Building Decision Trees Using Scikit-learn: Takeaways

Syntax

- Loading a partial dataset (i.e., selected columns) with a specified delimiter:

```
df = pd.read_csv("dataset.csv",
             usecols = ["selected_column_1",
                        "selected_column_2",
                        "selected_column_3",
                        "target_col"],
             sep = ";")
```

- Rearranging the columns of a dataset:

```
df[["selected_column_3",
    "selected_column_1",
    "selected_column_2",
    "target_col"]]
```

- Making an independent copy of a dataset:

```
new_df = df.copy()
```

Preprocessing

- Transforming ordinal columns into numericals with `pandas` :

```
df.loc[df["column_1"] == "Ordinal Value 1", "column_1"] = 0
df.loc[df["column_1"] == "Ordinal Value 2", "column_1"] = 1
df.loc[df["column_1"] == "Ordinal Value 3", "column_1"] = 2
```

- Transforming ordinal columns into numericals with `scikit-learn` :

```
from sklearn.preprocessing import OrdinalEncoder
hierarchy = [["Low",
              "Medium",
              "High"]] # Ensure you use double brackets
df["col"] = OrdinalEncoder(categories = hierarchy).fit_transform(df[["col"]])
```

- Transforming binary columns into numericals with `pandas` :

```
df["weather"].replace( {"rainy": 0, "sunny": 1}, inplace = True )
df.rename(columns = {"weather": "weather_sunny"}, inplace = True)
```

- Transforming multi-categorical columns into numericals with `pandas` :

```
dummies_single_column = pd.get_dummies(df["col1"],
                                       prefix = "col1_name")
dummies_many_columns = pd.get_dummies(df[["col1", "col2"]],
                                      prefix = ["col1_prefix", "col2_prefix"])
df = pd.concat([df, dummies_single_column],
               axis = 1) # Or [df, dummies_many_columns]
df.drop("col1", axis = 1, inplace = True) # Or df.drop(["col1", "col2]...
```

- Transforming multi-categorical columns into numericals with `scikit-learn`:

```
from sklearn.compose import make_column_transformer
from sklearn.preprocessing import OneHotEncoder
col_trans = make_column_transformer(
    (OneHotEncoder(), ["selected_column"]),
    remainder = "passthrough",
    verbose_feature_names_out = False)
onehot_df = col_trans.fit_transform(df)
df = pd.DataFrame(onehot_df,
                  columns = col_trans.get_feature_names_out())
```

- Creating target column for classification tree with the `mask` method:

```
df["target_col_classific"] = pd.Series(dtype = "object")
df["target_col_classific"].mask(df["temperature"] <= 4,
                                "Cold",
                                inplace = True)
df["target_col_classific"].mask(
    (df["temperature"] >= 17) & (df["temperature"] <= 24),
    "Warm",
    inplace = True) # More than one condition
```

Machine Learning

- Grouping feature columns into variable `X`:

```
X = df.drop(["target_column"], axis = 1)
```

- Isolating target column into variable `y`:

```
y = df["target_column"]
```

- Importing regression tree library from `scikit-learn`:

```
from sklearn.tree import DecisionTreeRegressor
regression_tree = DecisionTreeRegressor(
    criterion = "squared_error", # Or "absolute_error"
    max_depth = 3,
    random_state = 24)
```

- Importing classification tree library from `scikit-learn`:

```
from sklearn.tree import DecisionTreeClassifier
classification_tree = DecisionTreeClassifier(
                      criterion = "gini", # Or "entropy"
                      max_depth = 3,
                      random_state = 24)
```

- Creating training and test subsets with `train_test_split`:

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(
                                     X, y,
                                     test_size = 0.3,
                                     shuffle = True,
                                     random_state = 24)
```

- Fitting model:

```
decision_tree.fit(X_train, y_train)
```

- Using decision tree to make predictions on test subset:

```
y_pred = decision_tree.predict(X_test)
```

- Using decision tree to make predictions on a new observation:

```
new_observation = pd.DataFrame(
                 {"column1_input": [0.0],
                  "column2_input": [0.0],
                  ...
                  "column_n_input": [0.0]
                  })
decision_tree.predict(new_observation)
```

- Plotting the decision tree with `plot_tree`:

```
import matplotlib.pyplot as plt
from sklearn.tree import plot_tree
plt.rcParams["figure.figsize"] = [24.0, 8.0]
_ = plot_tree(decision_tree_instance,
              feature_names = X.columns,
              class_names = decision_tree_instance.classes_ # Class names. Exclusive for
Classification Trees.
              filled = True, # Nodes filled with colors
              proportion = True, # Or False
              precision = 0, # Number of decimals
              rounded = True, # Rounded nodes
              fontsize = 11)
plt.show()
```

- Exporting the decision tree with `export_text`:

```
from sklearn.tree import export_text
exported_tree = export_text(
                   decision_tree_instance,
                   feature_names = list(X.columns)) # Always ensure it's converted to a
list.
print(exported_tree)
```

Concepts

- `scikit-learn` is a powerful library that allows us to build decision trees in a matter of minutes, saving us from the thousands of calculations involved in finding optimal thresholds.
- `scikit-learn` only supports numerical feature columns, so we need to preprocess these columns accordingly.

- When building a classification tree, the categories from the target column don't need to be transformed.
- `scikit-learn` is a vast library; it includes several tools for data preprocessing, plotting, generating reports, etc.
- We use label encoding on columns with ordinal values, which are categorical values that have a hierarchy or logical progression (e.g., "Bad"|"Regular"|"Good"|"Very Good"|"Excellent", "Cold"|"Warm"|"Hot", "Mild"|"Moderate"|"Serious"|"Fatal").
- We use either pandas `pd.dummies` or scikit-learn's `OneHotEncoder` when dealing with categorical columns with multiple values. This creates a column for every possible value, and the one that belongs to that observation will be marked with a 1 and the rest with 0.
- When building a classification tree, if we want to transform continuous values in our target column to categorical values, we can use the pandas `mask` method, which assigns a label based on a condition.
- Once the dataset is properly processed, we store the feature columns in the variable `X` . This variable name is used by convention. Same for the `y` variable that exclusively stores the target column.
- By using `train_test_split` , we divide the dataset into a training subset and a test subset. This allows us to evaluate the precision and effectiveness of our tree at making predictions.
- We select the criterion we want to use when instantiating the decision tree with the `criterion` parameter.
- We fit our trees to the training data (`X_train` and `y_train`) and then make predictions on the `X_test` subset. We will then compare those predictions with the `y_test` column to evaluate the tree.
- When plotting the tree, we can assign the output to a variable like `_` to avoid the verbose array before every tree is plotted.
- The `export_text` tool from `scikit-learn` is used to create a text report of the rules of a decision tree.

Resources

- Ordinal data
- Nominal/Categorical Data
- pandas:
    - replace
    - rename
    - get_dummies
    - mask
- scikit-learn:
    - Decision Tree page
    - Encoding Categorical Features
    - OrdinalEncoder

    - OneHotEncoder
    - make_column_transformer
    - train_test_split
    - Decision Tree Regressor
    - Decision Tree Classifier
    - plot_tree
    - export_text
- Graphviz:
    - Official site
    - export_graphviz