University of Waterloo
Econ 424
2023-09-12
Anonymized Usernmae: IJustWannaPass

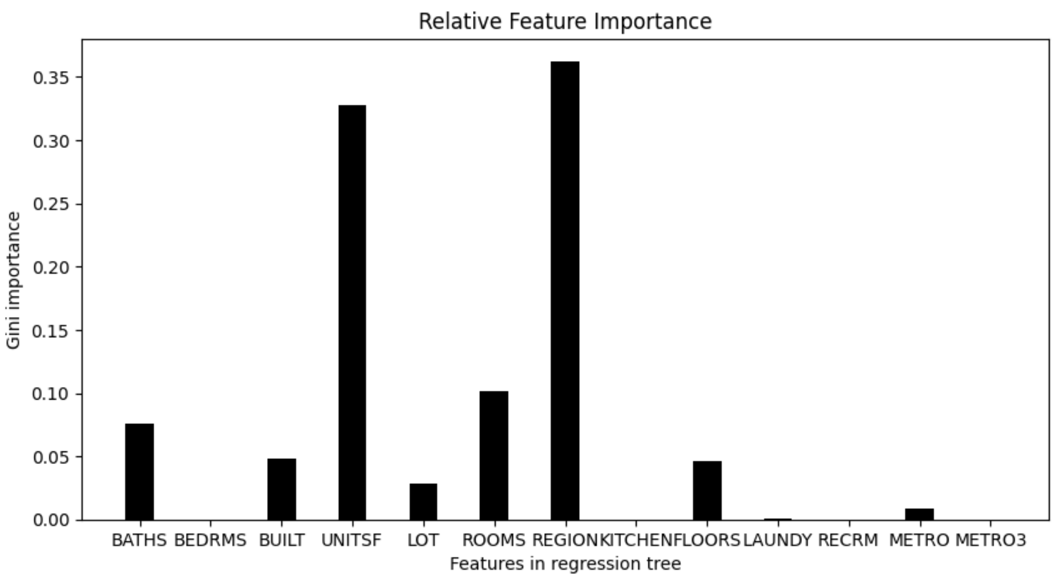
Prediction Competition # 2

Q1) For our training data we have the following values on our training data (done using a 10-90 split of test vs training), averaged across 100 different runs:

$$R^2 = 0.73569$$
$$MSE = 0.47404$$

The hyper parameters of this model where turned by comparing all iterations across 10 different splits of training and test data.

Q2) The relative importance of the features in the final model can be seen below:



Due to the fact that each leaf needs to have 50 observations alot of features arent used (and thus have zero importance) as the tree is not deep enough to make use of them.

Q3) To begin with question 1, I asked chatGPT 3.5 the following prompt:

R

Given 20,000 observations and 12 features, how can I code a regression tree algorithm to predict the expected value

To which chatGPT gave the following response:

‹ 2 / 2 ›

Creating a regression tree algorithm from scratch can be quite involved, but I can provide you with a high-level overview of the steps involved and some Python code using a popular library, scikit-learn, to help you get started. Keep in mind that building a robust decision tree algorithm is a complex task, and it's usually more efficient to use existing libraries for practical applications.

Here's a step-by-step guide to building a regression tree:

1. **Data Preparation:**
   - Load your dataset and split it into training and testing sets.
   - Ensure that your data is cleaned and properly formatted.
2. **Node Splitting:**
   - At each node of the tree, choose the best feature to split the data.
   - Calculate a splitting criterion, often mean squared error (MSE) for regression trees.
   - Find the feature and split point that minimize the MSE.
3. **Stopping Criteria:**
   - Define stopping criteria to determine when to stop splitting nodes.
   - This can include a maximum tree depth, a minimum number of samples per leaf node, or a minimum improvement in MSE.
4. **Create Tree Structure:**
   - Recursively split the data into child nodes until the stopping criteria are met for each branch.
5. **Prediction:**
   - For a new input, traverse the tree by following the splitting rules and return the predicted value, which is usually the mean of the target values in the leaf node.

Here's a simplified example of how to implement a regression tree using scikit-learn:

python Copy code

```python
from sklearn.tree import DecisionTreeRegressor

# Create a DecisionTreeRegressor with desired hyperparameters
regressor = DecisionTreeRegressor(max_depth=5, min_samples_leaf=10)

# Fit the model to your training data
regressor.fit(X_train, y_train)

# Make predictions on test data
y_pred = regressor.predict(X_test)
```

In the code above:

- `DecisionTreeRegressor` is used to create a regression tree model.
- `max_depth` controls the maximum depth of the tree to prevent overfitting.
- `min_samples_leaf` sets the minimum number of samples required in a leaf node.

Regenerate

As for question 2, I began with the prompt:

> R
>
> Given a decision regression tree using sklearn, how can I make a graph depicting the importance of each feature in the model

To which chatGPT replied:

> To create a graph depicting the feature importances in a decision tree regression model built using scikit-learn, you can follow these steps:
>
> 1. First, train your decision tree regression model on your dataset using scikit-learn.
> 2. After training the model, you can access the feature importances using the `feature_importances_` attribute of the trained model.
> 3. Next, you can create a bar chart or any other type of visualization to display the feature importances.
>
> Here's a step-by-step example using Python and scikit-learn:

```python
import numpy as np
import matplotlib.pyplot as plt
from sklearn.tree import DecisionTreeRegressor

# Create or load your dataset and split it into features (X) and target (y)

# Create and train your DecisionTreeRegressor
regressor = DecisionTreeRegressor(max_depth=5)
regressor.fit(X, y)

# Get feature importances
feature_importances = regressor.feature_importances_

# Sort feature importances in descending order and get the corresponding fea
feature_names = np.array(["Feature1", "Feature2", "Feature3", ...])  # Repla
sorted_idx = np.argsort(feature_importances)[::-1]
sorted_feature_importances = feature_importances[sorted_idx]
sorted_feature_names = feature_names[sorted_idx]

# Create a bar chart to visualize feature importances
plt.figure(figsize=(10, 6))
plt.bar(range(len(sorted_feature_importances)), sorted_feature_importances,
plt.xticks(range(len(sorted_feature_importances)), sorted_feature_names, rot
plt.xlabel("Feature")
plt.ylabel("Feature Importance")
plt.title("Feature Importances in Decision Tree Regression Model")
plt.show()
```

> In this example:
>
> - `feature_importances` contains the importance scores of each feature.
> - `feature_names` should be an array or list containing the names of your features.
> - `sorted_idx` is used to sort the features by importance in descending order.
> - The bar chart is created using Matplotlib to visualize the feature importances.

Code for Q1:

```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import heapq

from sklearn.tree import DecisionTreeRegressor
from sklearn.model_selection import train_test_split
from scipy import stats

data = pd.read_csv("Econ424_F2023_PC2_training_set_v1.csv", low_memory=False)

# Clean Data
data = data[(np.abs(stats.zscore(data)) < 4).all(axis=1)]

#  =============== PART 1 (Test the model) ===============
X_Train = [0]*100
Y_Train = [0]*100
X_Test = [0]*100
Y_Test = [0]*100

# Load up an array of test and training data

for i in range(0,100):
    train, test = train_test_split(data, test_size=0.1)

    Y_Train[i] = train.iloc[:, 0]
    X_Train[i] = train.iloc[:, 1:]
    Y_Test[i] = test.iloc[:, 0]
    X_Test[i] = test.iloc[:, 1:]

mse = [0]*100
TSS = [0]*100
RSS = [0]*100
regr_1 = DecisionTreeRegressor(max_depth=6, min_samples_leaf=50,min_samples_split=19)
for i in range(0,100):
    regr_1.fit(X_Train[i], Y_Train[i]) # Fit the regression tree with the corresponding values

    predicted_vals = regr_1.predict(X_Test[i])
    r2_mean = np.mean(Y_Test[i].iloc[i])
    for x in range(0, len(predicted_vals)):
        # Calculation for MSE
        mse[i] += ((predicted_vals[x] - Y_Test[i].iloc[x])**2)/len(predicted_vals)
        # Calculation for RSS
        RSS[i] += ((predicted_vals[x] - Y_Test[i].iloc[x])**2)
        TSS[i] += ((r2_mean - Y_Test[i].iloc[x])**2)

print("MSE: " + str(np.mean(mse)))
print("R2: " + str(1-(np.mean(RSS)/np.mean(TSS))))
#  =============== PART 2 (Predict the data) ===============

# Load Data for Prediction In
X_Test = pd.read_csv("Econ424_F2023_PC2_test_set_without_response_variable_v1.csv",
    low_memory=False)
X_Test.rename(str.upper, axis='columns', inplace=True)

Y_Train = data.iloc[:, 0]
X_Train = data.iloc[:, 1:]
```

```python
# Train the decision tree with the optimal values
regr_1.fit(X_Train, Y_Train)

estimates = regr_1.predict(X_Test)

# Write the output
f = open('predictions.csv', 'w')
for estimate in estimates:
    f.writelines(str(estimate) + ",\n")
```

Code for Q2:

```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import heapq

from sklearn.tree import DecisionTreeRegressor
from sklearn.model_selection import train_test_split
from scipy import stats

data = pd.read_csv("Econ424_F2023_PC2_training_set_v1.csv", low_memory=False)

# Clean Data
data = data[(np.abs(stats.zscore(data)) < 4).all(axis=1)]

regr_1 = DecisionTreeRegressor(max_depth=6, min_samples_leaf=50,min_samples_split=19)

Y_Train = data.iloc[:, 0]
X_Train = data.iloc[:, 1:]

X_Test = pd.read_csv("Econ424_F2023_PC2_test_set_without_response_variable_v1.csv",
    low_memory=False)
X_Test.rename(str.upper, axis='columns', inplace=True)

regr_1.fit(X_Train, Y_Train)


plt.figure(figsize=(10, 5))

plt.bar(regr_1.feature_names_in_, regr_1.feature_importances_, color='black',
       width=0.4)

plt.xlabel("Features in regression tree")
plt.ylabel("Gini importance")
plt.title("Relative Feature Importance")
plt.show()
```