# 043-decision-tree

May 31, 2022

4.3. Predicting Damage with Decision Trees

[1]:
```
import sqlite3
import warnings

import matplotlib.pyplot as plt
import pandas as pd
from category_encoders import OrdinalEncoder
from IPython.display import VimeoVideo
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
from sklearn.pipeline import Pipeline, make_pipeline
from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn.utils.validation import check_is_fitted

warnings.simplefilter(action="ignore", category=FutureWarning)
```

[2]:
```
VimeoVideo("665414130", h="71649d291e", width=600)
```

[2]:
```
<IPython.lib.display.VimeoVideo at 0x7f844835d0d0>
```

# 1 Prepare Data

## 1.1 Import

[3]:
```
def wrangle(db_path):
    # Connect to database
    conn = sqlite3.connect(db_path)

    # Construct query
    query = """
        SELECT distinct(i.building_id) AS b_id,
           s.*,
           d.damage_grade
        FROM id_map AS i
        JOIN building_structure AS s ON i.building_id = s.building_id
        JOIN building_damage AS d ON i.building_id = d.building_id
```

```
        WHERE district_id = 4
    """

    # Read query results into DataFrame
    df = pd.read_sql(query, conn, index_col="b_id")

    # Identify leaky columns
    drop_cols = [col for col in df.columns if "post_eq" in col]

    # Add high-cardinality / redundant column
    drop_cols.append("building_id")

    # Create binary target column
    df["damage_grade"] = df["damage_grade"].str[-1].astype(int)
    df["severe_damage"] = (df["damage_grade"] > 3).astype(int)

    # Drop old target
    drop_cols.append("damage_grade")

    # Drop multicollinearity column
    drop_cols.append("count_floors_pre_eq")

    # Drop columns
    df.drop(columns=drop_cols, inplace=True)

    return df
```

**Task 4.3.1:** Use the `wrangle` function above to import your data set into the DataFrame `df`. The path to the SQLite database is `"/home/jovyan/nepal.sqlite"`

- Read SQL query into a DataFrame using pandas.
- Write a function in Python.

[4]:
```
df = wrangle("/home/jovyan/nepal.sqlite")
df.head()
```

[4]:
```
        age_building  plinth_area_sq_ft  height_ft_pre_eq  \
b_id
164002            20                560                18
164081            21                200                12
164089            18                315                20
164098            45                290                13
164103            21                230                13

       land_surface_condition         foundation_type  \
b_id
164002                   Flat  Mud mortar-Stone/Brick
164081                   Flat  Mud mortar-Stone/Brick
```

```
164089                        Flat   Mud mortar-Stone/Brick
164098                        Flat   Mud mortar-Stone/Brick
164103                        Flat   Mud mortar-Stone/Brick

                       roof_type ground_floor_type   other_floor_type  \
b_id
164002  Bamboo/Timber-Light roof               Mud  TImber/Bamboo-Mud
164081  Bamboo/Timber-Light roof               Mud  TImber/Bamboo-Mud
164089  Bamboo/Timber-Light roof               Mud  TImber/Bamboo-Mud
164098  Bamboo/Timber-Light roof               Mud  TImber/Bamboo-Mud
164103  Bamboo/Timber-Light roof               Mud  TImber/Bamboo-Mud

            position plan_configuration     superstructure  severe_damage
b_id
164002  Not attached        Rectangular  Stone, mud mortar              0
164081  Not attached        Rectangular  Stone, mud mortar              0
164089  Not attached        Rectangular  Stone, mud mortar              0
164098  Not attached        Rectangular  Stone, mud mortar              0
164103  Not attached        Rectangular  Stone, mud mortar              0
```

[5]:
```
# Check your work
assert df.shape[0] == 70836, f"`df` should have 70,836 rows, not {df.shape[0]}."
assert df.shape[1] == 12, f"`df` should have 12 columns, not {df.shape[1]}."
```

### 1.2 Split

**Task 4.3.2:** Create your feature matrix `X` and target vector `y`. Your target is `"severe_damage"`.

- What's a feature matrix?
- What's a target vector?
- Subset a DataFrame by selecting one or more columns in pandas.
- Select a Series from a DataFrame in pandas.

[6]:
```
target = "severe_damage"
X = df.drop(columns=target)
y = df[target]
```

[7]:
```
# Check your work
assert X.shape == (70836, 11), f"The shape of `X` should be (70836, 11), not {X.
↪shape}."
assert y.shape == (70836,), f"The shape of `y` should be (70836,), not {y.
↪shape}."
```

[8]:
```
VimeoVideo("665415006", h="ecb1b87861", width=600)
```

[8]:
```
&lt;IPython.lib.display.VimeoVideo at 0x7f836ce4b1f0&gt;
```

**Task 4.3.3:** Divide your data (`X` and `y`) into training and test sets using a randomized train-test split. Your test set should be 20% of your total data. And don't forget to set a `random_state` for reproducibility.

- Perform a randomized train-test split using scikit-learn.

[9]:
```
X_train, X_test, y_train, y_test = train_test_split(
    X,y, test_size=0.2, random_state=42
)
```

[10]:
```
# Check your work
assert X_train.shape == (
    56668,
    11,
), f"The shape of `X_train` should be (56668, 11), not {X_train.shape}."
assert y_train.shape == (
    56668,
), f"The shape of `y_train` should be (56668,), not {y_train.shape}."
assert X_test.shape == (
    14168,
    11,
), f"The shape of `X_test` should be (14168, 11), not {X_test.shape}."
assert y_test.shape == (
    14168,
), f"The shape of `y_test` should be (14168,), not {y_test.shape}."
```

**Task 4.3.4:** Divide your training data (`X_train` and `y_train`) into training and validation sets using a randomized train-test split. Your validation data should be 20% of the remaining data. Don't forget to set a `random_state`.

- What's a validation set?
- Perform a randomized train-test split using scikit-learn.

[11]:
```
X_train, X_val, y_train, y_val = train_test_split(
    X_train,y_train, test_size=0.2, random_state=42
)
```

[12]:
```
# Check your work
assert X_train.shape == (
    45334,
    11,
), f"The shape of `X_train` should be (45334, 11), not {X_train.shape}."
assert y_train.shape == (
    45334,
), f"The shape of `y_train` should be (45334,), not {y_train.shape}."
assert X_val.shape == (
    11334,
    11,
), f"The shape of `X_val` should be (11334, 11), not {X_val.shape}."
assert y_val.shape == (
```

```
    11334,
), f"The shape of `y_val` should be (11334,), not {y_val.shape}."
```

# 2 Build Model

## 2.1 Baseline

**Task 4.3.5:** Calculate the baseline accuracy score for your model.

- What's accuracy score?
- Aggregate data in a Series using `value_counts` in pandas.

[13]:
```
acc_baseline = y_train.value_counts(normalize=True).max()
print("Baseline Accuracy:", round(acc_baseline, 2))
```

```
Baseline Accuracy: 0.64
```

## 2.2 Iterate

[14]:
```
VimeoVideo("665415061", h="6250826047", width=600)
```

[14]:
```
<IPython.lib.display.VimeoVideo at 0x7f8368a36310>
```

[15]:
```
VimeoVideo("665415109", h="b3bb82651d", width=600)
```

[15]:
```
<IPython.lib.display.VimeoVideo at 0x7f8368a36eb0>
```

**Task 4.3.6:** Create a pipeline named `model` that contains a `OrdinalEncoder` transformer and a `DecisionTreeClassifier` predictor. (Be sure to set a `random_state` for your predictor.) Then fit your model to the training data.

- What's a decision tree?
- What's ordinal encoding?
- Create a pipeline in scikit-learn.
- Fit a model to training data in scikit-learn.

[16]:
```
# Build Model
model = make_pipeline(
    OrdinalEncoder(),
    DecisionTreeClassifier(max_depth=6,random_state=42)
)
# Fit model to training data
model.fit(X_train, y_train)
```

[16]:
```
Pipeline(steps=[('ordinalencoder',
                 OrdinalEncoder(cols=['land_surface_condition',
                                      'foundation_type', 'roof_type',
                                      'ground_floor_type', 'other_floor_type',
                                      'position', 'plan_configuration',
```

```
                                        'superstructure'],
                                  mapping=[{'col': 'land_surface_condition',
                                            'data_type': dtype('O'),
                                            'mapping': Flat             1
Moderate slope    2
Steep slope       3
NaN              -2
dtype: int64},
                                           {'col': 'foundation_type',
                                            'dat…
Others                                9
Building with Central Courtyard      10
NaN                                  -2
dtype: int64},
                                           {'col': 'superstructure',
                                            'data_type': dtype('O'),
                                            'mapping': Stone, mud mortar        1
Stone                     2
RC, engineered            3
Brick, cement mortar      4
Adobe/mud                 5
Timber                    6
RC, non-engineered        7
Brick, mud mortar         8
Stone, cement mortar      9
Bamboo                   10
Other                    11
NaN                      -2
dtype: int64}])),
                ('decisiontreeclassifier',
                 DecisionTreeClassifier(max_depth=6, random_state=42))])
```

[17]:

```python
# Check your work
assert isinstance(
    model, Pipeline
), f"`model` should be a Pipeline, not type {type(model)}."
assert isinstance(
    model[0], OrdinalEncoder
), f"The first step in your Pipeline should be an OrdinalEncoder, not type␣
↪{type(model[0])}."
assert isinstance(
    model[-1], DecisionTreeClassifier
), f"The last step in your Pipeline should be an DecisionTreeClassifier, not␣
↪type {type(model[-1])}."
check_is_fitted(model)
```

[18]:

```python
VimeoVideo("665415153", h="f0ec320955", width=600)
```

[18]: <IPython.lib.display.VimeoVideo at 0x7f8368a36ca0>

**Task 4.3.7:** Calculate the training and validation accuracy scores for your models.

- Calculate the accuracy score for a model in scikit-learn.
- Generate predictions using a trained model in scikit-learn.

[19]:
```
acc_train = accuracy_score(y_train, model.predict(X_train))
acc_val = model.score(X_val,y_val)

print("Training Accuracy:", round(acc_train, 2))
print("Validation Accuracy:", round(acc_val, 2))
```

```
Training Accuracy: 0.72
Validation Accuracy: 0.72
```

[20]:
```
VimeoVideo("665415169", h="44702fc696", width=600)
```

[20]: <IPython.lib.display.VimeoVideo at 0x7f8368a36910>

**Task 4.3.8:** Use the `get_depth` method on the `DecisionTreeClassifier` in your model to see how deep your tree grew during training.

- Access an object in a pipeline in scikit-learn.

[21]:
```
tree_depth = model.named_steps["decisiontreeclassifier"].get_depth()
print("Tree Depth:", tree_depth)
```

```
Tree Depth: 6
```

[22]:
```
VimeoVideo("665415186", h="c4ce187170", width=600)
```

[22]: <IPython.lib.display.VimeoVideo at 0x7f836cd8abe0>

**Task 4.3.9:** Create a range of possible values for `max_depth` hyperparameter of your model's `DecisionTreeClassifier`. `depth_hyperparams` should range from `1` to `50` by steps of `2`.

- What's an iterator?
- Create a range in Python.

[23]:
```
depth_hyperparams = range(1,50,2)
```

[24]:
```
# Check your work
assert (
    len(list(depth_hyperparams)) == 25
), f"`depth_hyperparams` should contain 25 items, not␣
↪{len(list(depth_hyperparams))}."
assert (
    list(depth_hyperparams)[0] == 1
), f"`depth_hyperparams` should begin at 1, not {list(depth_hyperparams)[0]}."
```

```
assert (
    list(depth_hyperparams)[-1] == 49
), f"`depth_hyperparams` should end at 49, not {list(depth_hyperparams)[-1]}."
```

[25]:
```
VimeoVideo("665415198", h="b4b85c3308", width=600)
```

[25]: <IPython.lib.display.VimeoVideo at 0x7f836cd943d0>

**Task 4.3.10:** Complete the code below so that it trains a model for every `max_depth` in `depth_hyperparams`. Every time a new model is trained, the code should also calculate the training and validation accuracy scores and append them to the `training_acc` and `validation_acc` lists, respectively.

- Append an item to a list in Python.
- Create a pipeline in scikit-learn.
- Fit a model to training data in scikit-learn.
- Write a for loop in Python.

[33]:
```
# Create empty lists for training and validation accuracy scores
training_acc = []
validation_acc = []

for d in depth_hyperparams:
    # Create model with `max_depth` of `d`
    test_model = make_pipeline(
        OrdinalEncoder(),
        DecisionTreeClassifier(max_depth=d, random_state=42)
    )
    # Fit model to training data
    test_model.fit(X_train, y_train)
    # Calculate training accuracy score and append to `training_acc`
    training_acc.append(test_model.score(X_train,y_train))
    # Calculate validation accuracy score and append to `training_acc`
    validation_acc.append(test_model.score(X_val,y_val))

print("Training Accuracy Scores:", training_acc[:3])
print("Validation Accuracy Scores:", validation_acc[:3])
```

```
Training Accuracy Scores: [0.7071072484228174, 0.7117395332421582,
0.7162394670666608]
Validation Accuracy Scores: [0.7088406564319746, 0.7132521616375508,
0.7166049055937886]
```

[27]:
```
# Check your work
assert (
    len(training_acc) == 25
), f"`training_acc` should contain 25 items, not {len(training_acc)}."
assert (
```

```
    len(validation_acc) == 25
), f"`validation_acc` should contain 25 items, not {len(validation_acc)}."
```

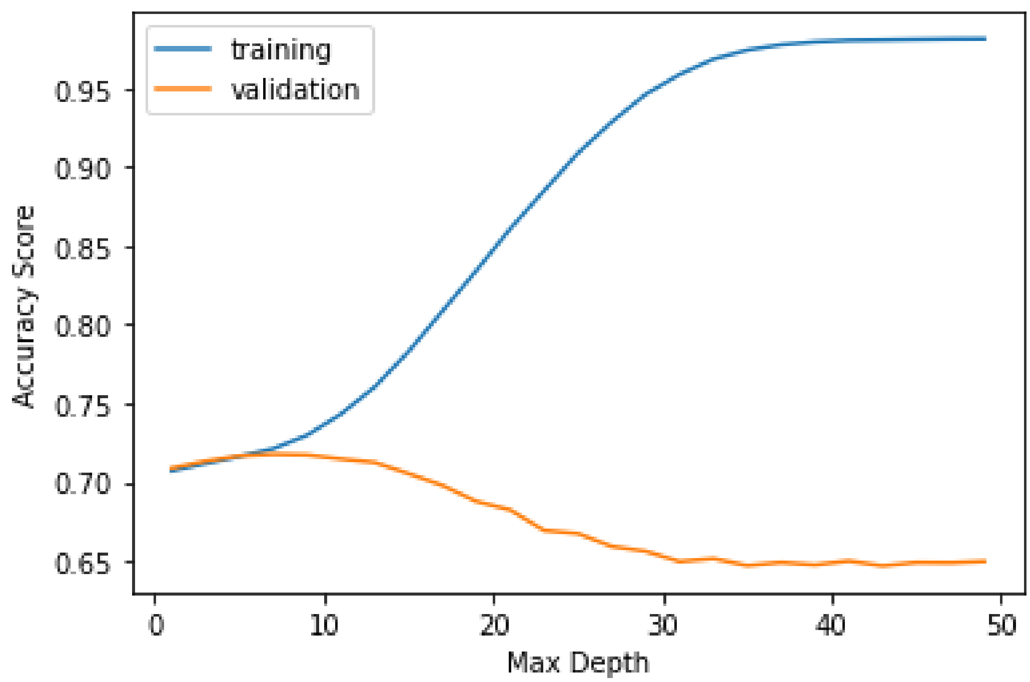
[28]:
```
VimeoVideo("665415236", h="51d4be13fa", width=600)
```

[28]: <IPython.lib.display.VimeoVideo at 0x7f836cd94400>

**Task 4.3.11:** Create a visualization with two lines. The first line should plot the `training_acc` values as a function of `depth_hyperparams`, and the second should plot `validation_acc` as a function of `depth_hyperparams`. You x-axis should be labeled `"Max Depth"`, and the y-axis `"Accuracy Score"`. Also include a legend so that your audience can distinguish between the two lines.

- What's a line plot?
- Create a line plot in Matplotlib.

[29]:
```
# Plot `depth_hyperparams`, `training_acc`
plt.plot(depth_hyperparams, training_acc, label="training")
plt.plot(depth_hyperparams, validation_acc, label="validation")
plt.xlabel("Max Depth")
plt.ylabel("Accuracy Score")
plt.legend();
```



### 2.3 Evaluate

[30]:
```python
VimeoVideo("665415255", h="573e9cfd74", width=600)
```

[30]: <IPython.lib.display.VimeoVideo at 0x7f836aca34c0>

**Task 4.3.12:** Based on your visualization, choose the `max_depth` value that leads to the best validation accuracy score. Then retrain your original model with that `max_depth` value. Lastly, check how your tuned model performs on your test set by calculating the test accuracy score below. Were you able to resolve the overfitting problem with this new `max_depth`?

- Calculate the accuracy score for a model in scikit-learn.
- Generate predictions using a trained model in scikit-learn.

[31]:
```python
test_acc = model.score(X_test,y_test)
print("Test Accuracy:", round(test_acc, 2))
```

```
Test Accuracy: 0.72
```

## 3 Communicate

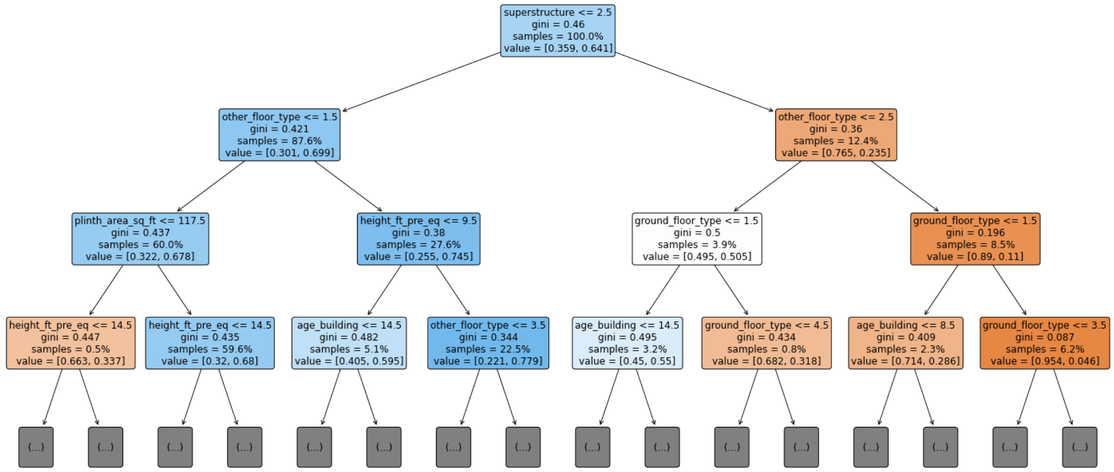
[32]:
```python
VimeoVideo("665415275", h="880366a826", width=600)
```

[32]: <IPython.lib.display.VimeoVideo at 0x7f8369237940>

**Task 4.3.13:** Complete the code below to use the `plot_tree` function from scikit-learn to visualize the decision logic of your model.

- Plot a decision tree using scikit-learn.

[34]:
```python
# Create larger figure
fig, ax = plt.subplots(figsize=(25, 12))
# Plot tree
plot_tree(
    decision_tree=model.named_steps["decisiontreeclassifier"],
    feature_names=X_train.columns,
    filled=True,  # Color leaf with class
    rounded=True,  # Round leaf edges
    proportion=True,  # Display proportion of classes in leaf
    max_depth=3,  # Only display first 3 levels
    fontsize=12,  # Enlarge font
    ax=ax,  # Place in figure axis
);
```




[35]:
```
VimeoVideo("665415304", h="c7eeac08c9", width=600)
```

[35]:
```
<IPython.lib.display.VimeoVideo at 0x7f8368a80160>
```

**Task 4.3.14:** Assign the feature names and importances of your model to the variables below. For the `features`, you can get them from the column names in your training set. For the `importances`, you access the `feature_importances_` attribute of your model's `DecisionTreeClassifier`.

- Access an object in a pipeline in scikit-learn.

[36]:
```
features = X_train.columns
importances = model.named_steps["decisiontreeclassifier"].feature_importances_

print("Features:", features[:3])
print("Importances:", importances[:3])
```

```
Features: Index(['age_building', 'plinth_area_sq_ft', 'height_ft_pre_eq'],
dtype='object')
Importances: [0.03515085 0.04618639 0.08839161]
```

[37]:
```
# Check your work
assert len(features) == 11, f"`features` should contain 11 items, not␣
↪{len(features)}."
assert (
    len(importances) == 11
), f"`importances` should contain 11 items, not {len(importances)}."
```

**Task 4.3.15:** Create a pandas Series named `feat_imp`, where the index is `features` and the values are your `importances`. The Series should be sorted from smallest to largest importance.

- Create a Series in pandas.

[40]:
```
feat_imp = pd.Series(importances, index=features).sort_values()
feat_imp.head()
```

[40]:
```
position                  0.000644
plan_configuration        0.004847
foundation_type           0.005206
roof_type                 0.007620
land_surface_condition    0.020759
dtype: float64
```

[ ]:
```
# Check your work
assert isinstance(
    feat_imp, pd.Series
), f"`feat_imp` should be a Series, not {type(feat_imp)}."
assert feat_imp.shape == (
    11,
), f"`feat_imp` should have shape (11,), not {feat_imp.shape}."
```

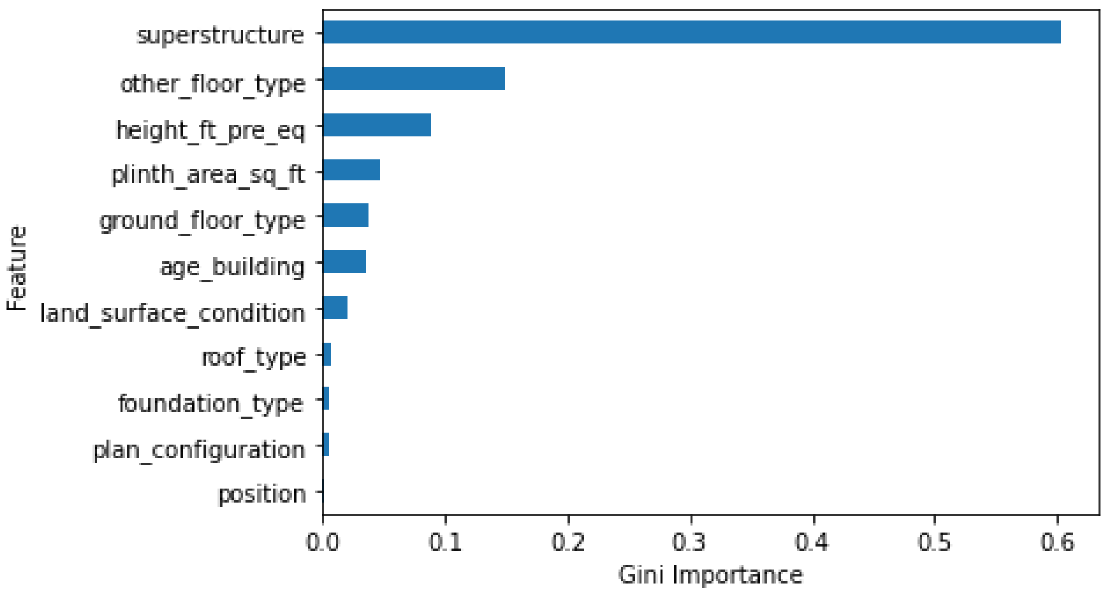
[39]:
```
VimeoVideo("665415316", h="0dd9004477", width=600)
```

[39]:
```
<IPython.lib.display.VimeoVideo at 0x7f8368bf0100>
```

**Task 4.3.16:** Create a horizontal bar chart with all the features in `feat_imp`. Be sure to label your x-axis `"Gini Importance"`.

- What's a bar chart?
- Create a bar chart using pandas.

[41]:
```
# Create horizontal bar chart
feat_imp.plot(kind="barh")
plt.xlabel("Gini Importance")
plt.ylabel("Feature");
```


superstructure
other_floor_type
height_ft_pre_eq
plinth_area_sq_ft
ground_floor_type
age_building
land_surface_condition
roof_type
foundation_type
plan_configuration
position
Feature
0.0
0.1
0.2
0.3
0.4
0.5
0.6
Gini Importance