11.2 Lesson Plan: Trees and Ensemble Learning

Overview

By the end of today's class, students will recognize the benefits of using tree-based algorithms for classifications problems. Also, students will gain hands-on experience with random forests and ensemble methods such as bagging and boosting.

Today's lesson also introduces students to dealing with categorical data in machine learning. Students will be able to identify when it is worth to use categorical data as a feature in a model.

Class Objectives

By the end of class, students will be able to:

- Identify when categorical variables are useful for a machine learning algorithm.
- Perform feature engineering on categorical features and convert labels to numerical class representations.
- Recognize the type of business cases where decision trees and random forests are a suitable solution for classification problems.
- Demonstrate how random forest performs better than decision trees by avoiding overfitting.
- Identify the pros and cons of tree-based algorithms.
- Understand the implications of overfitting and how boosting and bagging can help to deal with it.
- Apply Gradient Tree Boosting models in classification problems.

Instructor Notes

- Today's class is focused on teaching students how tree-based algorithms can be used for classification problems. Students start with an introduction to decision trees and are then introduced to Ensemble Learning algorithms such as Random Forests and Gradient Boosted Trees.
- Tree-based algorithms have a wide range of applications, but today's class will use them for risk analysis scenarios.
- Some of the demos in Today's class will use a lot of memory to train the models which may throw warning messages in Jupyter. Reassure students that these warnings are

typically not critical and can mostly be ignored.
- Overfitting is a common problem in machine learning that will be discussed today, so take your time to understand its implications and how the techniques covered in this class can help to avoid it.

Time Tracker

- Time Tracker

15. Instructor Do: Gradient Boosted Tree (10 min)

Corresponding Activity: Ins_Gradient_Boosted_Tree

The instructor will provide a demonstration on how to use **boosting** in **sklearn** to improve the performance of a decision tree.

File: gradient_boosted_tree_solved.ipynb

Open the unsolved file, and live code the following. Make sure to touch upon the below discussion points while coding.

- It is important to remember that **boosting** involves a set of meta-algorithms that are used to improve the performance of **weak learners**.
- There are a number of algorithms/libraries available. This activity and the next will focus on how to use the **sklearn** `GradientBoostingClassifier` algorithm.
- The `GradientBoostingClassifier` is a part of the `sklearn.ensemble` package. Like any other **sklearn** library, it has to be imported into the Python environment.

```
import pandas as pd
from path import Path
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import GradientBoostingClassifier
```

- Remind students that data has already been normalized/standardized with categories encoded. The `sklearn.preprocessing` `StandardScaler` functions were used to do this.
- The `GradientBoostingClassifier` has four main arguments: `n_estimators`, `learning_rate`, `max_depth`, and `random_state`. Explain each of these parameters while configuring them.
    - The `n_estimators` parameter configures the number of **weak learners** being used with the **boosting** algorithm. The higher the value of `n_estimators`, the

more trees that will be created to train the algorithm. The more trees, the better the performance.

  - Learning_rate controls overfitting. Smaller values should be used when setting learning_rate. Learning_rate will work with n_estimators to identify the number of **weak learners** to train.
    - The values should be between 0 and 1.
    - A common technique is to loop through a range of **learning rates**, creating and fitting the classifier with each value in the range. Once the classifier is created, it can be scored. The **learning rate** with the highest test accuracy should be used.
  - The max_depth argument identifies the size/depth of each decision tree being used. max_depth will dictate the number of levels between leaf nodes and the root.

Explain that using the GradientBoostingClassifier is like using any other machine learning algorithm: it requires training data, fitting, and scoring.

- The GradientBoostingClassifier will require values for arguments n_estimators, learning_Rate, and max_depth. The defaults will be used for n_estimators and max_depth.
- In order to determine the optimal learning_rate, a loop is used to iterate over each possible learning_rate, and then the model is built and scored using that value. The learning rate with the highest test accuracy should be chosen.

```
# Create a classifier object
learning_rates = [0.05, 0.1, 0.25, 0.5, 0.75, 1]
for learning_rate in learning_rates:
    classifier = GradientBoostingClassifier(
        n_estimators=100,
        learning_rate=learning_rate,
        max_features=2,
        max_depth=3,
        random_state=0
    )

    # Fit the model
    classifier.fit(X_train_scaled, y_train.ravel())
    print("Learning rate: ", learning_rate)

    # Score the model
```

```
print("Accuracy score (train): {0:.3f}".format(
    classifier.score(
        X_train_scaled,
        y_train.ravel())))
print("Accuracy score (test): {0:.3f}".format(
    classifier.score(
        X_test_scaled,
        y_test.ravel())))
print()
```

Output:

```
Learning rate:  0.05
Accuracy score (train): 0.717
Accuracy score (test): 0.536

Learning rate:  0.1
Accuracy score (train): 0.739
Accuracy score (test): 0.528

Learning rate:  0.25
Accuracy score (train): 0.808
Accuracy score (test): 0.544

Learning rate:  0.5
Accuracy score (train): 0.845
Accuracy score (test): 0.552

Learning rate:  0.75
Accuracy score (train): 0.853
Accuracy score (test): 0.560

Learning rate:  1
Accuracy score (train): 0.888
Accuracy score (test): 0.544
```

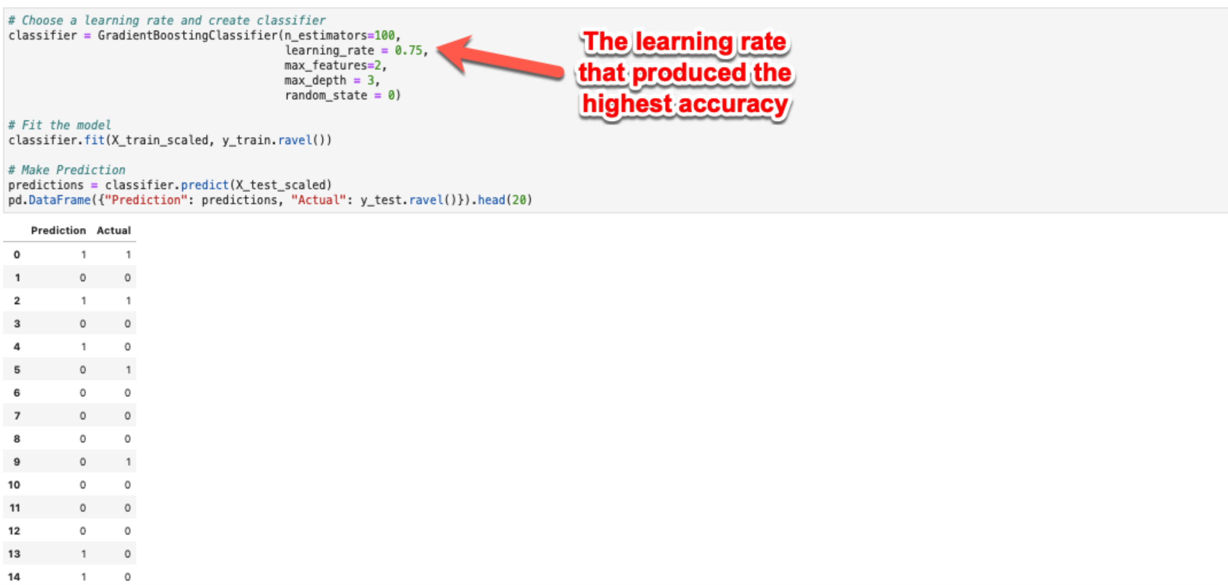
- The **learning rate** of `0.75` resulted in the highest test accuracy. Create a new classifier using this learning rate. Then, fit the model, score it, and then make predictions using the test data.

```
# Choose a learning rate and create classifier
classifier = GradientBoostingClassifier(
    n_estimators=100,
```

```
    learning_rate=0.75,
    max_features=2,
    max_depth=3,
    random_state=0
)

# Fit the model
classifier.fit(X_train_scaled, y_train.ravel())

# Make Prediction
predictions = classifier.predict(X_test_scaled)
pd.DataFrame({"Prediction": predictions, "Actual": y_test.ravel()}).head(20)
```



```
# Choose a learning rate and create classifier
classifier = GradientBoostingClassifier(n_estimators=100,
                                        learning_rate = 0.75,
                                        max_features=2,
                                        max_depth = 3,
                                        random_state = 0)

# Fit the model
classifier.fit(X_train_scaled, y_train.ravel())

# Make Prediction
predictions = classifier.predict(X_test_scaled)
pd.DataFrame({"Prediction": predictions, "Actual": y_test.ravel()}).head(20)
```


	Prediction	Actual
0	1	1
1	0	0
2	1	1
3	0	0
4	1	0
5	0	1
6	0	0
7	0	0
8	0	0
9	0	1
10	0	0
11	0	0
12	0	0
13	1	0
14	1	0

- Determine the accuracy rate using the `accuracy_score` function.

```
# Calculating the accuracy score
acc_score = accuracy_score(y_test, predictions)
print(f"Accuracy Score : {acc_score}")
```

```
Accuracy Score : 0.56
```

- Evaluate the performance of the model by generating a **confusion matrix** and **classification report**.

```
# Generate the confusion matrix
cm = confusion_matrix(y_test, predictions)
cm_df = pd.DataFrame(
    cm, index=["Actual 0", "Actual 1"],
    columns=["Predicted 0", "Predicted 1"]
)

# Displaying results
display(cm_df)

# Generate classification report
print("Classification Report")
print(classification_report(y_test, predictions))
```

```
Classification Report
         precision    recall  f1-score   support

      0       0.69      0.62      0.65       84
      1      0.36      0.44      0.40       41

accuracy                          0.56       125
macro avg       0.53      0.53      0.52       125
weighted avg       0.58      0.56      0.57       125
```

Ask if there are any questions before moving on.

---

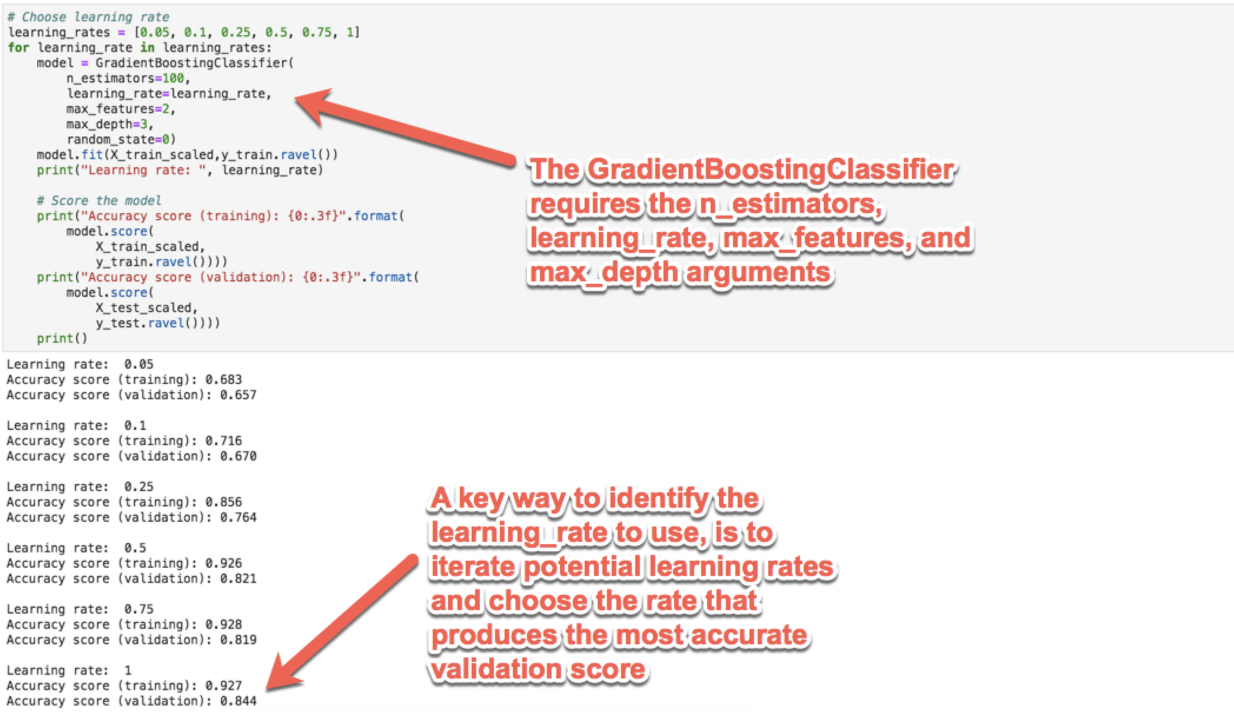
17. Instructor Do: Turbo Boost Activity Review (10 min)

Files: turbo_boost_solved.ipynb

Open the solution and explain the following:

- The GradientBoostedClassifier model was able to produce incredibly high accuracy scores, higher than some of the algorithms we have seen. What about the GradientBoostedClassifier makes it perform better than some other algorithms?
  - **Answer** GradientBoostClassifier is an **ensemble learning** algorithm. It pools **weak learners** together and executes them in parallel in order to refit the model

as needed. Because it leverages multiple algorithms and runs them in parallel, GradientBoostClassifier is a more robust algorithm than average.



- Even though the accuracy score was high, the classification report shows the precision and recall for detecting one class was greater than the classification for the other class.
    - Explain that this is because the classes are imbalanced, meaning that the algorithm was able to make predictions for one class better than it was for another, and as a result, the algorithm developed bias.
    - Let students know that they will learn what imbalanced classes are and how to deal with them in the next class.
- What are the three main parameters for the GradientBoostClassifier model?
    - **Answer n_estimators, learning_rate,** and **max_depth.**
        - n_estimators determines the number of trees/weak learners to use.
        - learning_rate identifies how aggressive the algorithm will learn.
        - max_depth dictates the size of each tree.

- Remind students that **boosting** algorithms are supervised learning algorithms, and they are built and trained just like any other algorithm. They can perform better than other algorithms because they make iterative predictions using more than one classifier.

Use the rest of the time for students to ask questions. If there are no questions, ask students how they are feeling about decision trees and **boosting** algorithms.

Move onto the next activity.

End Section