Anomaly detection - Part 2

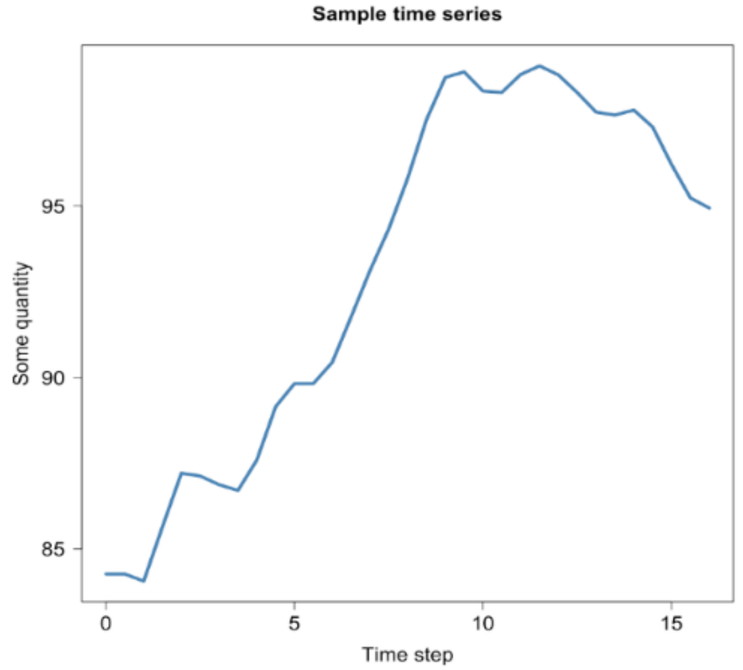
One should look for what is and not what he thinks should be. -Albert Einstein

Module completion checklist

Objective	Complete
Implement DBSCAN on time series data	
Examine why classification techniques are not useful for anomaly detection	
Identify SMOTE analysis and its implementation	

Time series data introduction

- The simplest forms of time series data consist of 2 variables:
  i. Some numeric **quantity**
  ii. **Time step** at which that quantity has occurred
- Here's a simple example use case visual for time series data
  - The **time step** in this case is equal to **0.5 seconds** and is recorded **on the `x-axis`**
  - The **quantity** is the **heart rate** and changes for every time step and is recorded **on the `y-axis`**



Dataset for this topic

- For this topic we will be working with the ***PJM*** energy consumption (East Region) dataset (`PJME_hourly.csv`)
- The dataset contains the following variables:
  - `Datetime` variable at each hour step
  - `PJME_MW` is the estimated energy consumption in Megawatts (MW) for PJM East Region
- We will be working with `PJME_MW` variable to **find anomalies in energy consumption** and detect things that fall out of the usual patterns
- **NOTE:** The exercise for this topic will use the PJM energy consumption (Load Combined) dataset (`PJM_Load_hourly.csv`), which contains the following variables:
  - `Datetime` variable at each hour step
  - `PJM_Load_MW` is the estimated energy consumption in Megawatts (MW) with load combined

Loading packages

Let’s load the packages we will be using:

```
import os
from pathlib import Path
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import pickle

from sklearn import metrics
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, roc_auc_score
from imblearn.over_sampling import SMOTE

from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import NearestNeighbors
from sklearn.cluster import DBSCAN
from sklearn.tree import DecisionTreeClassifier
```

Directory settings

- In order to maximize the efficiency of your workflow, you should encode your directory structure into `variables`
- We will use the `pathlib` library
- Let the `main_dir` be the variable corresponding to your course materials folder
- `data_dir` be the variable corresponding to your `data` folder

```
# Set 'main_dir' to location of the project folder
from pathlib import Path
home_dir = Path(".").resolve()
main_dir = home_dir.parent.parent
print(main_dir)
```

```
data_dir = str(main_dir) + "/data"
print(data_dir)
```

Time series data: load energy consumption

- We can now apply DBSCAN for time-series data
- Load the `PJME_hourly.csv` dataset and print the head

```
pjm_energy = pd.read_csv(str(data_dir)+"/PJME_hourly.csv")
pjm_energy.head()
```

```
             Datetime  PJME_MW
0  2002-12-31 01:00:00  26498.0
1  2002-12-31 02:00:00  25147.0
2  2002-12-31 03:00:00  24574.0
3  2002-12-31 04:00:00  24393.0
4  2002-12-31 05:00:00  24860.0
```

Time series data: preprocessing

- Let's convert `Datatime` variable from type **object** to **datetime**

```
pjm_energy['Datetime'] = pd.to_datetime(pjm_energy['Datetime'])
pjm_energy.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 145366 entries, 0 to 145365
Data columns (total 2 columns):
 #   Column    Non-Null Count   Dtype         
---  ------    --------------   -----         
 0   Datetime  145366 non-null  datetime64[ns]
 1   PJME_MW   145366 non-null  float64       
dtypes: datetime64[ns](1), float64(1)
memory usage: 2.2 MB
```

- We will filter the data to contain values for the year 2018

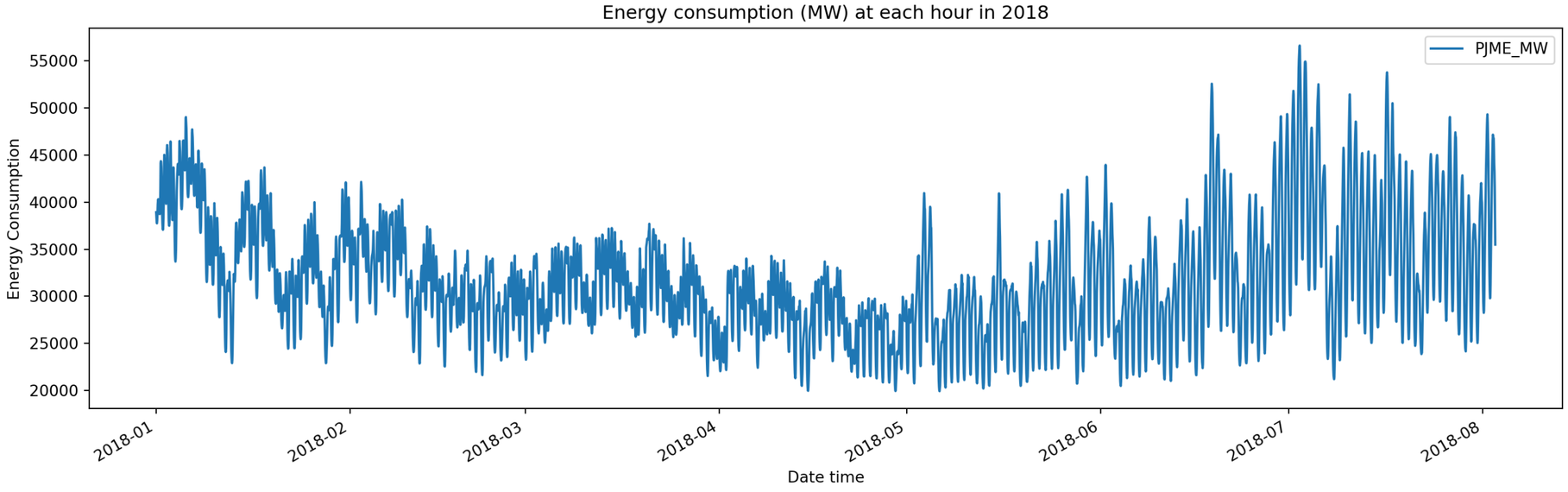
```
pjm_energy = pjm_energy[pjm_energy['Datetime'] > '2018-01-01 00:00:00']
pjm_energy.shape
```

```
(5135, 2)
```

Visualize the data: line plot

- A line plot is a common visualization type when plotting time series data where time is usually on the `x-axis` and the observation values (in this case, `Energy Consumption`) are on the `y-axis`

```
pjm_energy.plot(x='Datetime', y='PJME_MW', figsize=(17,5))
plt.xlabel('Date time')
plt.ylabel('Energy Consumption')
plt.title('Energy consumption (MW) at each hour in 2018')
plt.show()
```



Scaling on time series data

- We will use **StandardScaler()** to scale our data

```
scaler = StandardScaler()

# Scale the dataframe.
pjm_energy_scaled = scaler.fit_transform(pd.DataFrame(pjm_energy['PJME_MW']))
print(pjm_energy_scaled)
```

```
[[ 0.39848118]
 [ 0.04984823]
 [-0.19717311]
 ...
 [ 1.69156064]
 [ 1.33512366]
 [ 1.08730599]]
```

Optimal Eps determination

- Let’s find the optimal eps value
- We fit NearestNeighbor() with 5 neighbors on scaled data and plot the distances

```
nn_model = NearestNeighbors(n_neighbors=5)
nbrs = nn_model.fit(pjm_energy_scaled)
distances, indices =
nbrs.kneighbors(pjm_energy_scaled)
distances = np.mean(distances,axis=1)
distances = np.sort(distances, axis=0)
```

```
plt.figure(figsize=(5, 5))
plt.plot(distances)
plt.xlabel('index')
plt.ylabel('Distance (eps)')
plt.show()
```

Optimal Eps determination

- Zoom the plot for values after index 5000

```
plt.figure(figsize=(5, 5))
plt.plot(distances[5000:])
plt.xlabel('index')
plt.ylabel('Distance (eps)')
plt.show()
```

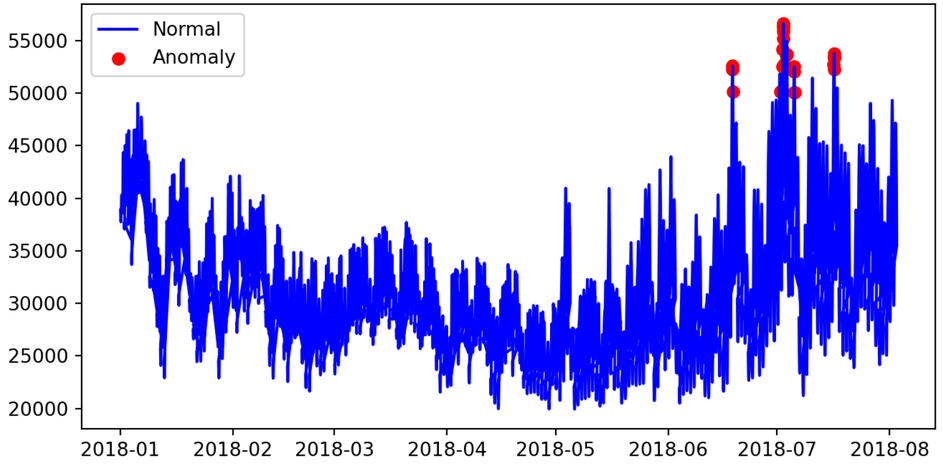
DBSCAN on time series data

- We will instantiate DBSCAN() with **eps: 0.03** and **min_samples: 5**

```
dbscan_energy = DBSCAN(eps = 0.03, metric='euclidean', min_samples=5, n_jobs = -1)

pjm_energy['anomaly'] =  dbscan_energy.fit_predict(pd.DataFrame(pjm_energy_scaled))
```

Visualize anomalies

```
# visualization
fig, ax = plt.subplots(figsize=(8,4))

# Anomalies detected have value = -1
a = pjm_energy.loc[pjm_energy['anomaly'] == -1,
['Datetime', 'PJME_MW']] #anomaly

# Plotting the anomalies
ax.plot(pjm_energy['Datetime'],
pjm_energy['PJME_MW'], color='blue', label =
'Normal')
ax.scatter(a['Datetime'],a['PJME_MW'],
color='red', label = 'Anomaly')
plt.legend()
plt.show()
```



Anomalies detected

```
pjm_energy[pjm_energy['anomaly'] == -1]
```

```
                    Datetime  PJME_MW  anomaly
140653 2018-07-16 15:00:00   52742.0       -1
140654 2018-07-16 16:00:00   53408.0       -1
140655 2018-07-16 17:00:00   53764.0       -1
140656 2018-07-16 18:00:00   53352.0       -1
140657 2018-07-16 19:00:00   52226.0       -1
140916 2018-07-05 14:00:00   50094.0       -1
140919 2018-07-05 17:00:00   52045.0       -1
140920 2018-07-05 18:00:00   52503.0       -1
140922 2018-07-05 20:00:00   50066.0       -1
140963 2018-07-03 13:00:00   53623.0       -1
140968 2018-07-03 18:00:00   53679.0       -1
140987 2018-07-02 13:00:00   52533.0       -1
140988 2018-07-02 14:00:00   54170.0       -1
140989 2018-07-02 15:00:00   55159.0       -1
140990 2018-07-02 16:00:00   55836.0       -1
140991 2018-07-02 17:00:00   56430.0       -1
140992 2018-07-02 18:00:00   56609.0       -1
140993 2018-07-02 19:00:00   56231.0       -1
```

Knowledge check 3

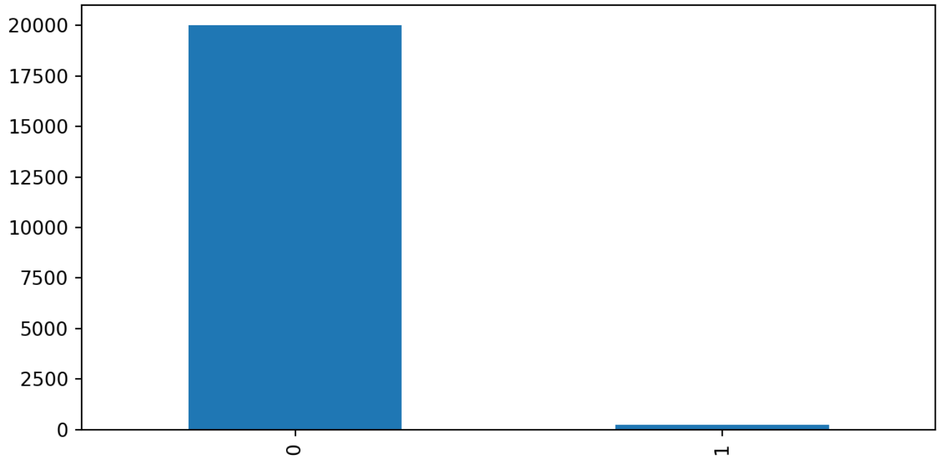
Exercise 3

Module completion checklist

Objective	Complete
Implement DBSCAN on time series data	✔
Examine why classification techniques are not useful for anomaly detection	
Identify SMOTE analysis and its implementation	

How anomaly detection is different?

- Anomaly detection is, at its core, a **classification problem**
- The key difference is that we distinguish between normal and anomalous behavior
- The anomalous observations do not conform to the expected pattern of other observations
- Because anomalous behavior is rare, the dataset is going to be highly imbalanced



Decision trees: what are they?

- Decision trees will be your **good friend** when you are data mining
- They are **intuitive** and popular because they **provide explicit rules for classification** and **cope well with heterogeneous data, missing data, and nonlinear effects**
- Decision trees **predict** the target value of an item by **mapping observations about the item**

Decision trees: pros and cons

- Decision trees are **great** when used for:
  - Classification and regression
  - Handling numerical and categorical data
  - Handling data with missing values
  - Handling data with nonlinear relationships between parameters

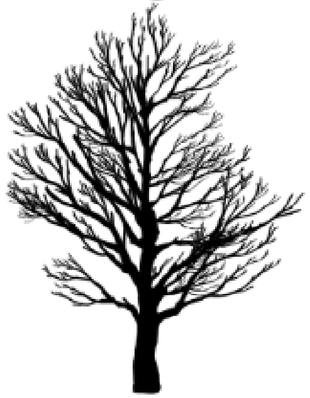
- Decision trees are **not very good** at:
  - **Generalization**: they are known for overfitting
  - **Robustness**: small variations in data can result in a different tree
  - **Mitigating bias**: if some classes dominate, trees may be unbalanced and biased

Decision trees: use cases

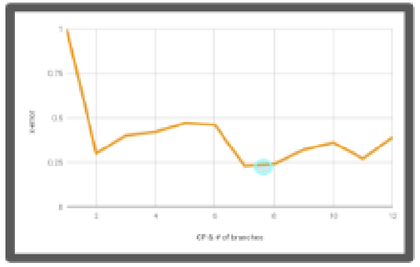
- Decision trees are **popular in a business setting** because they are intuitive
- They are easy to **walk through and explain to stakeholders who are not familiar with data science**
- Today, we will walk through the conceptual explanation and then use a small subset to see a decision tree in action
- We will then use it on the larger dataset so that we can compare the performance of the decision tree to previous and future models

Decision trees: process

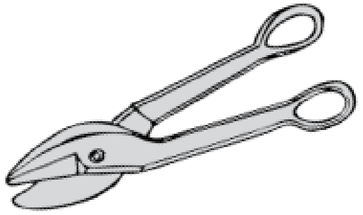
Step 1:
Grow tree on training data



Step 2:
Examine Model output



Step 3:
Prune Tree



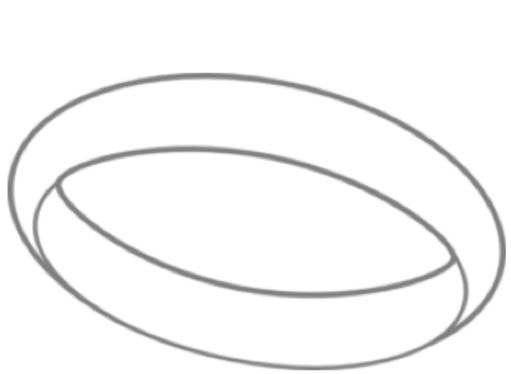
Step 4:
Check performance on test data

	Act +	Act -	
Pred +			
Pred -			

Growing decision trees: find the most important question

Which question is more important on a date?

The question with the most amount of relevant information

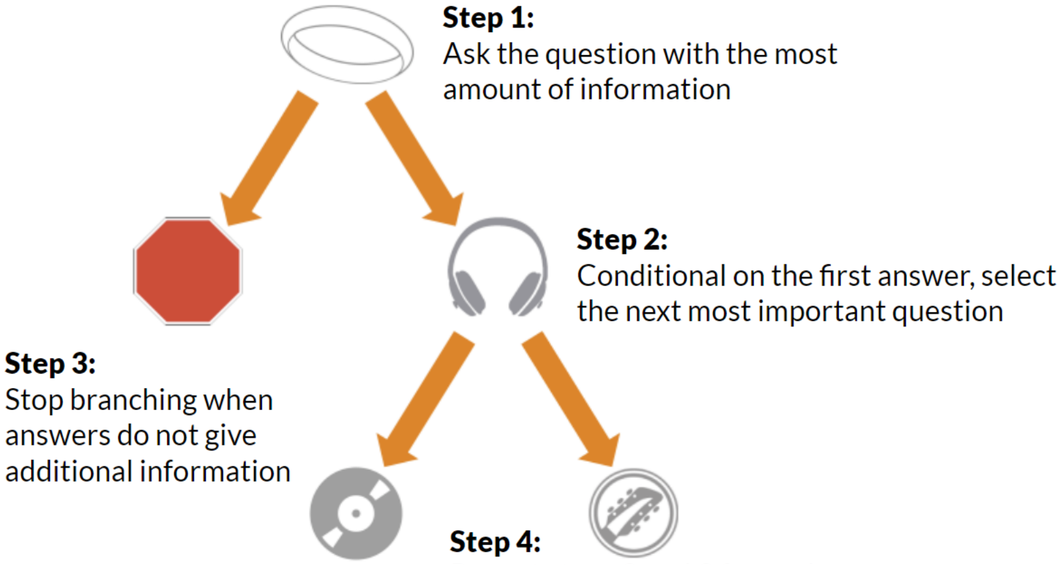


Are you married?

>

What music do you like?

Growing decision trees: four steps



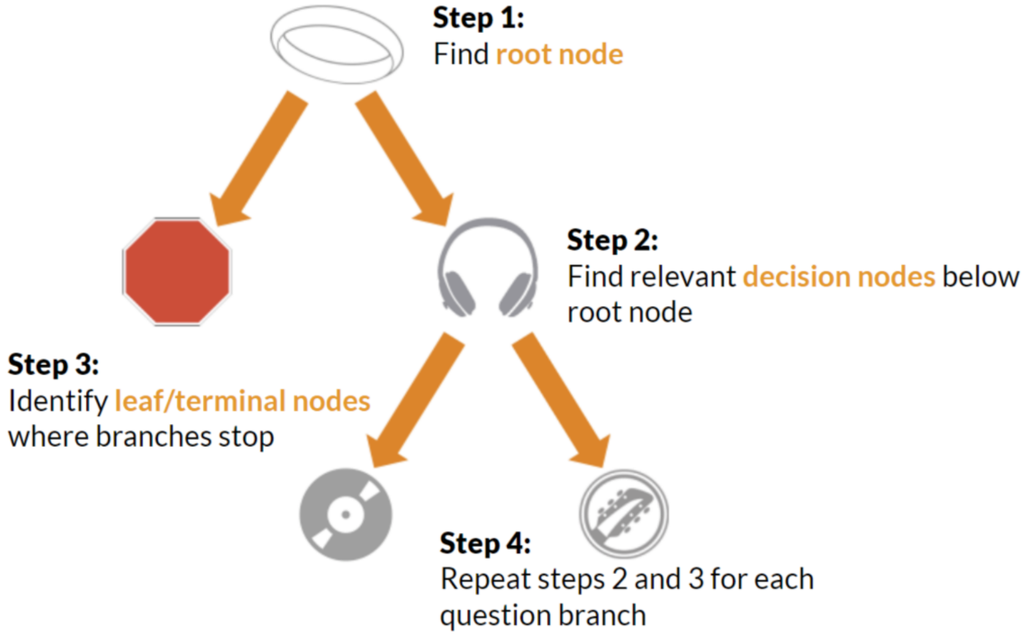
Step 1:
Ask the question with the most amount of information

Step 2:
Conditional on the first answer, select the next most important question

Step 3:
Stop branching when answers do not give additional information

Step 4:
Repeat steps 2 and 3 for each question branch

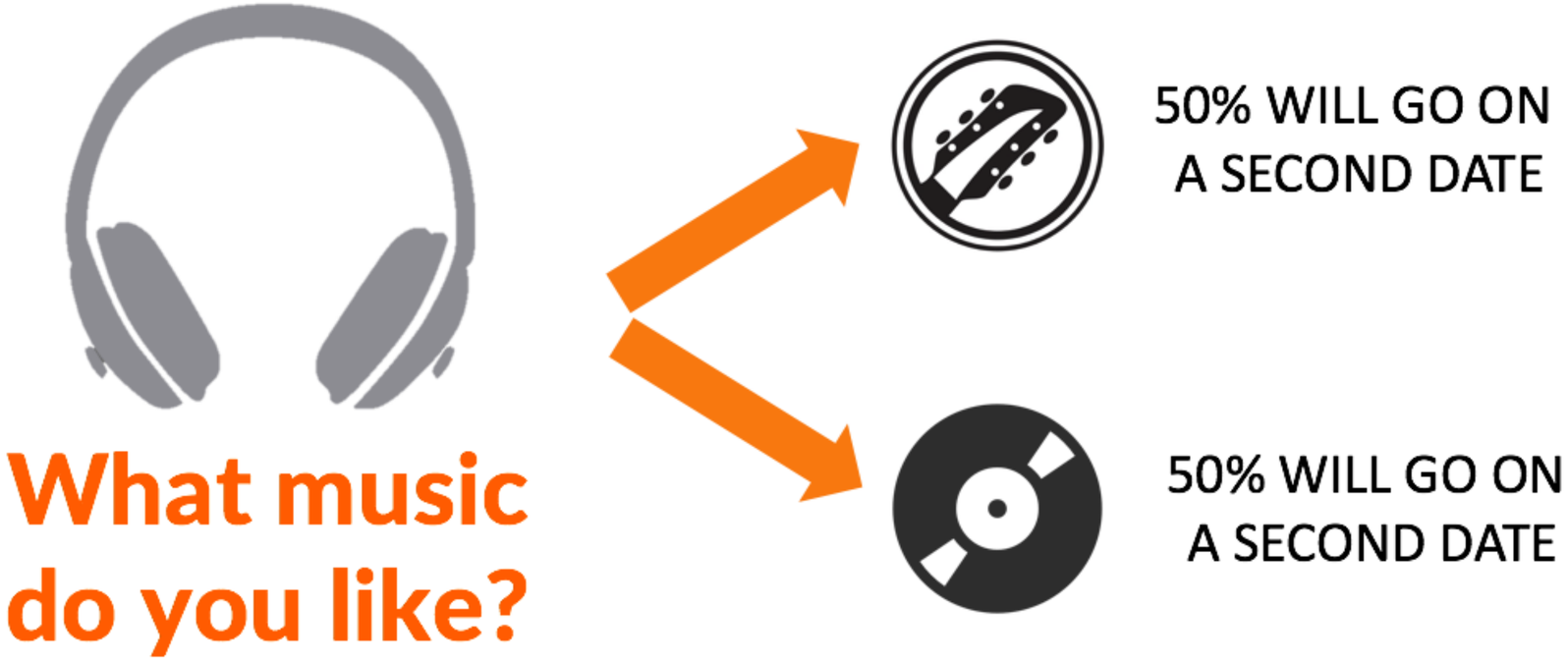
Growing decision trees: steps with vocabulary



Growing decision trees: when to stop

When should you stop asking questions?

When the answer no longer provides additional relevant information



Growing decision trees: the math of the splits

How do we decide which node to split and how to split it?

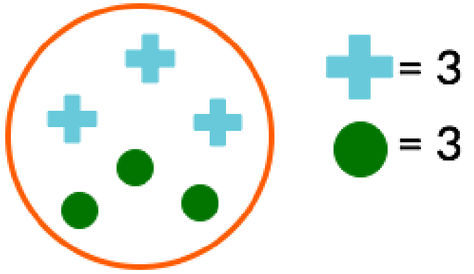
- There are two **impurity** functions that are most commonly used with tree-based models
  - Gini
  - Entropy
- The **`sklearn.tree` algorithm** uses Gini, so this is the method we will focus during our modeling

Growing decision trees: Gini

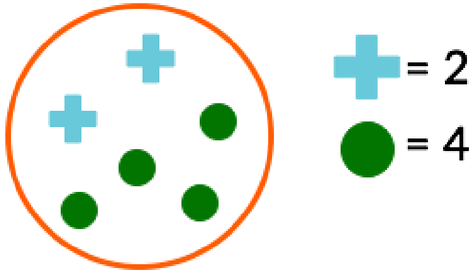
Gini measures the **probability of misclassification** in the model for each branch. Gini ranges from 0 to 1.

P*i* = the probability that a random selection would have state *i*

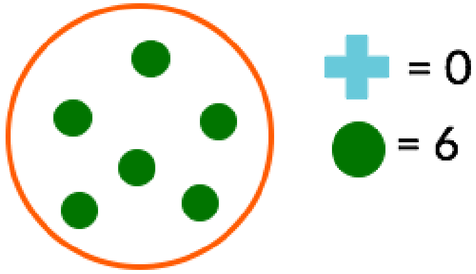
$$\text{Gini impurity} = 1 - \text{sum}[\ (P_i)^2\ ]$$



$1-(3/6)^2-(3/6)^2$



$1-(4/6)^2-(2/6)^2$



$1-(6/6)^2-(0/6)^2$

Prepare a dataset for decision tree modeling

- For this topic, we will again be working with the Payism dataset (`paysim_transactions.csv`) to determine if a particular transaction is fraudulent
- We will use `type`, `oldbalanceOrg`, `newbalanceOrg`, `OldbalanceDest` and `newbalanceDest`, as our predictors
- The other variables are categorical - for us to understand the technique better, we'll ignore the other variables

```
paysim = pd.read_csv(str(data_dir)+"/paysim_transactions.csv")
```

```
# Drop columns.
paysim = paysim.drop(['step', 'nameOrig', 'nameDest', 'isFlaggedFraud'], axis = 1)
```

```
paysim.columns
```

```
Index(['type', 'amount', 'oldbalanceOrg', 'newbalanceOrig', 'oldbalanceDest',
       'newbalanceDest', 'isFraud'],
      dtype='object')
```

Convert categorical to dummy

- Convert the `type` column to a dummy variable

```
paysim['type'] = pd.Categorical(paysim['type'])
paysim['type'] = paysim['type'].cat.codes
colname = pd.get_dummies(paysim['type'], prefix = 'type', drop_first = True)
paysim = pd.concat([paysim, colname], axis = 1)
paysim.drop(['type'], axis = 1, inplace = True)

paysim.columns
```

```
Index(['amount', 'oldbalanceOrg', 'newbalanceOrig', 'oldbalanceDest',
       'newbalanceDest', 'isFraud', 'type_1', 'type_2', 'type_3', 'type_4'],
      dtype='object')
```

Decision tree classification

- We will build a decision tree model with **max_depth = 10** and see the result
- This will be our baseline for comparison with all other models

```
# Select predictors and target.
y = paysim['isFraud']
X = paysim.drop(['isFraud'], axis = 1)

# Build a logistic regression model.
np.random.seed(1)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.30)
dtree = DecisionTreeClassifier(max_depth = 10)
dtree.fit(X_train, y_train)
```

```
DecisionTreeClassifier(max_depth=10)
```

Predict the target

- Let’s use our model to predict our training and test dataset

```
dtree_y_train_pred = dtree.predict(X_train)
dtree_y_test_pred = dtree.predict(X_test)
dtree_accuracy = metrics.accuracy_score(y_test, dtree_y_test_pred)
print("Accuracy of test data:\t", dtree_accuracy)
```

```
Accuracy of test data:   0.9960435212660732
```

```
# ROC AUC value.
roc_auc_score(y_test, dtree_y_test_pred)
```

```
0.8356542010733131
```

- We got 0.99 accuracy
- Do you think it is a good model for fraud detection?

Confusion matrix of training data

```
print('Confusion Matrix - Training Dataset')
```

```
Confusion Matrix - Training Dataset
```

```
print(pd.crosstab(y_train, dtree_y_train_pred, rownames = ['True'], colnames = ['Predicted'],
margins = True))
```

```
Predicted       0    1    All
True
0           13998    3  14001
1              20  133    153
All         14018  136  14154
```

- Out of 153 fraud transactions, 133 are predicted as fraud

```
print('Percentage of accurate fraud cases is ', 133/153)
```

```
Percentage of accurate fraud cases is  0.869281045751634
```

Confusion matrix of test data

```
print('Confusion Matrix - Testing Dataset')
```

```
Confusion Matrix - Testing Dataset
```

```
print(pd.crosstab(y_test, dtree_y_test_pred,
rownames = ['True'], colnames = ['Predicted'],
margins = True))
```

```
Predicted     0   1   All
True
0          5997   2  5999
1            22  45    67
All        6019  47  6066
```

- Out of 67 fraud transactions, 45 are predicted as fraud

```
print('Percentage of accurate fraud cases is',
45/67)
```

```
Percentage of accurate fraud cases is
0.6716417910447762
```

- Even though our model shows 99% accuracy, this model can still be improved
- It does not capture the fraud cases as accurately as 99%
- Also, since the data is highly imbalanced, using accuracy as our validation metric might be misleading
- We will use other metrics

TPR as our metric

True positive rate (Sensitivity): how often does it predict yes?

TP / actual yes

	Predicted Y1	**Predicted Y2**	**Actual totals**
Y1	True Negative (TN)	False Positive (FP)	Total negatives
Y2	False Negative (FN)	**True Positive (TP)**	**Total positives**
Predicted totals	Total predicted negatives	Total predicted positives	Total

TNR as our metric

True Negative Rate (Specificity): when it's actually no, how often does it predict no?
TN / actual no

	Predicted Y1	Predicted Y2	Actual totals
Y1	True Negative (TN)	False Positive (FP)	Total negatives
Y2	False Negative (FN)	True Positive (TP)	Total positives
Predicted totals	Total predicted negatives	Total predicted positives	Total

Find TPR and TNR

- We will find true positive rate (TPR) and true negative rate (TNR) for our model
- For our fraud dataset, `not fraud - 0` is negative and `fraud - 1` class is positive

```
tn, fp, fn, tp = confusion_matrix(y_test, dtree_y_test_pred).ravel()

# Find the TNR.
non_fraud_eval = tn / (tn + fp)
print(non_fraud_eval)
```

```
0.9996666111018503
```

```
# Find the TPR.
fraud_eval = tp / (tp + fn)
print(fraud_eval)
```

```
0.6716417910447762
```

- It is expected that **not fraud** will be predicted better than fraud cases because **not fraud** is the majority class

Save the metric

```
performance_df = pd.DataFrame(columns = ['model_name', 'TPR', 'TNR'])

s = pd.Series(['Decision_tree_baseline', fraud_eval, non_fraud_eval],
              index=['model_name', 'TPR', 'TNR'])
performance_df = performance_df.append(s, ignore_index = True)
performance_df
```

```
                model_name       TPR       TNR
0  Decision_tree_baseline  0.671642  0.999667
```

- We will keep this decision tree model as our baseline to compare with other anomaly models we will create

Knowledge check 4

Exercise 4

Module completion checklist

Objective	Complete
Implement DBSCAN on time series data	✔
Examine why classification techniques are not useful for anomaly detection	✔
Identify SMOTE analysis and its implementation	

Anomaly detection - supervised methods

- As we saw, we can still use traditional classification models for anomaly detection
- But there are two things to take care of before using any classification model
- **Handling the imbalanced classes**
  - Our outlier class is really small compared to the normal data points
  - We need to convert this imbalanced data to balanced class data
  - This can make our classification models perform better
- **Validation metric**
  - We already saw that accuracy cannot be a good metric here
  - Our aim should be accurately predicting the anomalies as much as possible
  - It is far more important than mislabeling normal objects as outliers
  - So, in this case, we should use a different evaluation metric

Handling imbalanced classes

- Imbalanced classes are a common problem not just in anomaly detection, but in many other applications
- The techniques we are going to learn in this section can be applied to any machine learning application, not just anomaly detection, you come across when you see class imbalance

Changing the performance metric

- We already saw that accuracy is not a good metric for evaluating our model if our data is imbalanced
- There are other metrics which we can use:
- **Precision**
  - Number of true positives divided by all positive predictions
  - Also called positive predicted value and measures classifier's exactness
  - Low precision indicates a high number of false positives
- **Recall**
  - Number of positives divided by number of positives in data
  - It is also called true positive rate or sensitivity and measures classifier's completeness
  - Low recall indicates a high number of false negatives
- **F1 score**
  - Weighted average of precision and recall

Resampling techniques

- Resampling is a technique we can use to have more balanced data
- The main aim is to decrease the majority class samples or increase the minority class samples
- The different resampling techniques are:
  - Random undersampling
  - Random oversampling
  - Informed over sampling aka SMOTE

Resampling techniques

- **Random undersampling**
  - This technique randomly eliminates majority class observations to make the data balanced
  - The main problem with this technique is that we could potentially lose useful information in our data by removing the observations
- **Random oversampling**
  - This method increases the number of instances in the minority class by randomly replicating them in order to present a higher representation of the minority class
  - This helps us avoid losing information and certainly outperforms undersampling
  - But it replicates the likelihood of overfitting since it replicates the minority class observations

SMOTE sampling

- SMOTE stands for **S**ynthetic **M**inority **O**versampling **TE**chnique
- It is also knows as “informed oversampling”
- This method oversamples the minority class but avoids overfitting
- Instead of randomly oversampling the minority class observations, it creates new observations based on the original training instance
- For example, let’s say there are two observations in training data of the minority class:
  - Random oversampling just replicates these two observations exactly
  - SMOTE creates a third artificial one, similar to the two observations, in the middle of the original data
- We will use SMOTE sampling for our fraud data and check our result

SMOTE in Python

- In Python, we have a package to perform SMOTE for us
- Read more about the package here: ***SMOTE***

Docs » imbalanced-learn API » `imblearn.over_sampling` .SMOTE

`imblearn.over_sampling` .SMOTE

class **`imblearn.over_sampling.SMOTE`**(*sampling_strategy='auto', random_state=None, k_neighbors=5, m_neighbors='deprecated', out_step='deprecated', kind='deprecated', svm_estimator='deprecated', n_jobs=1, ratio=None*) [source]

Class to perform over-sampling using SMOTE.

This object is an implementation of SMOTE - Synthetic Minority Over-sampling Technique as presented in [R001eabbe5dd7-1].

Read more in the User Guide.

Parameters:

sampling_strategy:float, str, dict or callable, (default='auto')

Sampling information to resample the data set.

- When `float`, it corresponds to the desired ratio of the number of samples in the minority class over the number of samples in the majority class after resampling. Therefore, the ratio is expressed as $\alpha_{os} = N_{rm}/N_M$ where N_{rm} is the number of samples in the minority class after resampling and N_M is the number of samples in the majority class.

SMOTE in fraud dataset

- Let's create synthetic oversampling in our fraud dataset

```
sm = SMOTE(random_state = 1)
X_train_new, y_train_new =
sm.fit_resample(X_train, y_train)
```

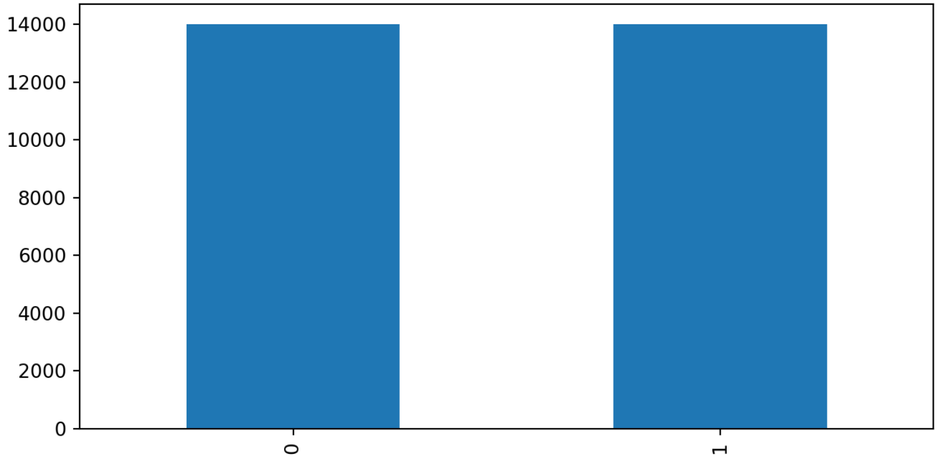
```
# Shape of X_train.
print(X_train.shape)
```

```
(14154, 9)
```

```
# Print shape of X_train_new.
print(X_train_new.shape)
```

```
(28002, 9)
```

```
# Double check that the data has been balanced.
pd.Series(y_train_new).value_counts().plot.bar()
```



Fit the model and predict

- Fit the model on the new data and predict

```
# Fit the model.
dtree.fit(X_train_new, y_train_new)

# Prediction for training data.
```

```
DecisionTreeClassifier(max_depth=10)
```

```
train_pred_sm = dtree.predict(X_train_new)

# Prediction for the test data.
test_pred_sm = dtree.predict(X_test)
train_pred_sm = dtree.predict(X_train_new)
```

Confusion matrix of training data

```
print('Confusion Matrix - Training Dataset')
```

```
Confusion Matrix - Training Dataset
```

```
print(pd.crosstab(y_train_new, train_pred_sm, rownames = ['True'], colnames = ['Predicted'],
margins = True))
```

```
Predicted      0      1    All
True
0          13939     62  14001
1              2  13999  14001
All        13941  14061  28002
```

- Out of 14,001 fraud transactions, 13,999 are predicted as fraud

```
print('Percentage of accurate fraud cases: ', 13999/14001)
```

```
Percentage of accurate fraud cases:  0.9998571530604957
```

Confusion matrix of test data

```
print('Confusion Matrix - Testing Dataset')
```

```
Confusion Matrix - Testing Dataset
```

```
print(pd.crosstab(y_test, test_pred_sm, rownames
= ['True'], colnames = ['Predicted'], margins =
True))
```

```
Predicted     0    1   All
True
0          5947   52  5999
1             9   58    67
All        5956  110  6066
```

- Out of 67 fraud transactions, 58 are predicted as fraud

```
print('Percentage of accurate fraud cases ',
58/67)
```

```
Percentage of accurate fraud cases
0.8656716417910447
```

- The SMOTE approach has improved our predictions from 0.67 on test data to 0.98 when compared to our previous baseline decision tree model
- We just balanced our dataset and did not add anything apart from that
- This technique decreases our risk to fraud in any transaction better than the previous method

Find TPR and TNR and save

```
# Find TPR and TNR and save the result.
tn, fp, fn, tp = confusion_matrix(y_test, test_pred_sm).ravel()
non_fraud_eval = tn / (tn + fp)
print(non_fraud_eval)
```

```
0.9913318886481081
```

```
fraud_eval = tp / (tp + fn)
print(fraud_eval)
```

```
0.8656716417910447
```

- Our TPR has increased a lot when we used SMOTE analysis

Add scores to the performance dataframe

- Let's add our SMOTE model score to the performance dataframe and then pickle the dataframe
- This way, we can use the `performance_df` dataframe across all our anomaly detection algorithms

```
s = pd.Series(['SMOTE', fraud_eval, non_fraud_eval],
              index=['model_name', 'TPR', 'TNR'])
performance_df = performance_df.append(s, ignore_index = True)
performance_df
```

	model_name	TPR	TNR
0	Decision_tree_baseline	0.671642	0.999667
1	SMOTE	0.865672	0.991332

```
pickle.dump(performance_df, open(str(data_dir) + "/performance_anomalies.sav","wb"))
```

- What other classification algorithm can you try out?
  - Logistic regression! You can try replacing decision trees with logistic regression and observe the results
  - Try the vanilla model and then with SMOTE analysis

Knowledge check 5

Exercise 5

Module completion checklist

Objective	Complete
Implement DBSCAN on time series data	✔
Examine why classification techniques are not useful for anomaly detection	✔
Identify SMOTE analysis and its implementation	✔

What's next?

- In this module we:
  - implemented DBSCAN on time series data
  - examined why classification techniques are not useful for anomaly detection
  - learned about SMOTE analysis
- In the next module, we will discuss anomaly detection techniques, including Local Outlier Factor

Congratulations on completing this module!