that the Boosting model has performed well in predicting the target variable for the test set.

- The confusion matrix shows that the Boosting model correctly predicted 8080 instances where the true label is 1, but it incorrectly predicted 92 instances where the true label is 1 as 0. Similarly, the model correctly predicted 6783 instances where the true label is 0, but it incorrectly predicted 90 instances where the true label is 0 as 1.

- Precision for the class labeled as "0" is 0.99. This means that 99% of the instances the model predicted as class "0" were actually class "0".

Precision for the class labeled as "1" is also 0.99. This means that 99% of the instances the model predicted as class "1" were actually class "1".

- Recall for the class "0" is 0.99. This means the model correctly identified 99% of the actual class "0" instances.

Recall for the class "1" is also 0.99. This means the model correctly identified 99% of the actual class "1" instances.

- F1-Score for both the class "0" and class "1" is 0.99, which is the harmonic mean of precision and recall.

We applied hyperparameter tuning by the params:

'n estimators': [100, 200, 350,400, 500,600]

'base_estimator__max_depth': [5, 10, 15,20,25,30]

'learning_rate': [0.1, 0.5, 1.0, 1.5, 2.0,2.5]

Best hyperparameters: 'base_estimator__max_depth': 20, 'learning_rate': 1.0, 'n_estimators': 400

After hyperparameter tuning:

- Test Accuracy: 0.9871053506148222

- Mean Squared Error: 0.0128946493851778

- The confusion matrix: 6779 true positive predictions, 94 true negative predictions, 100 false positive predictions, and 8072 false negative predictions.

4.Results and Technical Discussion

4.1 Results

	Logistic regression	Decision tree	Random forest	Bagging	Boosting
MSE	0.377999335 3273513	0.35247590 56164839	0.01136590 2293120638	0.049717514 12429379	0.01249584 5795945497
Accuracy	0.6220006646 726487	0.64752409 43835161	0.98863409 77068794	0.950282485 8757062	0.98750415 42040545
Confusion matrix	3678 3195 2492 5680	5145 1728 3575 4597	6788 85 86 8086	6650 223 525 7647	6778 95 93 8079
F1-Score	"0" is 0.56 "1" is 0.67	"0" is 0.66 "1" is 0.63	Both is 0.99	Both is 0.95	Both is 0.99

- Results after hyperparameter tuning

	Logistic regression	Decision tree	Random forest	Bagging	Boosting
MSE	0.3506148222 0006646	0.09458291 791292789	0.01143236 9557992688	0.014822200 066467265	0.01289464 93851778
Accuracy	0.6493851777 999335	0.90541708 20870722	0.98856763 04420074	0.985177799 9335327	0.98710535 06148222
Confusion matrix	3842 3031 2244 5928	6508 365 1058 7114	6788 85 87 8085	6755 118 105 8067	6779 94 100 8072

4.2 Result analysis

Logistic Regression:

- The logistic regression model has the highest Mean Squared Error (MSE) of 0.377999335273513 and the lowest accuracy of is 0.6220006646726487 compared to the other models.

- The F1-score for the "0" class is 0.56, and for the "1" class is 0.67, indicating that the model is not performing equally well for both classes.

Decision Tree:

- The decision tree model has an MSE of 0.35247590564839 and an accuracy of 0.64752409435161, which is lower than the logistic regression model but higher than the other models.

- The F1-score for the "0" class is 0.66, and for the "1" class is 0.63, indicating that the model is not performing equally well for both classes.

Random Forest:

- It has the lowest MSE of 0.011365902229931206 38 among all the models.

- The accuracy is 0.9886340977706879 4, which is the highest among the models.

- The F1-score for both the "0" and "1" classes is 0.99, indicating that the model is performing equally well for both classes.

Bagging:

- The bagging model has an MSE of 0.049717514429379 and an accuracy of 0.950282485757062, which are better than the logistic regression and decision tree models but worse than the random forest and boosting models.

- The F1-score for both the "0" and "1" classes is 0.95, indicating that the model is performing well for both classes.

Boosting:

- The boosting model has an MSE of 0.01249584542040545 and an accuracy of 0.98750415057955, which are very close to the random forest model.

- The F1-score for both the "0" and "1" classes is 0.99, indicating that the model is performing equally well for both classes.

After the hyperparameter tuning:

Logistic Regression:

- The logistic regression model has an MSE of 0.3506148220006646 after hyperparameter tuning, which is lower than the original MSE but still the highest among the models.

- The accuracy of the logistic regression model is 0.649385177999335, which is the lowest among the models after tuning.

- The improvement in MSE and accuracy after tuning indicates that the hyperparameter optimization helped, but the logistic regression model is still not the best performer.

Decision Tree:

- The decision tree model has an MSE of 0.09458291820870722 after hyperparameter tuning, which is a significant improvement from the original MSE.

- The accuracy of the decision tree model is 0.9054170820870722, which is also an impressive improvement from the original accuracy.

- The hyperparameter tuning has helped the decision tree model become a much stronger performer compared to the original results.

Random Forest:

- The random forest model has an MSE of 0.011432365044200 74 and an accuracy of 0.9885676349557992 6 after hyperparameter tuning.

- These results are very similar to the original random forest performance, indicating that the model was already performing well, and the tuning did not significantly improve it further.

Bagging:

- The bagging model has an MSE of 0.014822200066467265 and an accuracy of 0.9851777999335327 after hyperparameter tuning.

- These results are also very close to the original bagging performance, suggesting that the tuning did not drastically improve the model.

Boosting:

- The boosting model has an MSE of 0.012894649353272 65 and an accuracy of 0.9871053506148222 after hyperparameter tuning.

- These results are again very similar to the original boosting performance, indicating that the model was already performing well and the tuning did not significantly enhance its performance.

Conclusion of the Result Analysis:

Logistic Regression:

-The logistic regression model has the weakest performance among the models, with the highest MSE and lowest accuracy.

-The model also struggles to perform equally well on both classes, as indicated by the difference in F1-scores.

Decision Tree:

-The decision tree model shows improved performance compared to logistic regression, but still lags behind the ensemble models.

- the decision tree is better than logistic regression, it may not be the optimal choice for this classification task.

Random Forest, Bagging, and Boosting:

- These ensemble models are the top performers, with very low MSE, high accuracy, and equal performance across both classes.

- The random forest and boosting models slightly outperform the bagging model, but all three are excellent choices for this dataset.

4.3 Error analysis

Logistic Regression:

- The logistic regression model has the highest error, as indicated by the high MSE and low accuracy.

- This suggests that the logistic regression model may not be the best choice for this dataset, and other models might be more appropriate.

Decision Tree:

- The decision tree model has a higher error compared to the random forest, bagging, and boosting models.

- The model is also not performing equally well for both classes, as shown by the difference in F1-scores.

- This indicates that the decision tree model may not be capturing the underlying patterns in the dataset effectively.

Random Forest, Bagging, and Boosting:

- These ensemble models have the lowest errors, as indicated by the low MSE and high accuracy.

- The models are performing equally well for both classes, as shown by the equal F1-scores.

- These models appear to be the best choices for this dataset, with the random forest and boosting models slightly outperforming the bagging model.

After the hyperparameter tuning:

Logistic Regression:

-The logistic regression model still has the highest error among the models, as indicated by the high MSE and low accuracy, even after hyperparameter tuning.

Decision Tree:

- The decision tree model has shown a significant improvement in performance after hyperparameter tuning, with a lower MSE and higher accuracy.

- The tuning has helped the decision tree model become a much stronger performer, narrowing the gap with the ensemble models.

Random Forest, Bagging, and Boosting:

- These ensemble models continue to be the strongest performers, with very low MSE and high accuracy, even after hyperparameter tuning.

- The tuning did not significantly improve their performance, as they were already performing exceptionally well on the dataset.

Conclusion of the Error Analysis:

Logistic Regression:

- The logistic regression model has the highest error, as indicated by its high MSE and low accuracy.

- The model is also not performing equally well for both classes, suggesting it may not be capturing the underlying patterns in the dataset effectively.

- Logistic regression is not the best model for this classification task.

Decision Tree:

- The decision tree model has a higher error compared to the ensemble models, though the error has been significantly reduced after hyperparameter tuning.

- The model still exhibits unequal performance across the two classes, indicating that it may not be the optimal choice.

Random Forest, Bagging, and Boosting:

- These ensemble models have the lowest errors, with very low MSE and high accuracy.

- They also perform equally well on both classes, as shown by the equal F1-scores.

- These ensemble models are the best choices for this classification task, with the random forest and boosting models being the top performers.

5.Conclusion

Through the experimental simulation and evaluation of various classification algorithms, we have observed the performance of each model in predicting whether a mushroom is edible or poisonous based on its features.

The ensemble models, particularly the random forest and boosting, are the recommended choices for this mushroom dataset classification problem, as they consistently outperform the logistic regression and decision tree models in terms of both performance and error analysis.

The results demonstrate the importance of hyperparameter tuning in improving the performance of the models. The decision tree model, in particular, showed significant improvement in accuracy after hyperparameter tuning. The results also highlight the potential impact of data preprocessing on model performance, as the removal of outliers and duplicates improved the accuracy of all models.

5.1 Future work

We could explore the use of other classification algorithms, such as Support Vector Machines (SVM) or k-Nearest Neighbors (k-NN), to determine if they perform better than the current models. We could also consider using dimensionality reduction techniques, such as Principal Component Analysis (PCA), to reduce the number of features and improve the computational efficiency of the models.

Finally, we could deploy the best-performing model in a real-world application, such as a mobile app or a web service, to help users determine whether a mushroom is safe to eat or not. This would

require further testing and validation to ensure the model's accuracy and reliability in different environments and scenarios.

6. References

The following references provide additional information on the machine learning algorithms and techniques used in this project:

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., ... & Vanderplas, J. (2011). Scikit-learn: machine learning in Python. Journal of machine learning research, 12(Oct), 2825-2830.

Breiman, L. (2001). Random forests. Machine learning, 45(1), 5-32.

Chen, T., & Guestrin, C. (2016, August). Xgboost: A scalable tree boosting system. In Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining (pp. 785-794).

7. Appendix

Source code:

Mushroom Dataset Classification

In [3]:
```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
from sklearn import tree
data = pd.read_csv('mushroom.csv',header='infer')
df = pd.DataFrame(data)
data.columns=['cap-diameter','cap-shape','gill-attachment','gill-color','stem-height','stem-width','stem-color','season','poison
ous']
data
```

Out[3]:

	cap-diameter	cap-shape	gill-attachment	gill-color	stem-height	stem-width	stem-color	season	poisonous
0	1372	2	2	10	3.807467	1545	11	1.804273	1
1	1461	2	2	10	3.807467	1557	11	1.804273	1
2	1371	2	2	10	3.612496	1566	11	1.804273	1
3	1261	6	2	10	3.787572	1566	11	1.804273	1
4	1305	6	2	10	3.711971	1464	11	0.943195	1
5	1337	6	2	10	3.775635	1520	11	0.943195	1
6	1300	2	2	10	3.835320	1563	11	1.804273	1
7	1354	6	2	10	3.676160	1532	11	0.888450	1
8	1222	6	2	10	3.771656	1476	11	0.943195	1
9	1085	6	2	10	3.775635	1581	11	0.888450	1
10	1214	6	2	10	3.696055	1524	11	1.804273	1
11	642	6	2	10	0.286062	1311	11	0.943195	1
12	814	4	2	10	1.189292	1681	11	0.943195	1
13	550	4	2	10	0.548675	1220	11	0.888450	1
14	606	6	2	10	0.254230	1239	11	0.943195	1
15	721	6	2	10	0.950553	1445	11	0.943195	1
16	595	6	2	10	0.604381	1415	11	0.888450	1
17	661	4	2	10	0.437263	1150	11	0.888450	1
18	587	6	2	10	0.596423	1531	11	0.943195	1
19	835	6	2	10	0.906784	1603	11	0.888450	1
20	705	4	2	10	0.083133	1122	11	0.888450	1
21	599	4	2	10	0.576528	1306	11	0.888450	1
22	619	4	2	10	0.389515	1195	11	0.943195	1
23	648	6	2	10	0.727729	1370	11	0.888450	1
24	778	6	2	10	0.811288	1514	11	0.943195	1
25	692	6	2	10	0.652128	1307	11	0.943195	1
26	804	4	2	10	0.870973	1517	11	0.888450	1
27	660	6	2	10	0.982385	1626	11	0.943195	1
28	547	6	2	10	0.520822	1191	11	0.888450	1
29	846	6	2	10	1.173376	1613	11	0.888450	1
...	...	...	...	...	...	...	...	...	...
54005	75	5	3	2	1.130459	451	12	0.888450	1
54006	71	6	3	2	1.428883	489	12	0.943195	1
54007	81	6	3	2	0.871824	547	12	0.888450	1
54008	71	2	3	2	0.987215	544	12	0.943195	1
54009	76	6	3	2	1.102606	514	12	0.888450	1
54010	89	6	3	2	0.855908	576	12	0.943195	1
54011	71	2	3	2	1.114543	441	12	0.943195	1
54012	75	5	3	2	1.015068	466	12	0.888450	1
54013	81	2	3	2	1.086690	475	12	0.888450	1
54014	74	6	3	2	1.162290	563	12	0.943195	1
54015	71	6	3	2	0.991194	445	12	0.943195	1
54016	70	2	3	2	1.241870	395	12	0.888450	1
54017	72	6	3	2	1.206059	490	12	0.888450	1
54018	85	2	3	2	1.122501	498	12	0.888450	1
54019	86	6	3	2	0.859887	541	12	0.943195	1
54020	75	2	3	2	1.512442	397	12	0.888450	1
54021	82	2	3	2	1.353282	462	12	0.888450	1
54022	84	6	3	2	0.828055	575	12	0.943195	1
54023	99	5	3	2	0.967320	534	12	0.943195	1
54024	74	2	3	2	1.042921	517	12	0.943195	1
54025	70	2	3	2	1.325429	454	12	0.888450	1
54026	68	6	3	2	1.233912	429	12	0.888450	1
54027	80	6	3	2	0.700728	608	12	0.943195	1
54028	80	5	3	2	1.054858	412	12	0.888450	1
54029	97	6	3	2	0.509736	527	12	0.943195	1
54030	73	5	3	2	0.887740	569	12	0.943195	1
54031	82	2	3	2	1.186164	490	12	0.943195	1
54032	82	5	3	2	0.915593	584	12	0.888450	1
54033	79	2	3	2	1.034963	491	12	0.888450	1
54034	72	5	3	2	1.158311	492	12	0.888450	1

54035 rows × 9 columns

Mushroom Dataset Classification

Data Exploration

1.Summary Statistics

In [4]:
```
data.isnull().sum()
```

Out[4]:
```
cap-diameter       0
cap-shape          0
gill-attachment    0
gill-color         0
stem-height        0
stem-width         0
stem-color         0
season             0
poisonous          0
dtype: int64
```

In [5]:
```
data.dtypes
```

Out[5]:
```
cap-diameter         int64
cap-shape            int64
gill-attachment      int64
gill-color           int64
stem-height        float64
stem-width           int64
stem-color           int64
season             float64
poisonous            int64
dtype: object
```

In [6]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 54035 entries, 0 to 54034
Data columns (total 9 columns):
cap-diameter       54035 non-null int64
cap-shape          54035 non-null int64
gill-attachment    54035 non-null int64
gill-color         54035 non-null int64
stem-height        54035 non-null float64
stem-width         54035 non-null int64
stem-color         54035 non-null int64
season             54035 non-null float64
poisonous          54035 non-null int64
dtypes: float64(2), int64(7)
memory usage: 3.7 MB
```

In [7]:
```
from pandas.api.types import is_numeric_dtype

for col in data.columns:
    if is_numeric_dtype(data[col]):
        print('%s:' % (col))
        print('\t Mean = %.2f' % data[col].mean())
        print('\t Standard deviation = %.2f' % data[col].std())
        print('\t Minimum = %.2f' % data[col].min())
        print('\t Maximum = %.2f' % data[col].max())
```

```
cap-diameter:
         Mean = 567.26
         Standard deviation = 359.88
         Minimum = 0.00
         Maximum = 1891.00
cap-shape:
         Mean = 4.00
         Standard deviation = 2.16
         Minimum = 0.00
         Maximum = 6.00
gill-attachment:
         Mean = 2.14
         Standard deviation = 2.23
         Minimum = 0.00
         Maximum = 6.00
gill-color:
         Mean = 7.33
         Standard deviation = 3.20
         Minimum = 0.00
         Maximum = 11.00
stem-height:
         Mean = 0.76
         Standard deviation = 0.65
         Minimum = 0.00
         Maximum = 3.84
stem-width:
         Mean = 1051.08
         Standard deviation = 782.06
         Minimum = 0.00
         Maximum = 3569.00
stem-color:
         Mean = 8.42
         Standard deviation = 3.26
         Minimum = 0.00
         Maximum = 12.00
season:
         Mean = 0.95
         Standard deviation = 0.31
         Minimum = 0.03
         Maximum = 1.80
poisonous:
         Mean = 0.55
         Standard deviation = 0.50
         Minimum = 0.00
         Maximum = 1.00
```

In [8]:
```
data['poisonous'].value_counts()
```

Out[8]:
```
1    29675
0    24360
Name: poisonous, dtype: int64
```

In [9]:
```
data.describe(include='all')
```

Out[9]:

	cap-diameter	cap-shape	gill-attachment	gill-color	stem-height	stem-width	stem-color	season	poisonous
count	54035.000000	54035.000000	54035.000000	54035.000000	54035.000000	54035.000000	54035.000000	54035.000000	54035.000000
mean	567.257204	4.000315	2.142056	7.329509	0.759110	1051.081299	8.418062	0.952163	0.549181
std	359.883763	2.160505	2.228821	3.200266	0.650969	782.056076	3.262078	0.305594	0.497580
min	0.000000	0.000000	0.000000	0.000000	0.000426	0.000000	0.000000	0.027372	0.000000
25%	289.000000	2.000000	0.000000	5.000000	0.270997	421.000000	6.000000	0.888450	0.000000
50%	525.000000	5.000000	1.000000	8.000000	0.593295	923.000000	11.000000	0.943195	1.000000
75%	781.000000	6.000000	4.000000	10.000000	1.054858	1523.000000	11.000000	0.943195	1.000000
max	1891.000000	6.000000	6.000000	11.000000	3.835320	3569.000000	12.000000	1.804273	1.000000

In [10]:
```
print('Covariance:')
data.cov()
```

```
Covariance:
```

Out[10]:

	cap-diameter	cap-shape	gill-attachment	gill-color	stem-height	stem-width	stem-color	season	poisonous
cap-diameter	129516.323081	158.625044	160.809485	214.654986	31.779700	233172.092943	143.055276	12.464316	-29.667811
cap-shape	158.625044	4.667783	0.207380	0.908435	-0.014617	375.933535	0.204628	0.036605	-0.143342
gill-attachment	160.809485	0.207380	4.967642	0.715248	-0.109229	427.573157	0.145944	-0.027459	-0.058269
gill-color	214.654986	0.908435	0.715248	10.241701	0.031367	276.014814	1.942692	0.058644	-0.101828
stem-height	31.779700	-0.014617	-0.109229	0.031367	0.423760	49.939605	0.005572	-0.000058	0.059390
stem-width	233172.092943	375.933535	427.573157	276.014814	49.939605	611611.705997	401.532877	9.722024	-71.155820
stem-color	143.055276	0.204628	0.145944	1.942692	0.005572	401.532877	10.641151	0.010716	-0.208313
season	12.464316	0.036605	-0.027459	0.058644	-0.000058	9.722024	0.010716	0.093387	-0.012608
poisonous	-29.667811	-0.143342	-0.058269	-0.101828	0.059390	-71.155820	-0.208313	-0.012608	0.247586

In [11]:
```
print('Correlation:')
data.corr()
```

```
Correlation:
```

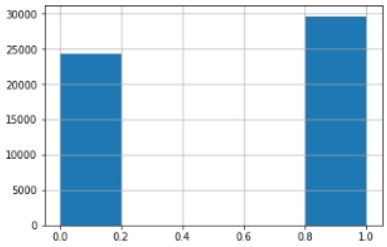
Out[11]:

	cap-diameter	cap-shape	gill-attachment	gill-color	stem-height	stem-width	stem-color	season	poisonous
cap-diameter	1.000000	0.204011	0.200481	0.186377	0.135652	0.828469	0.121856	0.113334	-0.165676
cap-shape	0.204011	1.000000	0.043066	0.131387	-0.010393	0.222494	0.029035	0.055442	-0.133338
gill-attachment	0.200481	0.043066	1.000000	0.100276	-0.075284	0.245300	0.020073	-0.040315	-0.052541
gill-color	0.186377	0.131387	0.100276	1.000000	0.015057	0.110283	0.186090	0.059965	-0.063947
stem-height	0.135652	-0.010393	-0.075284	0.015057	1.000000	0.098095	0.002624	-0.000292	0.183354
stem-width	0.828469	0.222494	0.245300	0.110283	0.098095	1.000000	0.157394	0.040679	-0.182856
stem-color	0.121856	0.029035	0.020073	0.186090	0.002624	0.157394	1.000000	0.010750	-0.128339
season	0.113334	0.055442	-0.040315	0.059965	-0.000292	0.040679	0.010750	1.000000	-0.082919
poisonous	-0.165676	-0.133338	-0.052541	-0.063947	0.183354	-0.182856	-0.128339	-0.082919	1.000000

2.Data Visualization

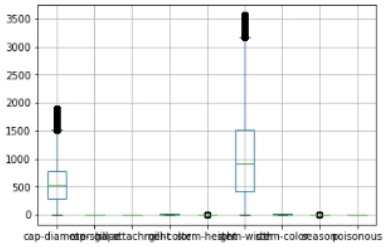
In [12]:
```
%matplotlib inline

data['poisonous'].hist(bins=5)
```

Out[12]: <matplotlib.axes._subplots.AxesSubplot at 0x2e1089c1128>

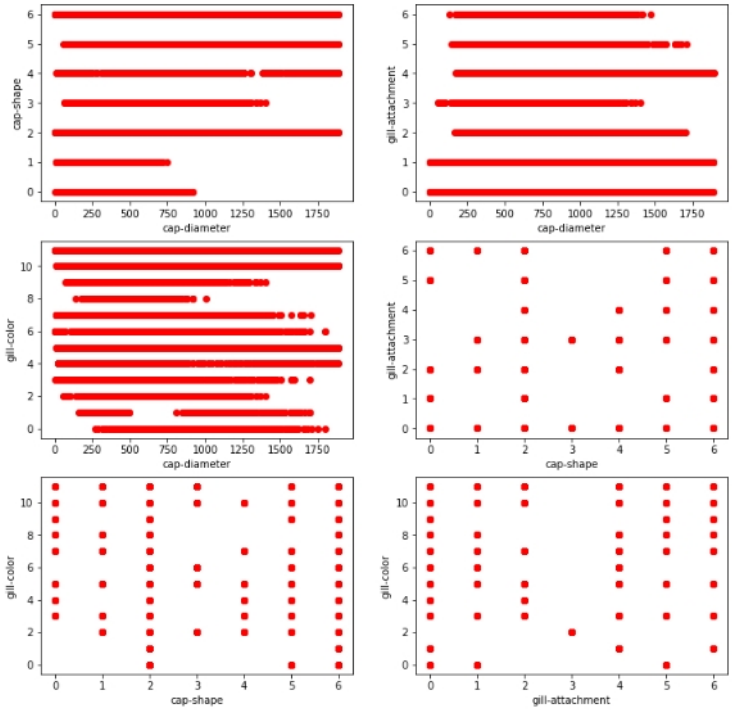


In [13]:
```
data.boxplot()
```

Out[13]: <matplotlib.axes._subplots.AxesSubplot at 0x2e1409d53c8>



In [14]:
```
import matplotlib.pyplot as plt

fig, axes = plt.subplots(3, 2, figsize=(12,12))
index = 0
for i in range(3):
    for j in range(i+1,4):
        ax1 = int(index/2)
        ax2 = index % 2
        axes[ax1][ax2].scatter(data[data.columns[i]], data[data.columns[j]], color='red')
        axes[ax1][ax2].set_xlabel(data.columns[i])
        axes[ax1][ax2].set_ylabel(data.columns[j])
        index = index + 1
```



data preprocessing

1.Missing values

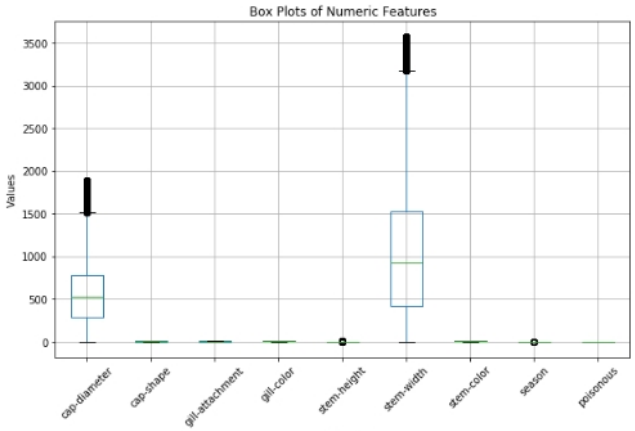
In [15]:
```
data = data.replace('?',np.NaN)

print('Number of instances = %d' % (data.shape[0]))
print('Number of attributes = %d' % (data.shape[1]))

print('Number of missing values:')
for col in data.columns:
    print('\t%s: %d' % (col,data[col].isna().sum()))
```

```
Number of instances = 54035
Number of attributes = 9
Number of missing values:
        cap-diameter: 0
        cap-shape: 0
        gill-attachment: 0
        gill-color: 0
        stem-height: 0
        stem-width: 0
        stem-color: 0
        season: 0
        poisonous: 0
```

there is no missing values

2.Outliers

In [16]:
```
from scipy import stats
import matplotlib.pyplot as plt
import seaborn as sns
# Select the numerical columns (assuming they are already preprocessed as numeric)
numeric_columns = ['cap-diameter','cap-shape','gill-attachment','gill-color','stem-height','stem-width','stem-color','season','p
oisonous']  # Replace with the actual column names

plt.figure(figsize=(10, 6))
data[numeric_columns].boxplot()
plt.title('Box Plots of Numeric Features')
plt.xlabel('Features')
plt.ylabel('Values')
plt.xticks(rotation=45)
plt.show()

# Plot scatter plots to visualize relationships between pairs of numeric features
sns.pairplot(data=data, vars=numeric_columns)
plt.title('Pairwise Scatter Plots of Numeric Features')
plt.show()
```



In [17]:
```
# Calculate z-scores for each numeric column
z_scores = np.abs(stats.zscore(data[numeric_columns]))

# Define a threshold for outlier detection
threshold = 3

# Identify outliers based on the z-scores exceeding the threshold
outlier_indices = np.where(z_scores > threshold)

# Remove outliers from the dataset
mushroom_data_cleaned = data.drop(data.index[outlier_indices[0]])

# Print the number of removed outliers
print("Number of outliers removed:", len(outlier_indices[0]))
```

```
Number of outliers removed: 3827
```

In [18]:
```
#put the new cleaned data in "data"
data = mushroom_data_cleaned
```

3.Duplicates

In [19]:
```
dups = data.duplicated()
print('Number of duplicate rows = %d' % (dups.sum()))
```

```
Number of duplicate rows = 250
```

In [20]:
```
print('Number of rows before discarding duplicates = %d' % (data.shape[0]))
data2 = data.drop_duplicates()
print('Number of rows after discarding duplicates = %d' % (data2.shape[0]))
```

```
Number of rows before discarding duplicates = 50397
Number of rows after discarding duplicates = 50147
```

In [21]:
```
#put the new cleaned data in "data"
data = data2
```

we think that this data set doesnt need Aggregation , Discretization or Principal Component Analysis

Classification

data spliting

In [22]:
```
from sklearn.model_selection import train_test_split

X = data[['cap-diameter','cap-shape','gill-attachment','gill-color','stem-height','stem-width','stem-color','season']]
Y = data['poisonous']


# Split the dataset into training and test data
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.3, random_state=42)
```

In [23]:
```
#applying Pandas cross-tabulation to examine the relationship between the attributes with respect to "poisonous" class.
pd.crosstab([data['cap-diameter'],data['cap-shape'], data['gill-attachment'],data['gill-color'],data['stem-height'],data['stem-width'],data['stem-color'],data['season']],data['poisonous'])
```

Out[23]:

							poisonous	0	1
cap-diameter	cap-shape	gill-attachment	gill-color	stem-height	stem-width	stem-color	season		
0	6	0	11	1.019047	21	12	0.943195	0	1
1	2	1	6	0.895698	42	7	0.943195	0	1
	6	0	11	1.198101	1	11	0.888450	0	1
2	0	0	3	1.452757	174	6	0.943195	0	1
	2	1	6	1.058837	31	7	0.943195	0	1
	6	1	6	0.752455	40	12	0.943195	0	1
3	6	0	11	1.098627	22	11	0.888450	0	1
				1.142396	10	12	0.888450	0	1
4	6	0	11	1.023026	23	12	0.943195	0	1
		1	6	0.983236	34	7	0.888450	0	1
5	2	1	6	0.776329	37	12	0.943195	0	1
				0.784287	35	7	0.888450	0	1
				0.923551	32	7	0.943195	0	1
	6	0	11	1.090669	22	11	0.943195	0	1
				1.098627	14	12	0.943195	0	1
				1.245849	4	11	0.943195	0	1
		1	6	0.545547	55	7	0.943195	0	1
6	2	1	6	0.577379	47	7	0.943195	0	1
				0.824076	43	12	0.943195	0	1
				0.887740	42	7	0.943195	0	1

1.Decision Tree Classifier

In [24]:
```
from sklearn.tree import DecisionTreeRegressor
from sklearn.metrics import mean_squared_error

clf = tree.DecisionTreeClassifier(criterion='entropy',max_depth=3)

# Fit the model on the training data
clf.fit(X_train, y_train)
```

Out[24]:
```
DecisionTreeClassifier(criterion='entropy', max_depth=3)
```

In [25]:
```
# Make predictions
y_predictions = clf.predict(X_test)
```

In [26]:
```
mse = mean_squared_error(y_test, y_predictions)
print('Mean Squared Error:',mse)
```

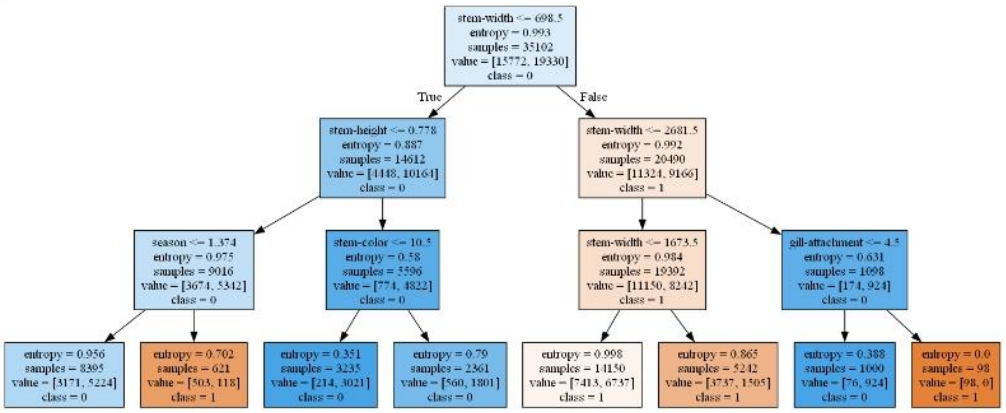
```
Mean Squared Error: 0.3524759056164839
```

In [27]:
```
clf.get_params()
```

Out[27]:
```
{'ccp_alpha': 0.0,
 'class_weight': None,
 'criterion': 'entropy',
 'max_depth': 3,
 'max_features': None,
 'max_leaf_nodes': None,
 'min_impurity_decrease': 0.0,
 'min_impurity_split': None,
 'min_samples_leaf': 1,
 'min_samples_split': 2,
 'min_weight_fraction_leaf': 0.0,
 'random_state': None,
 'splitter': 'best'}
```

In [28]:
```
import pydotplus
from IPython.display import Image

dot_data = tree.export_graphviz(clf, feature_names=X.columns, class_names=['1','0'], filled=True,
                                out_file=None)
graph = pydotplus.graph_from_dot_data(dot_data)
Image(graph.create_png())
```

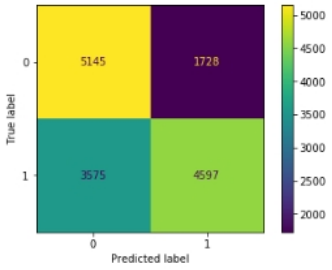
Out[28]:



In [29]:
```
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix

# Evaluate the model
accuracy = accuracy_score(y_test, y_predictions)
print("Accuracy:", accuracy)

# Print classification report
print("Classification Report:")
print(classification_report(y_test, y_predictions))
```

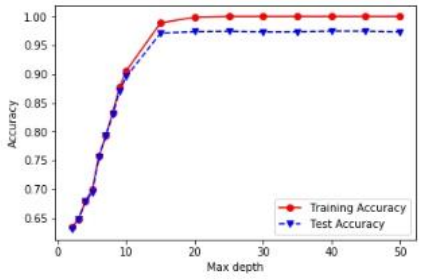
```
Accuracy: 0.6475240943835161
Classification Report:
              precision    recall  f1-score   support

           0       0.59      0.75      0.66      6873
           1       0.73      0.56      0.63      8172

    accuracy                           0.65     15045
   macro avg       0.66      0.66      0.65     15045
weighted avg       0.66      0.65      0.65     15045
```

In [30]:
```
import matplotlib.pyplot as plt
import numpy
from sklearn import metrics

confusion_matrix = metrics.confusion_matrix(y_test,y_predictions )

cm_display = metrics.ConfusionMatrixDisplay(confusion_matrix = confusion_matrix, display_labels = [0, 1])

cm_display.plot()
plt.show()
```



In [31]:
```
#model overfitting.

maxdepths = [2,3,4,5,6,7,8,9,10,15,20,25,30,35,40,45,50]

trainAcc = np.zeros(len(maxdepths))
testAcc = np.zeros(len(maxdepths))

index = 0
for depth in maxdepths:
    clf = tree.DecisionTreeClassifier(max_depth=depth)
    clf = clf.fit(X_train, y_train)
    Y_predTrain = clf.predict(X_train)
    Y_predTest = clf.predict(X_test)
    trainAcc[index] = accuracy_score(y_train, Y_predTrain)
    testAcc[index] = accuracy_score(y_test, Y_predTest)
    index += 1

########################################
# Plot of training and test accuracies
########################################

plt.plot(maxdepths,trainAcc,'ro-',maxdepths,testAcc,'bv--')
plt.legend(['Training Accuracy','Test Accuracy'])
plt.xlabel('Max depth')
plt.ylabel('Accuracy')
```

Out[31]:
```
Text(0,0.5,'Accuracy')
```



Hyperparameter Tuning

In [32]:
```
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import GridSearchCV, RandomizedSearchCV
from sklearn.metrics import accuracy_score

param_grid = {
    'criterion': ['gini', 'entropy'],
    'max_depth': [2, 3, 4, 5, 6, 7, 8, 9, 10],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
    'max_features': ['auto', 'sqrt', 'log2']
}

# Create the Decision Tree Classifier model
clf = DecisionTreeClassifier()

# Create the GridSearchCV object
grid_search = GridSearchCV(estimator=clf, param_grid=param_grid, cv=5, scoring='accuracy')

# Fit the GridSearchCV object to the training data
grid_search.fit(X_train, y_train)

# Get the best hyperparameters
best_params = grid_search.best_params_
print("Best hyperparameters:", best_params)

# Evaluate the model with the best hyperparameters
best_clf = grid_search.best_estimator_
y_pred = best_clf.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print("Test Accuracy:", accuracy)

mse = mean_squared_error(y_test, y_pred)
print('Mean Squared Error:',mse)
```

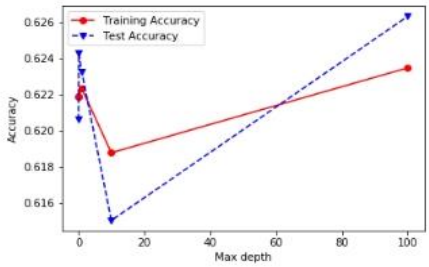
```
Best hyperparameters: {'criterion': 'entropy', 'max_depth': 10, 'max_features': 'log2', 'min_samples_leaf': 1, 'min_samples_spl
it': 10}
Test Accuracy: 0.8466600199401795
Mean Squared Error: 0.1533399800598205
```

```
Out[30]: LogisticRegression()

In [31]: y_pred = log.predict(X_test)

In [32]: mse = mean_squared_error(y_test, y_pred)
         print('Mean Squared Error:',mse)

         Mean Squared Error: 0.37673645729478233

In [33]: clf.get_params()

Out[33]: {'ccp_alpha': 0.0,
          'class_weight': None,
          'criterion': 'gini',
          'max_depth': 50,
          'max_features': None,
          'max_leaf_nodes': None,
          'min_impurity_decrease': 0.0,
          'min_impurity_split': None,
          'min_samples_leaf': 1,
          'min_samples_split': 2,
          'min_weight_fraction_leaf': 0.0,
          'random_state': None,
          'splitter': 'best'}
```

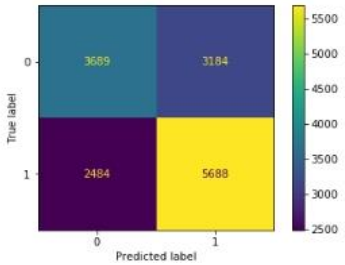
In [78]:

```python
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score


# Define a range of complexity (e.g., different values of C)
complexity_values = [0.01, 0.1, 1, 10, 100]

# Initialize lists to store training and validation accuracies
train_accuracies = []
val_accuracies = []

# Train and evaluate models for different levels of complexity
for complexity in complexity_values:
    # Create and train the logistic regression model
    clf = LogisticRegression(C=complexity)
    clf.fit(X_train, y_train)

    # Predict labels for training and validation sets
    train_predictions = clf.predict(X_train)
    val_predictions = clf.predict(X_test)

    # Calculate accuracies
    train_accuracy = accuracy_score(y_train, train_predictions)
    val_accuracy = accuracy_score(y_test, val_predictions)

    # Append accuracies to the lists
    train_accuracies.append(train_accuracy)
    val_accuracies.append(val_accuracy)

# Plot the graph
plt.plot(complexity_values,train_accuracies,'ro-',complexity_values,val_accuracies,'bv--')
plt.legend(['Training Accuracy','Test Accuracy'])
plt.xlabel('Max depth')
plt.ylabel('Accuracy')
```

Out[78]: Text(0,0.5,'Accuracy')



In [35]:
```
import matplotlib.pyplot as plt
import numpy
from sklearn import metrics

confusion_matrix = metrics.confusion_matrix(y_test,y_pred )

cm_display = metrics.ConfusionMatrixDisplay(confusion_matrix = confusion_matrix, display_labels = [0, 1])

cm_display.plot()
plt.show()
```



In [36]:
```
print(classification_report(y_test, y_pred))
accuracy_score(y_test, y_pred)
```

```
              precision    recall  f1-score   support

           0       0.60      0.54      0.57      6873
           1       0.64      0.70      0.67      8172

    accuracy                           0.62     15045
   macro avg       0.62      0.62      0.62     15045
weighted avg       0.62      0.62      0.62     15045
```

Out[36]: 0.6232635427052177

Hyperparameter Tuning

In [119]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import GridSearchCV, RandomizedSearchCV
from sklearn.metrics import accuracy_score

param_grid = {
    'C': [0.1, 1, 10],
    'penalty': ['l1', 'l2'],
    'solver': ['liblinear', 'newton-cg', 'sag'],
    'max_iter': [100, 200, 300]
}


log = LogisticRegression()
# Create the GridSearchCV object
grid_search = GridSearchCV(estimator=log, param_grid=param_grid, cv=5, scoring='accuracy')

# Fit the Grid object to the training data
grid_search.fit(X_train, y_train)


# Get the best hyperparameters
best_params = grid_search.best_params_
print("Best hyperparameters:", best_params)

# Evaluate the model with the best hyperparameters
best_log = grid_search.best_estimator_
y_pred = best_log.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print("Test Accuracy:", accuracy)
```

```
Best hyperparameters: {'C': 0.1, 'max_iter': 100, 'penalty': 'l2', 'solver': 'newton-cg'}
Test Accuracy: 0.6493851777999335
```

3.Random forest

In [37]:
```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV
```

In [38]:
```
# parameter grid
param_grid = {
    'n_estimators': [50, 100, 200],        # Number of trees in the forest
    'max_depth': [None, 10, 20, 30],        # Maximum depth of the trees
    'min_samples_split': [2, 5, 10],        # Minimum number of samples required to split a node
    'min_samples_leaf': [1, 2, 4],          # Minimum number of samples required at each leaf node
    'bootstrap': [True, False]              # Whether bootstrap samples are used when building trees
}
```

In [39]:
```
rf = RandomForestClassifier()

grid = GridSearchCV(estimator= rf, param_grid=param_grid, cv=5, verbose = 1, scoring = 'accuracy', n_jobs= -1)
```

In [40]:
```
grid.fit(X_train, y_train)
```

```
Fitting 5 folds for each of 216 candidates, totalling 1080 fits
```

Out[40]:
```
GridSearchCV(cv=5, estimator=RandomForestClassifier(), n_jobs=-1,
             param_grid={'bootstrap': [True, False],
                         'max_depth': [None, 10, 20, 30],
                         'min_samples_leaf': [1, 2, 4],
                         'min_samples_split': [2, 5, 10],
                         'n_estimators': [50, 100, 200]},
             scoring='accuracy', verbose=1)
```

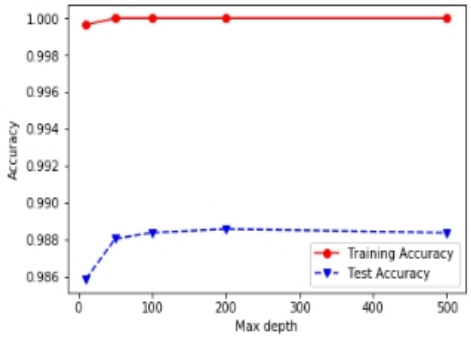
In [41]:
```
y_predic = grid.predict(X_test)
```

In [42]:
```
mse = mean_squared_error(y_test, y_predic)
print('Mean Squared Error:',mse)
```

```
Mean Squared Error: 0.011166500498504487
```

In [77]:
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score



# Define a range of complexity (e.g., different values of n_estimators or max_depth)
complexity_values = [10, 50, 100, 200, 500]

# Initialize lists to store training and validation accuracies
train_accuracies = []
val_accuracies = []

# Train and evaluate models for different levels of complexity
for complexity in complexity_values:
    # Create and train the random forest model
    model = RandomForestClassifier(n_estimators=complexity)
    model.fit(X_train, y_train)

    # Predict labels for training and validation sets
    train_predictions = model.predict(X_train)
    val_predictions = model.predict(X_test)

    # Calculate accuracies
    train_accuracy = accuracy_score(y_train, train_predictions)
    val_accuracy = accuracy_score(y_test, val_predictions)

    # Append accuracies to the lists
    train_accuracies.append(train_accuracy)
    val_accuracies.append(val_accuracy)

# Plot the graph
plt.plot(complexity_values,train_accuracies ,'ro-',complexity_values,val_accuracies,'bv--')
plt.legend(['Training Accuracy','Test Accuracy'])
plt.xlabel('Max depth')
plt.ylabel('Accuracy')
```

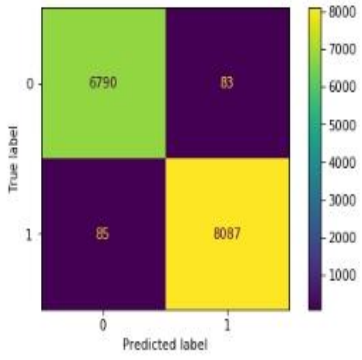
Out[77]: Text(0,0.5,'Accuracy')



In [44]:
```
grid.get_params()
```

Out[44]:
```
{'cv': 5,
 'error_score': nan,
 'estimator__bootstrap': True,
 'estimator__ccp_alpha': 0.0,
 'estimator__class_weight': None,
 'estimator__criterion': 'gini',
 'estimator__max_depth': None,
 'estimator__max_features': 'auto',
 'estimator__max_leaf_nodes': None,
 'estimator__max_samples': None,
 'estimator__min_impurity_decrease': 0.0,
 'estimator__min_impurity_split': None,
 'estimator__min_samples_leaf': 1,
 'estimator__min_samples_split': 2,
 'estimator__min_weight_fraction_leaf': 0.0,
 'estimator__n_estimators': 100,
 'estimator__n_jobs': None,
 'estimator__oob_score': False,
 'estimator__random_state': None,
 'estimator__verbose': 0,
 'estimator__warm_start': False,
 'estimator': RandomForestClassifier(),
 'n_jobs': -1,
 'param_grid': {'n_estimators': [50, 100, 200],
  'max_depth': [None, 10, 20, 30],
  'min_samples_split': [2, 5, 10],
  'min_samples_leaf': [1, 2, 4],
  'bootstrap': [True, False]},
 'pre_dispatch': '2*n_jobs',
 'refit': True,
 'return_train_score': False,
 'scoring': 'accuracy',
 'verbose': 1}
```

In [45]:
```
import matplotlib.pyplot as plt
import numpy
from sklearn import metrics

confusion_matrix = metrics.confusion_matrix(y_test,y_predic )

cm_display = metrics.ConfusionMatrixDisplay(confusion_matrix = confusion_matrix, display_labels = [0, 1])

cm_display.plot()
plt.show()
```



In [46]:
```
print(classification_report(y_test, y_predic))
accuracy_score(y_test, y_predic)
```

```
              precision    recall  f1-score   support

           0       0.99      0.99      0.99      6873
           1       0.99      0.99      0.99      8172

    accuracy                           0.99     15045
   macro avg       0.99      0.99      0.99     15045
weighted avg       0.99      0.99      0.99     15045
```

Out[46]: 0.9888334995014955

Hyperparameter Tuning

In [120]:
```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import accuracy_score

# Define the parameter grid
param_grid = {
    'n_estimators': [50, 100, 200, 300],
    'max_depth': [None, 10, 20, 30, 35, 40],
    'min_samples_split': [2, 5, 10, 15],
    'min_samples_leaf': [1, 2, 4, 6],
    'bootstrap': [True, False]
}

# Create the Random Forest Classifier and the GridSearchCV object
rf = RandomForestClassifier()
grid = GridSearchCV(estimator=rf, param_grid=param_grid, cv=5, verbose=1, scoring='accuracy', n_jobs=-1)

# Fit the GridSearchCV object
grid.fit(X_train, y_train)

# Get the best hyperparameters and evaluate the model
best_params = grid.best_params_
print("Best hyperparameters:", best_params)

best_rf = RandomForestClassifier(**best_params)
best_rf.fit(X_train, y_train)
y_pred = best_rf.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print("Test Accuracy:", accuracy)
```

```
Fitting 5 folds for each of 768 candidates, totalling 3840 fits
Best hyperparameters: {'bootstrap': False, 'max_depth': 35, 'min_samples_leaf': 1, 'min_samples_split': 5, 'n_estimators': 200}
Test Accuracy: 0.9883017613825191
```

4.Bagging

In [60]:
```
from sklearn import ensemble
from sklearn.tree import DecisionTreeClassifier

numBaseClassifiers = 500
maxdepth = 10


clf = ensemble.BaggingClassifier(DecisionTreeClassifier(max_depth=maxdepth),n_estimators=numBaseClassifiers)
clf.fit(X_train, y_train)
Y_predTrain = clf.predict(X_train)
Y_predTest = clf.predict(X_test)
```

In [61]:
```
mse = mean_squared_error(y_test, Y_predTest)
print('Mean Squared Error:',mse)
```

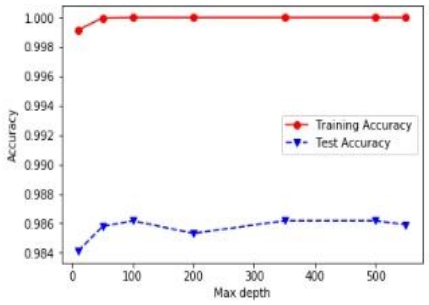
```
Mean Squared Error: 0.05004985044865404
```

In [62]:
```
clf.get_params()
```

Out[62]:
```
{'base_estimator__ccp_alpha': 0.0,
 'base_estimator__class_weight': None,
 'base_estimator__criterion': 'gini',
 'base_estimator__max_depth': 10,
 'base_estimator__max_features': None,
 'base_estimator__max_leaf_nodes': None,
 'base_estimator__min_impurity_decrease': 0.0,
 'base_estimator__min_impurity_split': None,
 'base_estimator__min_samples_leaf': 1,
 'base_estimator__min_samples_split': 2,
 'base_estimator__min_weight_fraction_leaf': 0.0,
 'base_estimator__random_state': None,
 'base_estimator__splitter': 'best',
 'base_estimator': DecisionTreeClassifier(max_depth=10),
 'bootstrap': True,
 'bootstrap_features': False,
 'max_features': 1.0,
 'max_samples': 1.0,
 'n_estimators': 500,
 'n_jobs': None,
 'oob_score': False,
 'random_state': None,
 'verbose': 0,
 'warm_start': False}
```

In [92]:
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.ensemble import BaggingClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score

# Define a range of complexity (e.g., different values of n_estimators)
complexity_values = [10, 50, 100, 200,350,500,550 ]

# Initialize lists to store training and validation accuracies
train_accuracies = []
val_accuracies = []

# Train and evaluate ensembles for different levels of complexity
for complexity in complexity_values:
    # Create and train the bagging ensemble
    base_estimator = DecisionTreeClassifier()
    model = BaggingClassifier(base_estimator=base_estimator, n_estimators=complexity)
    model.fit(X_train, y_train)

    # Predict labels for training and validation sets
    train_predictions = model.predict(X_train)
    val_predictions = model.predict(X_test)

    # Calculate accuracies
    train_accuracy = accuracy_score(y_train, train_predictions)
    val_accuracy = accuracy_score(y_test, val_predictions)

    # Append accuracies to the lists
    train_accuracies.append(train_accuracy)
    val_accuracies.append(val_accuracy)

# Plot the graph
plt.plot(complexity_values,train_accuracies,'ro-',complexity_values,val_accuracies,'bv--')
plt.legend(['Training Accuracy','Test Accuracy'])
plt.xlabel('Max depth')
plt.ylabel('Accuracy')
```

Out[92]: Text(0,0.5,'Accuracy')



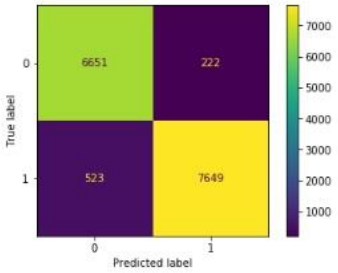
In [51]:
```
print(accuracy_score(y_test, Y_predTest))
```

```
0.9504818876703224
```

In [52]:
```
print(classification_report(y_test, Y_predTest))
```

```
              precision    recall  f1-score   support

           0       0.93      0.97      0.95      6873
           1       0.97      0.94      0.95      8172

    accuracy                           0.95     15045
   macro avg       0.95      0.95      0.95     15045
weighted avg       0.95      0.95      0.95     15045
```

In [53]:
```
confusion_matrix = metrics.confusion_matrix(y_test, Y_predTest)

cm_display = metrics.ConfusionMatrixDisplay(confusion_matrix = confusion_matrix, display_labels = [0, 1])

cm_display.plot()
plt.show()
```



Hyperparameter Tuning

In [122]:
```
from sklearn import ensemble
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import GridSearchCV

# Define the parameter grid
param_grid = {
    'base_estimator__max_depth': [5, 10, 15,20,25,30],
    'n_estimators': [100, 200, 350,400, 500,600],
    'bootstrap': [True, False]
}

# Create the Bagging Classifier and the GridSearchCV object
clf = ensemble.BaggingClassifier(DecisionTreeClassifier())
grid = GridSearchCV(estimator=clf, param_grid=param_grid, cv=5, verbose=1, scoring='accuracy', n_jobs=-1)

# Fit the GridSearchCV object
grid.fit(X_train, y_train)

# Get the best hyperparameters and evaluate the model
best_params = grid.best_params_
print("Best hyperparameters:", best_params)

best_clf = ensemble.BaggingClassifier(DecisionTreeClassifier(max_depth=best_params['base_estimator__max_depth']), n_estimators=b
est_params['n_estimators'], bootstrap=best_params['bootstrap'])
best_clf.fit(X_train, y_train)
y_pred = best_clf.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print("Test Accuracy:", accuracy)
```

```
Fitting 5 folds for each of 72 candidates, totalling 360 fits
Best hyperparameters: {'base_estimator__max_depth': 25, 'bootstrap': True, 'n_estimators': 400}
Test Accuracy: 0.9863742107012297
```

5.Boosting

In [54]:
```
numBaseClassifiers = 500
maxdepth = 10


clf = ensemble.AdaBoostClassifier(DecisionTreeClassifier(max_depth=maxdepth),n_estimators=numBaseClassifiers)
clf.fit(X_train, y_train)
Y_predTrain = clf.predict(X_train)
Y_predTestboos = clf.predict(X_test)
```

In [55]:
```
clf.get_params()
```

Out[55]:
```
{'algorithm': 'SAMME.R',
 'base_estimator__ccp_alpha': 0.0,
 'base_estimator__class_weight': None,
 'base_estimator__criterion': 'gini',
 'base_estimator__max_depth': 10,
 'base_estimator__max_features': None,
 'base_estimator__max_leaf_nodes': None,
 'base_estimator__min_impurity_decrease': 0.0,
 'base_estimator__min_impurity_split': None,
 'base_estimator__min_samples_leaf': 1,
 'base_estimator__min_samples_split': 2,
 'base_estimator__min_weight_fraction_leaf': 0.0,
 'base_estimator__random_state': None,
 'base_estimator__splitter': 'best',
 'base_estimator': DecisionTreeClassifier(max_depth=10),
 'learning_rate': 1.0,
 'n_estimators': 500,
 'random_state': None}
```

In [56]:
```
mse = mean_squared_error(y_test, Y_predTestboos)
print('Mean Squared Error:',mse)
```

```
Mean Squared Error: 0.012097042206713194
```

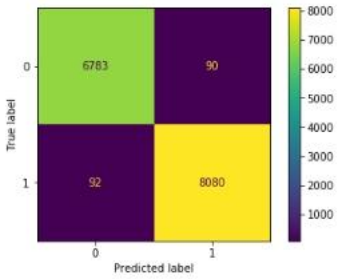
In [57]:
```
print(accuracy_score(y_test, Y_predTestboos))
```

```
0.9879029577932869
```

In [58]:
```
print(classification_report(y_test,Y_predTestboos))
```

```
              precision    recall  f1-score   support

           0       0.99      0.99      0.99      6873
           1       0.99      0.99      0.99      8172

    accuracy                           0.99     15045
   macro avg       0.99      0.99      0.99     15045
weighted avg       0.99      0.99      0.99     15045
```

In [59]:
```
confusion_matrix = metrics.confusion_matrix(y_test,Y_predTestboos)

cm_display = metrics.ConfusionMatrixDisplay(confusion_matrix = confusion_matrix, display_labels = [0, 1])

cm_display.plot()
plt.show()
```



In [95]:
```
# Define a range of complexity (e.g., different values of n_estimators)
complexity_values = [10, 50, 100, 200,350, 500]

# Initialize lists to store training and validation accuracies
train_accuracies = []
val_accuracies = []

# Train and evaluate ensembles for different levels of complexity
for complexity in complexity_values:
    # Create and train the bagging ensemble
    base_estimator = DecisionTreeClassifier()
    model = BaggingClassifier(base_estimator=base_estimator, n_estimators=complexity)
    model.fit(X_train, y_train)

    # Predict labels for training and validation sets
    train_predictions = model.predict(X_train)
    val_predictions = model.predict(X_test)

    # Calculate accuracies
    train_accuracy = accuracy_score(y_train, train_predictions)
    val_accuracy = accuracy_score(y_test, val_predictions)

    # Append accuracies to the lists
    train_accuracies.append(train_accuracy)
    val_accuracies.append(val_accuracy)

# Plot the graph
plt.plot(complexity_values,train_accuracies,'ro-',complexity_values,val_accuracies,'bv--')
plt.legend(['Training Accuracy','Test Accuracy'])
plt.xlabel('Max depth')
plt.ylabel('Accuracy')
```

Out[95]:
```
Text(0,0.5,'Accuracy')
```