05_Trees_Hitters_Task_solved

May 5, 2025

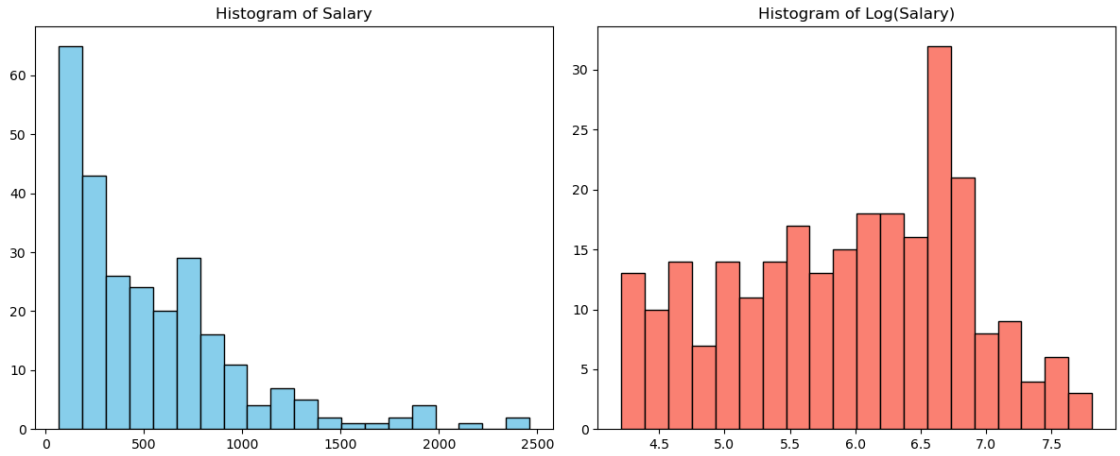
1 Preliminary setup

[1]:
```
import numpy as np
import pandas as pd
from ISLP import load_data
from matplotlib.pyplot import subplots, show
import matplotlib.pyplot as plt


# Load and preprocess data
Hitters = load_data('Hitters').dropna()
```

2 Task 1

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]:
```
# Task 1: Load and preprocess data
Hitters = load_data('Hitters').dropna()

# Add log of Salary
Hitters['LogSalary'] = np.log(Hitters['Salary'])

# Histograms
fig, axs = plt.subplots(1, 2, figsize=(12, 5))
axs[0].hist(Hitters['Salary'], bins=20, color='skyblue', edgecolor='black')
axs[0].set_title('Histogram of Salary')
axs[1].hist(Hitters['LogSalary'], bins=20, color='salmon', edgecolor='black')
axs[1].set_title('Histogram of Log(Salary)')
plt.tight_layout()
plt.show()
```

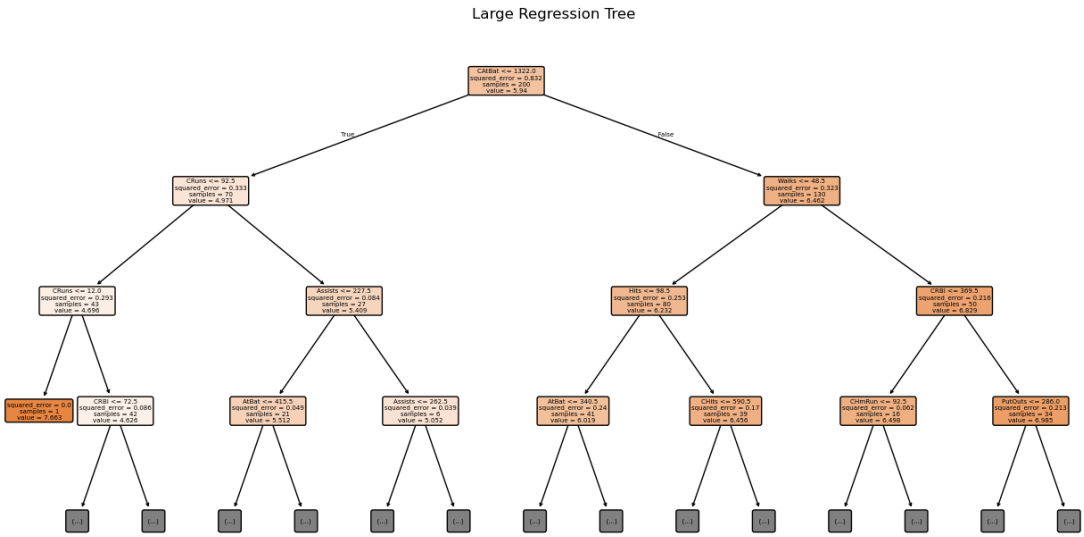


2. Split the sample into a training dataset consisting of the first 200 observations and a test dataset containing the remaining observations.

[3]:
```
# Task 2: Split the dataset
Hitters_dummies = pd.get_dummies(Hitters.drop(columns=['Salary']),␣
 ↪drop_first=True)
X_train = Hitters_dummies.iloc[:200].drop(columns='LogSalary')
y_train = Hitters_dummies.iloc[:200]['LogSalary']
X_test = Hitters_dummies.iloc[200:].drop(columns='LogSalary')
y_test = Hitters_dummies.iloc[200:]['LogSalary']
```

3. Fit a large, unpruned regression tree to predigt 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]:
```
# Task 3: Fit a large tree

from sklearn.tree import DecisionTreeRegressor, plot_tree
tree_model = DecisionTreeRegressor(random_state=1)
tree_model.fit(X_train, y_train)

# Plot the tree
plt.figure(figsize=(16, 8))
plot_tree(tree_model, feature_names=X_train.columns, filled=True, rounded=True,␣
 ↪max_depth=3)
plt.title("Large Regression Tree")
plt.show()
```



[26]:
```
# Task 3: feature importance
important_features = pd.Series(tree_model.feature_importances_, index=X_train.
  ↪columns)
important_features = important_features[important_features > 0].
  ↪sort_values(ascending=False)
print(important_features, n_leaves, mse_test)
```

```
CAtBat          0.614032
CRuns           0.105176
Walks           0.082480
Hits            0.039794
CRBI            0.030783
AtBat           0.022928
CHits           0.019713
PutOuts         0.018320
CHmRun          0.017818
RBI             0.014231
Assists         0.013396
Errors          0.005891
HmRun           0.005503
NewLeague_N     0.003566
CWalks          0.003453
Years           0.002096
Runs            0.000610
Division_W      0.000177
League_N        0.000033
dtype: float64 183 0.39152675958676264
```

[6]:
```
# Task 3: number of terminal nodes
n_leaves = tree_model.get_n_leaves()
print(n_leaves)
```

```
183
```

4. Compute the mean squared prediction error for the test data.

[27]:
```
# Task 4: Test MSE

from sklearn.metrics import mean_squared_error
y_pred = tree_model.predict(X_test)
mse_test = mean_squared_error(y_test, y_pred)
print(mse_test)
```

```
0.39152675958676264
```

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 that minimize the mean squared cross validation error.

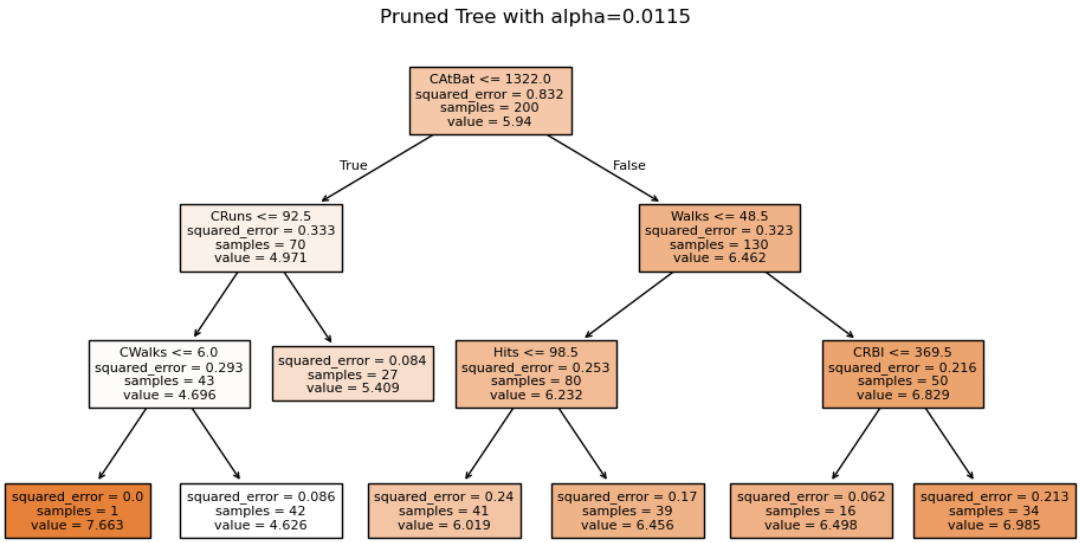
[21]:
```
from sklearn.model_selection import cross_val_score

# Task 5: Cross-validation with cost-complexity pruning
path = tree_model.cost_complexity_pruning_path(X_train, y_train)
ccp_alphas = path.ccp_alphas[:-1]  # skip the maximum alpha
trees = []
cv_results = []

for ccp_alpha in ccp_alphas:
    tree = DecisionTreeRegressor(random_state=2, ccp_alpha=ccp_alpha)
    scores = cross_val_score(tree, X_train, y_train,␣
 ↪scoring='neg_mean_squared_error', cv=5)
    cv_results.append(-scores.mean())
    tree.fit(X_train, y_train)
    trees.append(tree)

# Ausgabe der besten Ergebnisse
best_idx = np.argmin(cv_results)
print(f"Min CV MSE: {cv_results[best_idx]:.4f} at alpha = {ccp_alphas[best_idx]:
 ↪.5f}")
```

```
Min CV MSE: 0.2644 at alpha = 0.01155
```

6. Use the pruning parameter from the previous task to prune the tree. Plot the tree and report the most important variables.

[11]:
```
# Task 6: Plot pruned tree
fig, ax = plt.subplots(figsize=(12, 6))
plot_tree(optimal_tree, feature_names=X_train.columns, filled=True, fontsize=8)
```

```
plt.title(f"Pruned Tree with alpha={optimal_alpha:.4f}")
plt.show()
```



7. Compute the test mean squared prediction error for pruned tree and compare to the results from Task 4.

[29]:
```
# Task 7: Compute MSE on test set with pruned tree
from sklearn.metrics import mean_squared_error

# Pruned MSE
y_pred_pruned = optimal_tree.predict(X_test)
mse_pruned = mean_squared_error(y_test, y_pred_pruned)

print(f"Test-MSE (unpruned): {mse_test:.4f}")
print(f"Test-MSE (pruned with   = {optimal_alpha:.5f}): {mse_pruned:.4f}")
```

```
Test-MSE (unpruned): 0.3915
Test-MSE (pruned with   = 0.01155): 0.3079
```

8. Use random forest to improve the predictions. Fit 500 trees using $m = \sqrt{(p)}$ (round to the nearest integer).

[32]:
```
from sklearn.ensemble import RandomForestRegressor

# Random Forest with m = sqrt(p)
p = X_train.shape[1]
m_try = int(np.round(np.sqrt(p)))
```

```
rf_model = RandomForestRegressor(n_estimators=500, max_features=m_try,␣
 ↪oob_score=True, random_state=1)
rf_model.fit(X_train, y_train)

# Evaluate
y_pred = rf_model.predict(X_test)
mse_rf = mean_squared_error(y_test, y_pred)
print(f"Random Forest MSE (500 trees): {mse_rf:.4f}")
```

```
Random Forest MSE (500 trees): 0.2122
```

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.

[34]:
```
# Task 9 - Find best number of trees via OOB error
oob_errors = []
tree_counts = list(range(10, 501, 10))
for n in tree_counts:
    model = RandomForestRegressor(n_estimators=n, max_features=m_try,␣
 ↪oob_score=True, random_state=1)
    model.fit(X_train, y_train)
    oob_errors.append(1 - model.oob_score_)

best_n = tree_counts[np.argmin(oob_errors)]
print(f"Lowest OOB Error at {best_n} trees: {min(oob_errors):.4f}")

# Plot OOB error
plt.plot(tree_counts, oob_errors, marker='o')
plt.axvline(x=best_n, color='red', linestyle='--', label=f'Optimal trees:␣
 ↪{best_n}')
plt.xlabel("Number of Trees")
plt.ylabel("OOB Error")
plt.title("Out-of-Bag Error vs Number of Trees")
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()
```

```
/usr/lib/python3.13/site-packages/sklearn/ensemble/_forest.py:612: UserWarning:
Some inputs do not have OOB scores. This probably means too few trees were used
to compute any reliable OOB estimates.
  warn(
```

```
Lowest OOB Error at 490 trees: 0.2531
```

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.

[37]:
```
print(f"Min CV MSE: {cv_results[best_idx]:.4f}")
print(f"Lowest OOB Error {min(oob_errors):.4f} at {best_n} Trees")
```

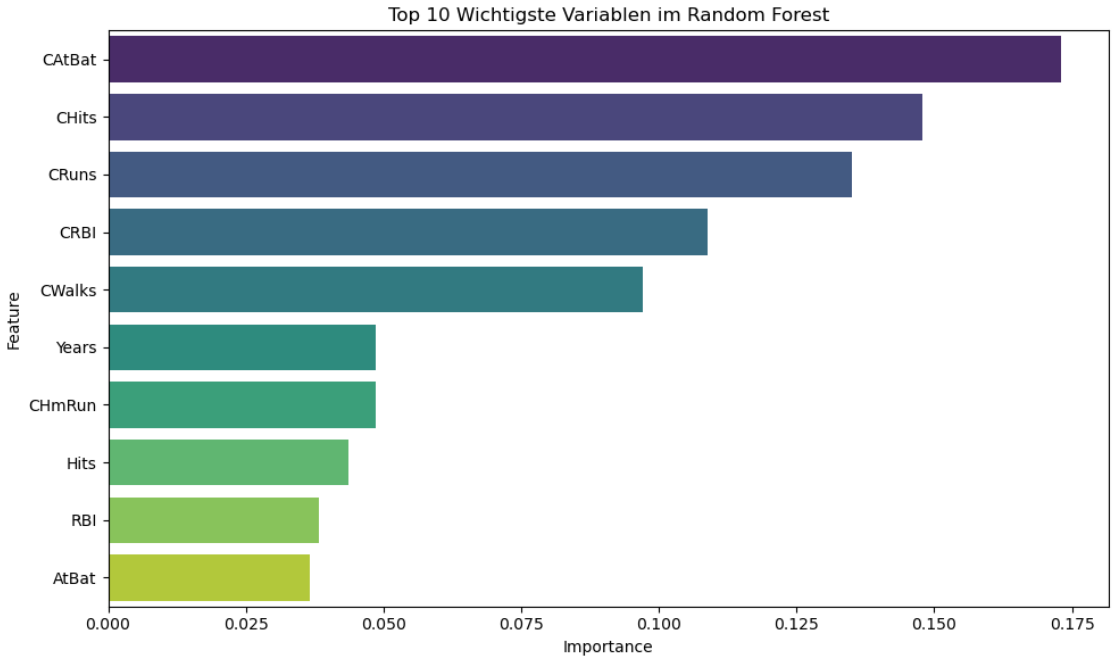
```
Min CV MSE: 0.2644
Lowest OOB Error 0.2531 at 490 Trees
```

11. Which are the most important variables used in the random forest?

[39]:
```
# Feature Importances
importances = rf_model.feature_importances_
feature_names = X_train.columns

# DataFrame overview
importance_df = pd.DataFrame({
    'Feature': feature_names,
    'Importance': importances
}).sort_values(by='Importance', ascending=False)

print(importance_df.head(10))  # Top 10 Features
```

```
    Feature  Importance
7    CAtBat    0.173027
8     CHits    0.147864
10    CRuns    0.134998
11     CRBI    0.108832
12   CWalks    0.097070
6     Years    0.048685
9    CHmRun    0.048505
1      Hits    0.043583
4       RBI    0.038223
0     AtBat    0.036723
```

[42]:
```
# Optional / Extra: plot
import seaborn as sns
plt.figure(figsize=(10, 6))
sns.barplot(data=importance_df.head(10), x='Importance', y='Feature',␣
 ↪hue='Feature', palette='viridis')
plt.title("Top 10 Wichtigste Variablen im Random Forest")
plt.tight_layout()
plt.show()
```



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.

[44]:
```
n_features = X_train.shape[1]
mtry_range = range(1, n_features + 1)

test_mse_grid = []
oob_error_grid = []

# Grid-search over mtry (max_features)
for m in mtry_range:
    rf_model = RandomForestRegressor(n_estimators=500,
                                     max_features=m,
                                     oob_score=True,
                                     random_state=1,
                                     bootstrap=True)
    rf_model.fit(X_train, y_train)

    # Test-MSE
    y_pred_test = rf_model.predict(X_test)
    mse_test = mean_squared_error(y_test, y_pred_test)
    test_mse_grid.append(mse_test)

    # OOB-error
    oob_error = 1 - rf_model.oob_score_
    oob_error_grid.append(oob_error)

# results as DataFrame
results_df = pd.DataFrame({
    'mtry': list(mtry_range),
    'Test_MSE': test_mse_grid,
    'OOB_Error': oob_error_grid
})

# Finding best mtry-values
best_mtry_test = results_df.loc[results_df['Test_MSE'].idxmin()]
best_mtry_oob = results_df.loc[results_df['OOB_Error'].idxmin()]

print("Best mtry based on Test-MSE:")
print(best_mtry_test)

print("\nBest mtry based on OOB-Error:")
print(best_mtry_oob)
```

```
Best mtry based on Test-MSE:
mtry         3.000000
Test_MSE     0.207397
OOB_Error    0.249531
Name: 2, dtype: float64

Best mtry based on OOB-Error:
```

```
mtry          2.000000
Test_MSE      0.212309
OOB_Error     0.248776
Name: 1, dtype: float64
```

13. For the best model, compute the test errors and compare them to the best pruned model from Task 7.

[45]:
```python
# Optimal alpha-value
optimal_tree = DecisionTreeRegressor(random_state=1, ccp_alpha=optimal_alpha)
optimal_tree.fit(X_train, y_train)

# MSE for Pruned Tree
y_pred_pruned = optimal_tree.predict(X_test)
mse_pruned = mean_squared_error(y_test, y_pred_pruned)
print(f"Test-MSE for best Pruned Tree: {mse_pruned:.4f}")
```

```
Test-MSE for best Pruned Tree: 0.3079
```

[46]:
```python
# Compare with #12
print(f"\nTest-MSE best Random Forest Model: {best_mtry_test['Test_MSE']:.4f}")
print(f"Test-MSE best Pruned Tree Model  : {mse_pruned:.4f}")
```

```
Test-MSE best Random Forest Model: 0.2074
Test-MSE best Pruned Tree Model  : 0.3079
```

14. What is the OOB error obtained from bagging (you can infer the answer from the previous task).

[ ]:
```
```