Tree-Based Methods

Machine Learning for Economics and Finance
Bachelor in Economics

Marcel Weschke

27.05.2025

Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

Roadmap

- Introduction to tree-based methods
- Regression trees
- Pruning
- Classification trees
- Combining multiple trees
    - Bagging
    - Random Forest
    - Variable importance measures

Learning Goals

At the end of this lecture, you should:

- Be able to use tree-based methods to solve prediction problems (both regression and classification)
- Have a good understanding of decision trees including interpreting them
- Be able to use cross-validation to evaluate the performance of tree-based method
- Understand how the combination of multiple trees can increase predictive power
- Be able to apply the above concepts above in R

Book Chapter: 8

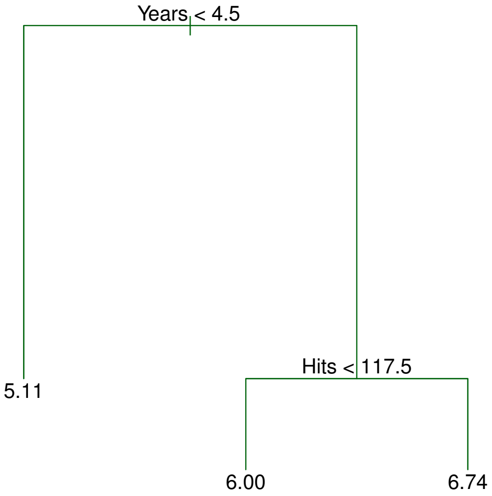
Big picture

- Tree-based methods are simple tool to solve supervised learning problems
- Even though they can capture non-linear relationships, they are easy to interpret
- But simple trees are typically not competitive with the best supervised learning approaches in terms of prediction accuracy
- The Interpretability vs. “black-box” tradoff
- Combining multiple trees often results in higher prediction accuracy, but loses some interpretation
- Popular approaches:
  - Bagging
  - Random Forest
  - Boosting (not covered in this course)

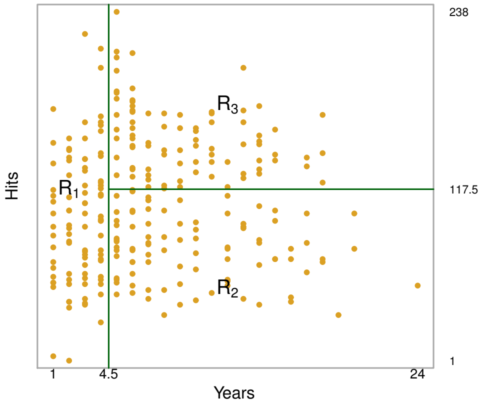
The basic idea

- We use the Hitters data as an introductory example
- The goal is to predict the log salary of a baseball player
- As predictors we use the players' years in the major league and the number of hits in the previous year
- A tree consists of a series of splitting rules that start at the top of the tree
- For example: if the number of years is smaller than 4.5, the predicted (log) salary is $5.11.

Decision tree for log salary



Segment the predictor space



Segment the predictor space cont'

- The tree segments the players into three regions of predictor space:
  - $R_1 = \{X|\text{Years} < 4.5\}$
  - $R_2 = \{X|\text{Years} \geq 4.5, \text{Hits} < 117.5\}$
  - $R_3 = \{X|\text{Years} \geq 4.5\,\text{Hits} \geq 117.5\}$
- Predicted salaries:
  - $R_1 : \$1.000e^{5.107} = \$165,174$
  - $R_2 : \$1.000e^{5.999} = \$402,834$
  - $R_3 : \$1.000e^{6.740} = \$845,346$

Summarizing the results

- Years is the most important factor in determining Salary
- Players with less experience earn lower salaries
- Given that a player is less experienced, the number of Hits that he made in the previous year seems to play little role in explaining his Salary
- But among players who have been in the major leagues for five or more years, the number of Hits made in the previous year is positively related to Salary

Terminology

- Features: Number of predictors
- Root node: The starting point of the tree
- Splitting: The process of dividing a node into two or more sub-nodes
- Branch/Sub-Tree: A subsection of the entire tree
- Internal nodes: The points along the tree where the predictor space is split
- Terminal nodes/Leafs: Nodes that do not split
- Size: Number of terminal nodes

The tree-building process

- We divide the predictor space, $X_1, X_2, ..., X_p$, into J distinct and non-overlapping regions, $R_1, R_2, ..., R_j$
- For every observation in region R_j, we make the same prediction
- The prediction is the mean of the response values in the region
- Want to find boxes $R_1, R_2, ..., R_j$ that minimize the RSS

$$RSS = \sum_{j=1}^{J} \sum_{i \in R_j} (y_i - \hat{y}_{R_j})^2 \tag{1}$$

- With $\hat{y}_{R_j}$ being the mean response within the jth box

The tree-building process

- How do we minimize RSS?

$$
\begin{aligned}
RSS = & \sum_{j=1}^{J} \sum_{i \in R_j} (y_i - \hat{y}_{R_j})^2 & (2) \\
= & \sum_{i \in R_1} (y_i - \hat{y}_{R_1})^2 + ... + \sum_{i \in R_J} (y_i - \hat{y}_{R_J})^2 & (3) \\
= & \text{Sum of MSE across all terminal nodes} & (4)
\end{aligned}
$$

- Computationally impossible to consider every possible partition of the feature space into J boxes

An intuitive algorithm

- Solution is to take a top-down greedy approach
    - **Top-down**: because it begins at the top of the tree and then successively splits the predictor space
    - Each split is indicated via two new branches further down on the tree
    - **Greedy**: because at each step of the tree-building process, the best split (e.g. highest improvement in RSS) is made at that particular step, rather than looking ahead and picking a split that will lead to a better tree in some future step
- Stop after a specific criterion is reached; for instance, we may continue until no region contains more than five observations
- **Problem**: This approach often leads to large trees that overfit the data

An intuitive algorithm

```python
from sklearn.tree import DecisionTreeRegressor
from sklearn.metrics import mean_squared_error

tree_model = DecisionTreeRegressor(random_state=1)
tree_model.fit(X_train, y_train)  # Fit a large tree

print(f"Nr. of leaves: {tree_model.get_n_leaves()}")
print(f"Get max. depth: {tree_model.get_depth()}")

y_pred = tree_model.predict(X_test) # Make pred.
```

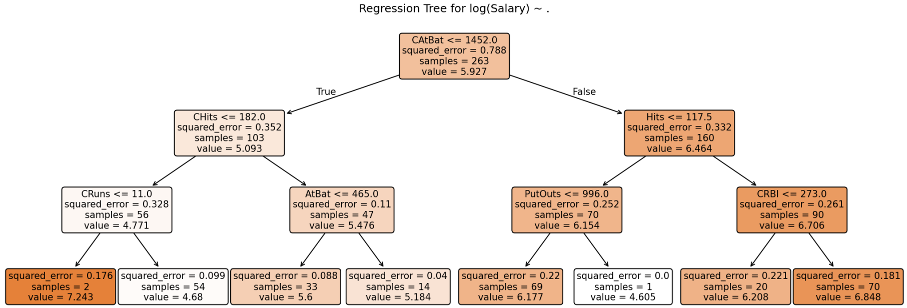
Hitters data: large tree

```python
# Load and preprocess data
Hitters = load_data('Hitters').dropna()
Hitters = pd.get_dummies(Hitters, drop_first=True)

y = np.log(Hitters['Salary'])
X = Hitters.drop(columns=['Salary'])

# Fit a large tree
# Note: We limit the tree to a maximum depth of 3
tree_model = DecisionTreeRegressor(random_state=1, max_depth=3)
tree_model.fit(X, y)
```

Hitters data: large tree



```python
import matplotlib.pyplot as plt
fig, ax = plt.subplots(figsize=(12, 6))
plot_tree(tree_model,
     feature_names=X.columns,
     filled=True,
     rounded=True,
     fontsize=10,
     ax=ax)
plt.show()
```

Find the right tree size by Pruning

- **Idea**:
  - First grow a large tree T_0
  - Then sequentially cut off some of the terminal nodes/leafs ("prune" it back)
  - This is done via the non-negative tuning parameter (cost-complexity parameter) α

Find the right tree size by Pruning

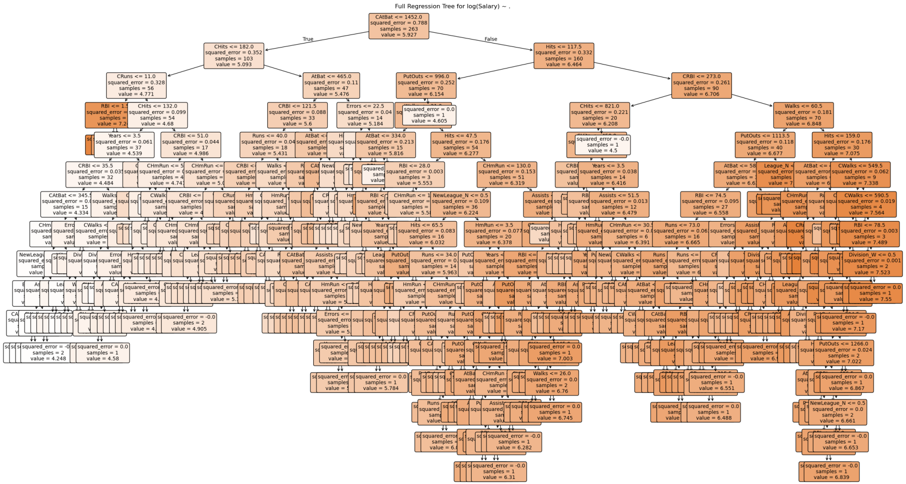
- **Approach**:
  - For each value of α there corresponds a subtree $T \in T_0$ such that the below sum is minimized

$$\sum_{m=1}^{|T|} \sum_{x_i \in R_m} (y_i - \hat{y}_{R_m})^2 + \alpha |T| \tag{5}$$

  - $|T|$ indicates the number of terminal nodes of the tree T
  - R_m is the "box" corresponding to the *m*th terminal node
  - $\hat{y}_{R_m}$ is the mean of the training observations in R_m
  - With increasing α, more and more nodes (leafs) will be "cut off" resulting in a smaller tree

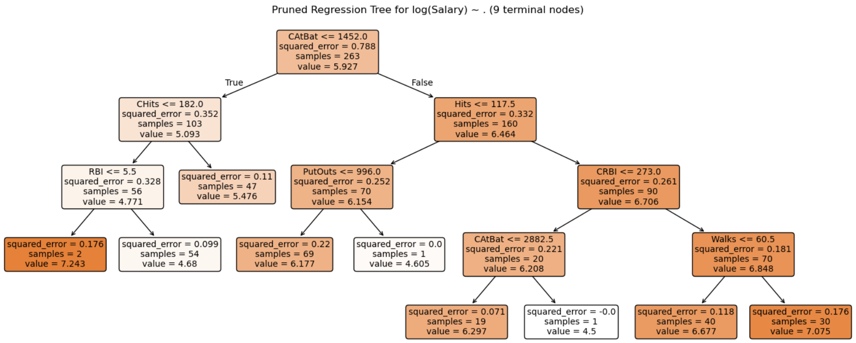
```
## prune tree to 4 terminal nodes
tree_model_pruned = DecisionTreeRegressor(random_state=1,
↪  max_leaf_nodes=4)
```

Hitters data: unpruned tree



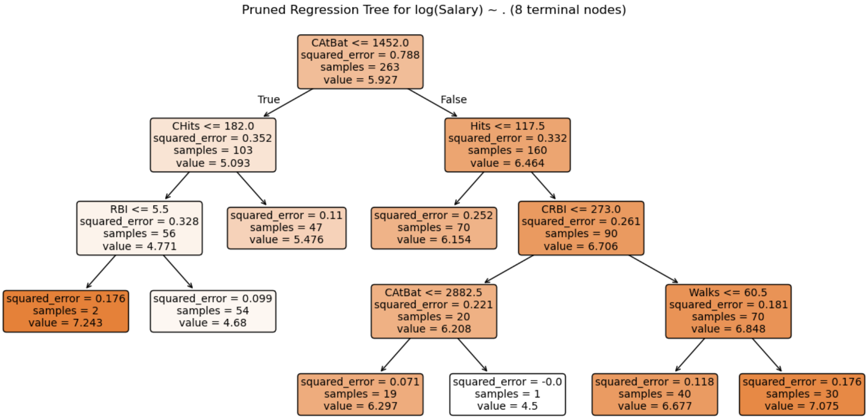
```
tree_model = DecisionTreeRegressor(random_state=1)
tree_model.fit(X, y)
```

Hitters data: pruned tree with 9 terminal leafs



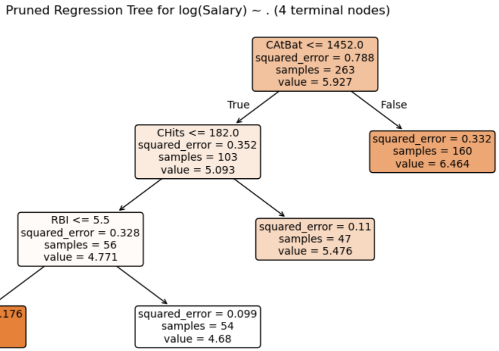
```
tree_model = DecisionTreeRegressor(max_leaf_nodes=9, random_state=1)
tree_model.fit(X, y)
```

Hitters data: pruned tree with 8 terminal leafs



```
tree_model = DecisionTreeRegressor(max_leaf_nodes=8, random_state=1)
tree_model.fit(X, y)
```

Hitters data: pruned tree with 4 terminal leafs



```python
tree_model = DecisionTreeRegressor(max_leaf_nodes=4, random_state=1)
tree_model.fit(X, y)
```

Choosing the best subtree

- The tuning parameter α controls a trade-off between the subtree's complexity and its fit to the training data
- We select an optimal value $\hat{\alpha}$ using cross-validation
- We then return to the full data set and obtain the subtree corresponding to $\hat{\alpha}$

```python
path = tree_model.cost_complexity_pruning_path(X, y)
ccp_alphas = path.ccp_alphas

# Cross-Validation-results for range of prunings
results = []
for alpha in ccp_alphas:
  model = DecisionTreeRegressor(random_state=1, ccp_alpha=alpha)
  scores = cross_val_score(model, X, y, cv=5,
  ↪  scoring='neg_mean_squared_error')
  results.append({
   'ccp_alpha': alpha,
   'cv_mse': -np.mean(scores),
   'leaves': model.fit(X, y).get_n_leaves()
})
```

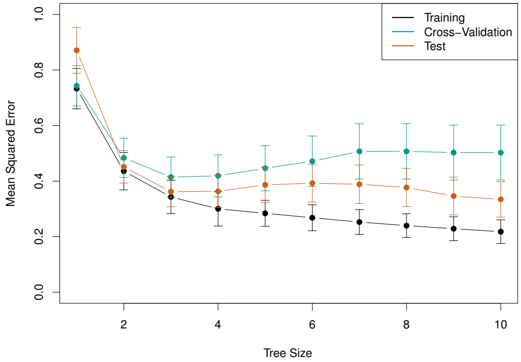
Choosing the best subtree

```
# Print as table
cv_results = pd.DataFrame(results).sort_values(by='leaves',
↪  ascending=False)
print(cv_results)

# Output:
           ccp_alpha    cv_mse  leaves
  0     0.000000e+00  0.401226     245
  1     2.701683e-17  0.401226     244
  2     5.403367e-17  0.401226     243
  3     8.105050e-17  0.401226     241
  4     8.105050e-17  0.401226     241
  ..             ...       ...     ...
  234   2.424895e-02  0.279697       5
  235   4.551431e-02  0.322510       4
  236   4.820091e-02  0.351775       3
  237   4.827370e-02  0.351775       2
  238   4.481278e-01  0.669474       1

  [239 rows x 3 columns]
  Best number of leaves: 10
```

Hitters data: pruning and cross validation



Tree algorithm

- Grow a large tree on the training data (recursive binary splitting)
- Stop when each terminal node has fewer than some minimum number of observations
- Apply cost complexity pruning to the large tree in order to obtain a sequence of best subtrees, as a function of α
- Use K-fold cross-validation to choose α.

- For each $k = 1, ..., K$:
    - Repeat above Steps on the $(K-1)/K$th fraction of the training data, excluding the kth fold.
    - Evaluate the mean squared prediction error on the data in the left-out kth fold, as a function of α

- Average the results, and pick α to minimize the average error
- Return the subtree that corresponds to optimal α

Task 1:

Start with the R-File *05_Trees_Hitters_Task.R* (or *05_Trees_Hitters_Task.Rmd*)

1. Use the Hitters data and remove all rows that contain missing values. Create a new variable that is the log of Salary and provide histograms for Salary and Log(Salary). Interpret.
2. Split the sample into a training dataset consisting of the first 200 observations and a test dataset containing the remaining observations.
3. Fit a large, unpruned regression tree to predict Log(Salary). Which features are used to construct the tree, which features are the most important and how many terminal nodes does the tree have? You might want to plot the tree for this exercise.
4. Compute the mean squared prediction error for the test data.
5. Let's try to improve predictions using k-fold CV. Set the seed to 2 and run 5-fold cross validation. Plot the mean squared cross validation error against the tree size and report the tree size and the pruning parameter α that minimize the mean squared cross validation error.
6. Use the pruning parameter from the previous task to prune the tree. Plot the tree and report the most important variables.
7. Compute the test mean squared prediction error for pruned tree and compare to the results from Task 4.

Roadmap

- Introduction to tree-based methods
- Regression trees
- Pruning
- **Classification trees**
- Combining multiple trees
  - Bagging
  - Random Forest
  - Variable importance measures

Classification Trees

- Very similar to a regression tree, except that it is used to predict a qualitative response rather than a quantitative one
- For a classification tree, we predict that each observation belongs to the most commonly occurring class of training observations in the region to which it belongs
- Ideally we want to build a tree that has terminal nodes with many observations but high“node purity”
- A high node purity means that a terminal node consists predominantly of observations of a single class (high prediction accuracy)
- **Q**: Why do we care about whether such a terminal node has many observations or not?

Details of classification trees

- Just as in the regression setting, we use recursive binary splitting to grow a classification tree.
- In the classification setting, RSS cannot be used as a criterion for making the binary splits
- A natural alternative is to use the fraction of the training observations in that region that do not belong to the most common class, the error rate:

$$E = 1 - \max_k(\hat{p}_{mk}) \tag{6}$$

- $\hat{p}_{mk}$ represents the proportion of training observations in the mth region that are from the kth class

Gini index and Deviance

- In practice classification error is not sufficiently sensitive for tree-growing.
- Better metrics are the Gini index and Devince (or Entropy
- The Gini index is defined by

$$G = \sum_{k=1}^{K} \hat{p}_{mk}(1 - \hat{p}_{mk}) \tag{7}$$

- It measures the total variance across the K classes.
- A small G indicates that a node contains predominantly observations from a single class (node purity)

```python
# Fit classification tree using gini as splittin criterion
clf_gini = DecisionTreeClassifier(criterion='gini', random_state=0)
clf_gini.fit(X_train, y_train)
```

Note: y must be a factor (use as.factor() to transform numeric to factor)

Gini index and Deviance

- The Deviance is very similar and given by

$$D = -\sum_{k=1}^{K} \hat{p}_{mk} \ln(\hat{p}_{mk}) \tag{8}$$

```python
# Fit classification tree using deviance as splittin criterion
clf_entropy = DecisionTreeClassifier(criterion='entropy', random_state=0)
clf_entropy.fit(X_train, y_train)
```

Example: Sales of Carseats

- The Carseats dataset is part of the ISLR2 package
- Simulated dataset with 400 observations on the following 11 variables:
    - Sales: Unit sales (in thousands) at each location
    - CompPrice: Price charged by competitor at each location
    - Income: Community income level (in thousands of dollars)
    - Advertising: Local advertising budget for company at each location (in thousands of dollars)
    - ...
- Goal: predict whether sales are higher or smaller than 8,000
- File: 05_Trees_Carseats.R

Summary

+ Trees can be displayed graphically
+ Trees are very easy to explain to people
+ Closely mirror human decision-making?
+ Trees can easily handle qualitative predictors without the need to create dummy variables.

– Trees generally do have lower levels of predictive accuracy than other machine learning methods

Roadmap

- Introduction to tree-based methods
- Regression trees
- Pruning
- Classification trees
- **Combining multiple trees**
    - Bagging
    - Random Forest
    - Variable importance measures

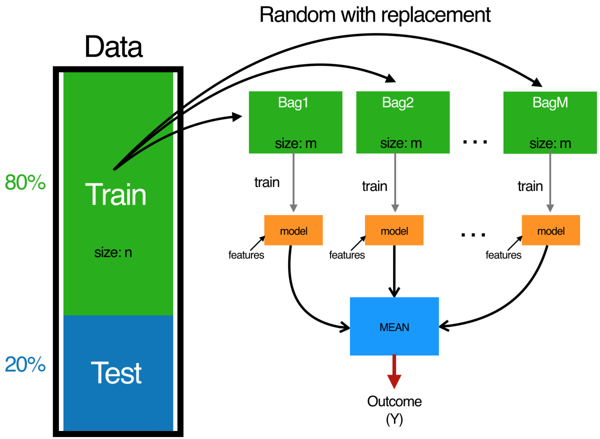
Combining multiple trees

- Question: How can we improve the predictive power of trees?
- We will discuss two popular approaches:
  - Bagging
  - Random Forest
  - (Boosting is not covered in this course)

Bagging

- Bootstrap aggregation ("bagging") is a method used to reduce the tendency of decision trees to overfit to a specific data sample
- Idea: Recall that for a sequence of n independent observations $Z_1, ..., Z_n$, each with variance σ^2, the variance of the mean goes to zero as n becomes sufficiently large
- The above logic applied also to Trees: Average predictions obtained from trees fitted to different training datasets to reduce variance of the predictions
- But we generally do not have access to multiple training sets?!?
- Solution is to bootstrap

Bagging



Source: UC Business Analytics R Programming Guide

Bagging

- Generate B different bootstrapped training data sets with replacement $\{x_1, ..., x_B\}$
- Each bootstrapped data set uses approx. $2/3$ of data
- Fit a tree to the bth bootstrapped training set to get a total of B trees, $\hat{f}^{\star b}$
- "Average" across all B trees to form a prediction

$$\hat{f}_{bag}(x) = \frac{1}{B} \sum_{b=1}^{B} \hat{f}^{\star b}(x) \tag{9}$$

- For classification trees, we take a so-called majority vote: the overall prediction is the most commonly occurring class among the B predictions

Estimate the test MSE

- Straightforward to estimate the test error of a bagged model
- Recall we draw with replacement from the original dataset to create a new dataset “in-the-bag”
- At the same time we create an additional dataset from the observations that are not “in-the-bag”, which is “out-of-bag” (OOB)
- We can estimate the test MSE (or the test error rate) from the OOB observations
- For this, use the average prediction for an OOB observation of all trees for which this observation hasn’t been used in the training step
- This is close to k-fold cross-validation and yields a good estimate of the true prediction error rate

Random Forests

- Problem with bagging:
  - Suppose there is one feature with very high predictive power and some other features with normal predictive power
  - Trees fitted to different bagged datasets will look very similar and all choose this one feature with high predictive power for the first
  - Hence, averaging does not yield large improvements
- Random forests provide an improvement over bagged trees by way of a small tweak that decorrelates the trees
- As with bagging, we build many large trees
- But: for each split in a tree, only a random subset of m out of the p predictors is considered

Random Forests

- Hence, at each split, the algorithm can only use a reduced subset of the features
- This ensures that the trees obtained from the different bagged subsets are actually distinct (less correlated)
- Decorrelating the trees makes the average of the resulting trees less variable and hence more reliable.
- A typical choice is to set $m \approx \sqrt{p}$
- Alternative: determine optimal m using cross-validation
- Q: What happens if we set $m = p$?

Python code: Random Forest & Bagging

```python
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score

# Random Forest
rf_model = RandomForestClassifier(
  n_estimators=500,      # ntree = 500
  max_features=3,        # mtry = 3
  random_state=1,
  oob_score=True         # Optional: get OOB error
)

rf_model.fit(X_train, y_train)
rf_pred = rf_model.predict(X_test)

# Evaluate prediction
accuracy = accuracy_score(y_test, rf_pred)
print(f"Random Forest Accuracy: {accuracy:.3f}")
print(f"OOB Error Rate: {1 - rf_model.oob_score_:.3f}")
```

Python code: Random Forest & Bagging

```python
# Bagging model (mtry = p)
bag_model = RandomForestClassifier(
  n_estimators=500,
  max_features=X_train.shape[1],  # all features = bagging
  random_state=1,
  oob_score=True
)

bag_model.fit(X_train, y_train)
bag_pred = bag_model.predict(X_test)

# Evaluate prediction
bag_accuracy = accuracy_score(y_test, bag_pred)
print(f"Bagging Accuracy: {bag_accuracy:.3f}")
print(f"OOB Error Rate (Bagging): {1 - bag_model.oob_score_:.3f}")
```

Example: Sales of Carseats

Let's use random forest and bagging to improve the predictions in the Carseats data (see again file 05_Trees_Carseats.R)

Tuning Random Forest

- Quite easy to tune
- Key hyperparameter:
  - Number of trees (ntree)
  - Number of m variables randomly sampled from the p predictors (mtry)
- Other hyperparameters:
  - Sample size for bootstrapped samples (sampsize)
  - Minimum node size (tree complexity) (nodesize)
  - Maximum number of terminal nodes (maxnodes)
- $\Rightarrow$ Try different hyperparamters and choose the one that minimizes the OOB errors

Variable importance measure

- Bagging and random forests are good methods for improving the prediction accuracy of trees
- But their results can be difficult to interpret
- To understand which features matter, we record the total amount that the RSS is decreased due to splits over a given predictor, averaged over all B trees
- For classification trees, we add up the total amount that the Gini index is decreased by splits over a given predictor, averaged over all B trees
- We can then rank the variables bases on their relative influence

Variable importance measure - table

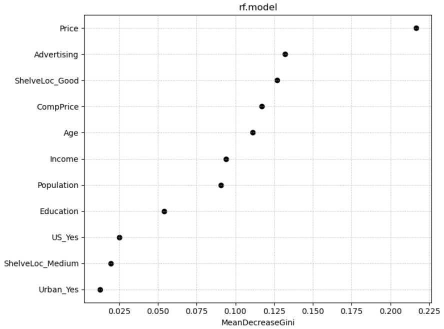
```
rf_model = RandomForestClassifier(n_estimators=500, max_features=3,
↪  random_state=1, oob_score=True)
rf_model.fit(X_train, y_train)

# create importance table
importances = rf_model.feature_importances_
feature_names = X_train.columns if isinstance(X_train, pd.DataFrame) else
↪  [f"X{i}" for i in range(X_train.shape[1])]
importance_df = pd.DataFrame({
 'Feature': feature_names,
 'MeanDecreaseGini': importances
}).sort_values(by="MeanDecreaseGini", ascending=True)

# table results
print(importance_df)
# Output:
  Feature  MeanDecreaseGini
  4              Price          0.216420
  2        Advertising          0.132140
  7     ShelveLoc_Good          0.127002
  0          CompPrice          0.116924
  5                Age          0.111315
  1             Income          0.093965
  3         Population          0.090505
```

Variable importance measure - plot

```python
plt.figure(figsize=(8, 6))
plt.scatter(importance_df['MeanDecreaseGini'], range(len(importance_df)),
↪  color='black')
plt.yticks(range(len(importance_df)), importance_df['Feature'])
plt.xlabel("MeanDecreaseGini")
plt.title("rf.model")
plt.grid(True, linestyle='dotted')
plt.tight_layout()
plt.show()
```



Task 1 - continued:

Start with the R-File you prepared for Task 1:

8. Use random forest to improve the predictions. Fit 500 trees using $m = \sqrt{p}$ (round to the nearest integer).
9. Do you think it was necessary to fit 500 trees or would have fewer trees be sufficient? Determine the number of trees that provides the lowest OOB error.
10. Compute the OOB estimate of the out-of-sample error and compare it to best pruned model from CV of Task 5. Interpret the outcomes.
11. Which are the most important variables used in the random forest?
12. Let's try to improve the random forest by trying out different values for m. Set up a grid for m going from 1 to p. Write a loop that fits a random forest for each m. Explain which model you would choose.
13. For the best model, compute the test errors and compare them to the best pruned model from Task 7.
14. What is the OOB error obtained from bagging (you can infer the answer from the previous task).