3_Decision trees and model architecture

October 6, 2021

1 Exercise: Decision trees and model architecture

Our goal in this exercise is to use a decision tree classifier to predict whether an individual crime will be resolved, based on simple information such as where it took place and what kind of crime it was.

1.1 Data visualization

As usual, let's begin by loading in and having a look at our data:

[ ]:
```
import pandas

# Import the data from the .csv file
dataset =pandas.read_csv('san_fran_crime.csv', delimiter="\t")

#Let's have a look at the data and the relationship we are going to model
dataset.head()
```

[ ]:
```
       Category  DayOfWeek  PdDistrict  Resolution           X          Y  \
0   WEAPON LAWS          5    SOUTHERN        True -122.403405  37.775421
1   WEAPON LAWS          5    SOUTHERN        True -122.403405  37.775421
2      WARRANTS          1     BAYVIEW        True -122.388856  37.729981
3  NON-CRIMINAL          2  TENDERLOIN       False -122.412971  37.785788
4  NON-CRIMINAL          5     MISSION       False -122.419672  37.765050

   day_of_year  time_in_hours
0           29      11.000000
1           29      11.000000
2          116      14.983333
3            5      23.833333
4            1       0.500000
```

[ ]:
```
dataset.shape
```

[ ]:
```
(150431, 8)
```

[ ]:
```
dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150431 entries, 0 to 150430
Data columns (total 8 columns):
 #   Column         Non-Null Count   Dtype
---  ------         --------------   -----
 0   Category       150431 non-null  object
 1   DayOfWeek      150431 non-null  int64
 2   PdDistrict     150430 non-null  object
 3   Resolution     150431 non-null  bool
 4   X              150431 non-null  float64
 5   Y              150431 non-null  float64
 6   day_of_year    150431 non-null  int64
 7   time_in_hours  150431 non-null  float64
dtypes: bool(1), float64(3), int64(2), object(2)
memory usage: 8.2+ MB
```

[ ]:
```
print(f' Category feature unique value are {dataset.Category.nunique()} in␣
 ↪number')
print(f'PdDistrict unique features are {dataset.PdDistrict.nunique()} in␣
 ↪number')
```

```
 Category feature unique value are 38 in number
PdDistrict unique features are 10 in number
```

Our data looks to be a mix of *categorical* variables like Crime `Category` or `PdDistrict`, and *numerical* variables like the `day_of_year` (1-365) and `time_in_hours` (time of day, converted to a float). We also have `X` and `Y` which refer to GPS coordinates, and `Resolution` which is our label.

Let's visualize our data:

[ ]:
```
import graphings
import numpy as np

# Crime category
graphings.multiple_histogram(dataset, label_x ='Category',␣
 ↪label_group="Resolution", histfunc='sum', show=True)
```

- Larsony/Thef was overwhelmingly the most common crime reported

[ ]:
```
# District
graphings.multiple_histogram(dataset, label_x="PdDistrict",␣
 ↪label_group="Resolution", show=True)
```

- Different police districts reported different volumes of crime
- Different police districts reported different success rates resolving crimes

[ ]:
```
# Map of crimes
graphings.scatter_2D(dataset, label_x="X", label_y="Y",␣
 ↪label_colour="Resolution", title="GPS Coordinates", size_multiplier=0.8,␣
 ↪show=True)
```

- Most reported crimes were not resolved in 2016

```
[ ]: # Day of the week
graphings.multiple_histogram(dataset, label_group="Resolution",␣
 ↪label_x="DayOfWeek", show=True)
```

- Friday and Saturday typically had more crimes than other days

```
[ ]: # day of the year
# For graphing we simplify this to week or the graph becomes overwhelmed with␣
 ↪bars
dataset["week_of_year"] = np.round(dataset.day_of_year / 7.0)
graphings.multiple_histogram(dataset,
                    label_x='week_of_year',
                    label_group='Resolution',
                    histfunc='sum', show=True)
```

```
[ ]: del dataset["week_of_year"]
```

It always pays to inspect your data before diving in. What we can see here is that:

- Most reported crimes were not resolved in 2016
- Different police districts reported different volumes of crime
- Different police districts reported different success rates resolving crimes
- Friday and Saturday typically had more crimes than other days
- Larsony/Theft was overwhelmingly the most common crime reported

1.2 Finalising Data preparation

Let's finalise our data preparation by one-hot encoding our categorical features:

```
[ ]: # One-hot encode categorical features
dataset = pandas.get_dummies(dataset, columns=["Category", "PdDistrict"],␣
 ↪drop_first=False)
```

```
   DayOfWeek  Resolution           X          Y  day_of_year  time_in_hours  \
0          5        True -122.403405  37.775421           29      11.000000
1          5        True -122.403405  37.775421           29      11.000000
2          1        True -122.388856  37.729981          116      14.983333
3          2       False -122.412971  37.785788            5      23.833333
4          5       False -122.419672  37.765050            1       0.500000

   Category_ARSON  Category_ASSAULT  Category_BAD CHECKS  Category_BRIBERY  \
0               0                 0                    0                 0
1               0                 0                    0                 0
2               0                 0                    0                 0
3               0                 0                    0                 0
4               0                 0                    0                 0

   …  PdDistrict_BAYVIEW  PdDistrict_CENTRAL  PdDistrict_INGLESIDE  \
```

```
0  …                0                0                0
1  …                0                0                0
2  …                1                0                0
3  …                0                0                0
4  …                0                0                0

   PdDistrict_MISSION  PdDistrict_NORTHERN  PdDistrict_PARK  \
0                   0                    0                0
1                   0                    0                0
2                   0                    0                0
3                   0                    0                0
4                   1                    0                0

   PdDistrict_RICHMOND  PdDistrict_SOUTHERN  PdDistrict_TARAVAL  \
0                    0                    1                   0
1                    0                    1                   0
2                    0                    0                   0
3                    0                    0                   0
4                    0                    0                   0

   PdDistrict_TENDERLOIN
0                      0
1                      0
2                      0
3                      1
4                      0

[5 rows x 54 columns]
```

[ ]:
```
dataset.head()
```

[ ]:
```
   DayOfWeek  Resolution           X          Y  day_of_year  time_in_hours  \
0          5        True -122.403405  37.775421           29      11.000000
1          5        True -122.403405  37.775421           29      11.000000
2          1        True -122.388856  37.729981          116      14.983333
3          2       False -122.412971  37.785788            5      23.833333
4          5       False -122.419672  37.765050            1       0.500000

   Category_ARSON  Category_ASSAULT  Category_BAD CHECKS  Category_BRIBERY  \
0               0                 0                    0                 0
1               0                 0                    0                 0
2               0                 0                    0                 0
3               0                 0                    0                 0
4               0                 0                    0                 0

   …  PdDistrict_BAYVIEW  PdDistrict_CENTRAL  PdDistrict_INGLESIDE  \
0  …                   0                   0                     0
```

```
1  …                    0                   0                      0
2  …                    1                   0                      0
3  …                    0                   0                      0
4  …                    0                   0                      0

   PdDistrict_MISSION  PdDistrict_NORTHERN  PdDistrict_PARK  \
0                   0                    0                0
1                   0                    0                0
2                   0                    0                0
3                   0                    0                0
4                   1                    0                0

   PdDistrict_RICHMOND  PdDistrict_SOUTHERN  PdDistrict_TARAVAL  \
0                    0                    1                   0
1                    0                    1                   0
2                    0                    0                   0
3                    0                    0                   0
4                    0                    0                   0

   PdDistrict_TENDERLOIN
0                      0
1                      0
2                      0
3                      1
4                      0

[5 rows x 54 columns]
```

We also need to make a training and test set.

Did you notice how much data we were working with before? If not, re-check the printouts from above.

We have over 150,000 samples to work with. That is a very large amount of data. Due to the sheer size, we can afford to have a larger proportion in the training set that we would normally have.

[ ]:
```python
from sklearn.model_selection import train_test_split

# Split the dataset in an 90/10 train/test ratio.
# We can afford to do this here because our dataset is very very large
# Normally we would choose a more even ratio
train, test = train_test_split(dataset, test_size=0.1, random_state=2,␣
 ↪shuffle=True)

print("Data shape:")
print("train", train.shape)
print("test", test.shape)
```

```
Data shape:
```

```
train (135387, 54)
test (15044, 54)
```

1.3 Model assessment code

We will fit several models here, so to maximise code reuse, we will make a dedicated method that trains a model and then tests it.

Our test stage uses a metric called "balanced accuracy", which we will refer to as "accuracy" for short throughout this exercise. It is not critical that you understand this metric for these exercises, but in essence this is between `0` and `1`: * `0` means no answers were correct * `1` means all answers were correct

Balanced accuracy takes into account that our data set has more unresolved than resolved crimes. We will cover what this means in later learning material in this course.

[ ]:
```
from sklearn.metrics import balanced_accuracy_score

# Make a utility method that we can re-use throughout this exercise
# To easily fit and test out model

features = [c for c in dataset.columns if c != "Resolution"]
```

[ ]:
```
['DayOfWeek',
 'X',
 'Y',
 'day_of_year',
 'time_in_hours',
 'Category_ARSON',
 'Category_ASSAULT',
 'Category_BAD CHECKS',
 'Category_BRIBERY',
 'Category_BURGLARY',
 'Category_DISORDERLY CONDUCT',
 'Category_DRIVING UNDER THE INFLUENCE',
 'Category_DRUG/NARCOTIC',
 'Category_DRUNKENNESS',
 'Category_EMBEZZLEMENT',
 'Category_EXTORTION',
 'Category_FAMILY OFFENSES',
 'Category_FORGERY/COUNTERFEITING',
 'Category_FRAUD',
 'Category_GAMBLING',
 'Category_KIDNAPPING',
 'Category_LARCENY/THEFT',
 'Category_LIQUOR LAWS',
 'Category_LOITERING',
 'Category_MISSING PERSON',
 'Category_NON-CRIMINAL',
```

```
 'Category_OTHER OFFENSES',
 'Category_PORNOGRAPHY/OBSCENE MAT',
 'Category_PROSTITUTION',
 'Category_RECOVERED VEHICLE',
 'Category_ROBBERY',
 'Category_RUNAWAY',
 'Category_SECONDARY CODES',
 'Category_SEX OFFENSES, FORCIBLE',
 'Category_SEX OFFENSES, NON FORCIBLE',
 'Category_STOLEN PROPERTY',
 'Category_SUSPICIOUS OCC',
 'Category_TREA',
 'Category_TRESPASS',
 'Category_VANDALISM',
 'Category_VEHICLE THEFT',
 'Category_WARRANTS',
 'Category_WEAPON LAWS',
 'PdDistrict_BAYVIEW',
 'PdDistrict_CENTRAL',
 'PdDistrict_INGLESIDE',
 'PdDistrict_MISSION',
 'PdDistrict_NORTHERN',
 'PdDistrict_PARK',
 'PdDistrict_RICHMOND',
 'PdDistrict_SOUTHERN',
 'PdDistrict_TARAVAL',
 'PdDistrict_TENDERLOIN']
```

[ ]:
```
train.head()
```

[ ]:
```
        DayOfWeek  Resolution           X          Y  day_of_year  \
51292           4       False -122.412931  37.783834          182
65844           2       False -122.449918  37.716611           61
60018           6        True -122.421382  37.764948          212
130629          3       False -122.403405  37.775421          230
1455            4       False -122.407110  37.798646           28

        time_in_hours  Category_ARSON  Category_ASSAULT  Category_BAD CHECKS  \
51292       15.500000               0                 0                    0
65844        0.016667               0                 1                    0
60018       12.916667               0                 0                    0
130629      20.500000               0                 0                    0
1455         2.000000               0                 0                    0

        Category_BRIBERY  …  PdDistrict_BAYVIEW  PdDistrict_CENTRAL  \
51292                  0  …                   0                   0
65844                  0  …                   0                   0
```

```
60018                    0  ...                   0                    0
130629                   0  ...                   0                    0
1455                     0  ...                   0                    1

        PdDistrict_INGLESIDE  PdDistrict_MISSION  PdDistrict_NORTHERN  \
51292                      0                   0                    0
65844                      0                   0                    0
60018                      0                   1                    0
130629                     0                   0                    0
1455                       0                   0                    0

        PdDistrict_PARK  PdDistrict_RICHMOND  PdDistrict_SOUTHERN  \
51292                 0                    0                    0
65844                 0                    0                    0
60018                 0                    0                    0
130629                0                    0                    1
1455                  0                    0                    0

        PdDistrict_TARAVAL  PdDistrict_TENDERLOIN
51292                    0                      1
65844                    1                      0
60018                    0                      0
130629                   0                      0
1455                     0                      0

[5 rows x 54 columns]
```

[ ]:
```
def fit_and_test_model(model):
    '''
    Trains a model and tests it against both train and test sets
    '''

    global features

    # Train the model
    model.fit(train[features], train.Resolution)

    # Assess its performance
    # -- Train
    predictions = model.predict(train[features])
    train_accuracy = balanced_accuracy_score(train.Resolution,predictions)

    # -- Test
    predictions = model.predict(test[features])
    test_accuracy = balanced_accuracy_score(test.Resolution, predictions)

    return train_accuracy, test_accuracy
```

```
print("Ready to go!")
```

```
Ready to go!
```

1.4 Fitting a decision tree

Let's use a decision tree to help us determine whether a not a crime will be resolved. Decision trees are categorisation models that break decisions down into multiple steps. They can be likened to a flow chart, with a decision being made at each subsequent level of the tree.

[ ]:
```
import sklearn.tree

# fit a simple tree using only three levels
model = sklearn.tree.DecisionTreeClassifier(random_state=2, max_depth=3)
train_accuracy, test_accuracy = fit_and_test_model(model)

print("Model trained!")
print("Train accuracy", train_accuracy)
print("Test accuracy", test_accuracy)
```

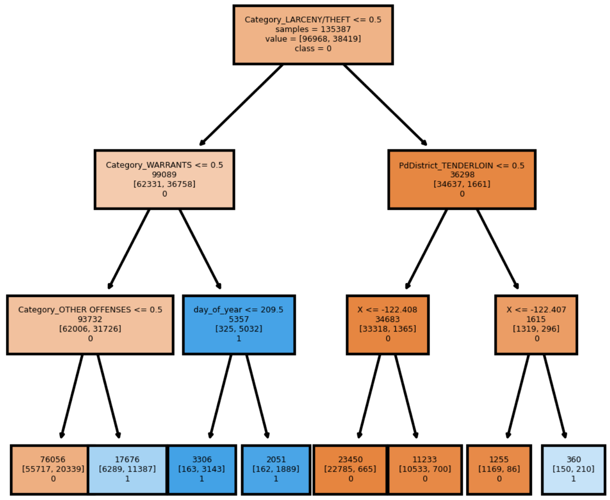
```
Model trained!
Train accuracy 0.6815388711342845
Test accuracy 0.674722862128782
```

That's not bad! Now that the model is trained, let's visualize it so we can get a better idea of how it works (and also see where it gets its tree moniker from!):

[ ]:
```
#----------------
from sklearn.tree import plot_tree
from matplotlib import pyplot as plt

plot = plt.subplots(figsize =(4,4), dpi=300)[0]
plot = plot_tree(model,
                fontsize=3,
                feature_names =features,
                class_names = ['0','1'], # class_names in ascending numerical␣
 ↪order
                label="root",
                impurity=False,
                filled=True)

plt.show()
```



All of the blue colored boxes correspond to prediction that a crime would be resolved.

Take a look at the tree to see what it thinks are important for predicting an outcome. Compare this to the graphs we made earlier. Can you see a relationship between the two?

The score we have is not bad, but the tree is pretty simple. Let's see if we can do better.

1.5 Improving performance through architecture

We will try and improve our model's performance by changing its architecture. Let's focus on the `maximum_depth` parameter.

Our previous tree was relatively simple and shallow with a `maximum_depth = 3`. Let's see what happens if we increase it to 100:

[ ]:
```
# fit a very deep tree
model = sklearn.tree.DecisionTreeClassifier(random_state=1, max_depth=100)

train_accuracy, test_accuracy = fit_and_test_model(model)
print("Train accuracy", train_accuracy)
print("Test accuracy", test_accuracy)
```

```
Train accuracy 0.9995864881946777
Test accuracy 0.7767968524220694
```

As you can imagine, a tree with a `maximum_depth = 100` is big. Too big to visualize here, so let's jump straight into seeing how the new model works on our training data.

Both the training and test accuracy have increased a lot. The training, however, has increased much more. While we're happy with the improvement in test accuracy, this is a clear sign of *overfitting*.

Overfitting with decision trees becomes even more obvious when we have more typical (smaller) sized datasets. Let's re-run the previous exercise but with only 100 training samples:

```
[ ]: # Temporarily shrink the training set to something
     # more realistic
     full_training_set = train
     train = train[:100]

     # fit the same tree as before
     model = sklearn.tree.DecisionTreeClassifier(random_state=1, max_depth=100)

     # Assess on the same test set as before
     train_accuracy, test_accuracy = fit_and_test_model(model)
     print("Train accuracy", train_accuracy)
     print("Test accuracy", test_accuracy)

     # Roll the training set back to the full set
     train = full_training_set
```

```
Train accuracy 1.0
Test accuracy 0.5850645895576951
```

The model performs badly on the test data. With reasonable sized datasets, *decision trees* are notoriously prone to *overfitting*. In other words, they tend to fit very well to the data they're trained on, but generalize very poorly to unknown data. This gets worse the deeper the tree gets or the smaller the training set gets. Let's see if we can mitigate this.

1.6 Pruning a tree

Pruning is the process of simplifying a *decision tree* so that it gives the best classification results while simultaneously reducing overfitting. There are two types of pruning: *pre-pruning* and *post-pruning*.

Pre-pruning involves restricting the model during training, so that it does not grow larger than is useful. We will cover this below.

Post-pruning is when we simplify the tree after training it. It does not involve the making of any design decision ahead of time, but simply optimizing the exisiting model. This is a valid technique but is quite involved, and so we do not cover it here due time constraints.

1.7 Prepruning

We can perform pre-pruning, by generating many models, each with different `max_depth` parameters. For each, we recording the *balanced accuracy* for the *test set*. To show that this is important even with quite large datasets, we will work with 10000 samples here.

```
[ ]: # Temporarily shrink the training set to 10000
     # for this exercise to see how pruning is important
     # even with moderately large datasets
     full_training_set = train
```

```
train = train[:10000]

# Loop through the values below and build a model
# each time, setting the maximum depth to the value
max_depth_range = [1,2,3,4,5,6,7,8,9,10,15,20,50,100]
accuracy_trainset =[]
accuracy_testset = []
for depth in max_depth_range:
    # Create and fit the model
    prune_model =sklearn.tree.
 ↪DecisionTreeClassifier(random_state=1,max_depth=depth)

    # Calculate and record its sensitivity
    train_accuracy, test_accuracy = fit_and_test_model(prune_model)
    accuracy_trainset.append(train_accuracy)
    accuracy_testset.append(test_accuracy)

# Plot the sensitivity as a function of depth
pruned_plot = pandas.
 ↪DataFrame(dict(max_depth=max_depth_range,accuracy=accuracy_trainset))
pruned_plot
```

[ ]:

	max_depth	accuracy
0	1	0.575199
1	2	0.687417
2	3	0.731848
3	4	0.731730
4	5	0.729716
5	6	0.742580
6	7	0.761021
7	8	0.776901
8	9	0.790832
9	10	0.801603
10	15	0.861056
11	20	0.901021
12	50	0.999643
13	100	0.999643

[ ]:

```
pandas.DataFrame(dict(max_depth=max_depth_range,␣
 ↪accuracy_train=accuracy_trainset, accuracy_test=accuracy_testset))
```

[ ]:

	max_depth	accuracy_train	accuracy_test
0	1	0.575199	0.560103
1	2	0.687417	0.673139
2	3	0.731848	0.718530
3	4	0.731730	0.717017
4	5	0.729716	0.703788

```
5      6    0.742580    0.717036
6      7    0.761021    0.732101
7      8    0.776901    0.742160
8      9    0.790832    0.745791
9     10    0.801603    0.743790
10    15    0.861056    0.731598
11    20    0.901021    0.731859
12    50    0.999643    0.721919
13   100    0.999643    0.721919
```

[ ]:
```
fig = graphings.line_2D(dict(train=accuracy_trainset, test=accuracy_testset),␣
 ↪x_range=max_depth_range, show=True)

# Roll the training set back to the full thing
train = full_training_set
```

We can see from our plot that the best *accuracy* is obtained for a `max_depth` of about 10. We are looking to simplify our tree, so we pick `max_depth = 10` for our final *pruned* tree:

[ ]:
```
# Temporarily shrink the training set to 10000
# for this exercise to see how pruning is important
# even with moderately large datasets
full_training_set = train
train = train[:10000]


# Not-pruned
model = sklearn.tree.DecisionTreeClassifier(random_state=1)
train_accuracy, test_accuracy = fit_and_test_model(model)
print("Unpruned Train accuracy", train_accuracy)
print("Unpruned Test accuracy", test_accuracy)


# re-fit our final tree to print out its performance
model = sklearn.tree.DecisionTreeClassifier(random_state=1, max_depth=10)
train_accuracy, test_accuracy = fit_and_test_model(model)
print("Train accuracy", train_accuracy)
print("Test accuracy", test_accuracy)

# Roll the training set back to the full thing
train = full_training_set
```

```
Unpruned Train accuracy 0.9996434937611408
Unpruned Test accuracy 0.7219189452075965
Train accuracy 0.8016027915999302
Test accuracy 0.7437900187051387
```

Our new and improved *pruned* model shows a marginally better *balanced accuracy* on the *test set* and much worse performance on the *training set* than the model that is not pruned. This means our pruning has significantly reduced overfitting.

If you would like, go back and change the number of samples to 100, and notice how the optimal `max_depth` changes. Think about why this might be (hint: model complexity versus sample size)

Another option that you may like to play with is how many features are entered into the tree. Similar patterns of overfitting can be observed by manipulating this. In fact, the number and type of the features provided to a decision tree can be even more important than its sheer size.

1.8 Summary

In this unit we covered the following topics: - Using visualization techniques to gain insights into our data - Building a simple *decision tree* model - Using the trained model to predict labels - *Pruning* a *decision tree* to reduce the effects of overfitting