DATA SOCIETY®

Advanced classification - day 5

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

Module completion checklist

Objective	Complete
Define anomaly concepts and uses	
Differentiate between types of anomalies	
Create a problem statement for our fraud dataset	
Examine why classification techniques are not useful for anomaly detection	
Summarize different techniques for anomaly detection	
Identify SMOTE analysis and its implementation	
Describe the isolation forest algorithm	
Implement isolation forest to detect credit fraud	

Directory settings

- In order to maximize the efficiency of your workflow, you should encode your directory structure into `variables`
- Let the `main_dir` be the variable corresponding to your `af-werx` folder

```
# Set `main_dir` to the location of your `af-werx` folder (for Linux).
main_dir = "/home/[username]/Desktop/af-werx"
```

```
# Set `main_dir` to the location of your `af-werx` folder (for Mac).
main_dir = '/Users/[username]/Desktop/af-werx'
```

```
# Set `main_dir` to the location of your `af-werx` folder (for Windows).
main_dir = "C:\\Users\\[username]\\Desktop\\af-werx"
```

```
# Make `data_dir` from the `main_dir` and
# remainder of the path to data directory.
data_dir = main_dir + "/data"
```

Loading packages

Let's load the packages we will be using:

```
import os
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import IsolationForest
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
```

```
Using TensorFlow backend.
```

```
import warnings

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

Working directory

- Set working directory to the `data_dir` variable we set
- We do this using the `os.chdir` function, change directory
- We can then check the working directory using `.getcwd()`
- For complete documentation of the `os` package, ***click here***

```
# Set working directory.
os.chdir(data_dir)
```

```
# Check working directory.
print(os.getcwd())
```

```
/home/[user-name]/Desktop/af-werx/data
```

What is an anomaly?

- We have already seen the concept of outliers in detail in our regression modules
- Anomalies are nothing but outliers
- Anomalies are the objects with **behaviors that are very different from expectations**
- **Anomaly detection**, the process of finding those anomalies, is also called outlier detection
- Today, we are going to learn some more **anomaly detection techniques**

Use cases of anomaly detection

- Imagine you are the transaction auditor in a credit card company
- To protect your customers from credit card fraud, you pay special attention to card usages that are rather different from typical cases
- For example, if the purchase amount is much bigger than usual or if the purchase occurs far from the owner's resident city, then the transaction is suspicious
- Anomaly detection techniques are useful in these cases

Other use cases

- Intrusion detection in computer networks
- Part failure detection in manufacturing or even aircraft maintenance
- Risk detection
- Disease detection in a hospital

Outliers

- As we know, an outlier is the data object that deviates significantly from the rest of the objects
- Outliers are different from noisy data
- Outlier detection can be related to novelty detection in evolving datasets
- Novelty detection may identify new topics and trends in a timely manner

Types of outliers

- There are three types of outliers:
  - Global outliers
  - Contextual outliers
  - Collective outliers
- **Global outliers**
  - Global outliers are the data points that deviate from the rest of the data
  - They are called as the point anomalies and are the simplest type of outliers

Contextual outliers

- Contextual outliers are the conditional outliers because they are conditional on the selected context
- The temperature today is 24 degrees Celsius
- Is that an outlier?
- It depends on the time and location
- If it is winter in Toronto, then yes, it is an outlier!

Collective outliers

- In the whole dataset, if a subset of data objects deviate significantly from the entire dataset, then they are called as collective outliers
- You handle the shipments of goods everyday in a supply chain organization
- If shipment of an order is delayed, it is not an outlier because delay occurs from time to time
- What if 100 orders are delayed on a single day?
- This is a change in usual behavior, which means those 100 shipments are outliers
- **In our module, we will concentrate on global outlier detection**

Module completion checklist

Objective	Complete
Define anomaly concepts and uses	✔
Differentiate between types of anomalies	✔
Create a problem statement for our fraud dataset	
Examine why classification techniques are not useful for anomaly detection	
Summarize different techniques for anomaly detection	
Identify SMOTE analysis and its implementation	
Describe the isolation forest algorithm	

Datasets for the day

- Today, we are going to use two datasets
- Paysim transactions for class
  - It is the fraud detection dataset which has rows of credit transactions and also the target specifying if the particular transaction is a fraud transaction/not
  - Read more about the dataset ***here***
  - We want to accurately find if a particular transaction is a fraud or not
- Seismic hazards for exercises
  - It contains data on the mining activity in different locations and our aim is to detect whether the location is prone to seismic (earthquake) hazards or not
  - Since earthquakes happen less, the rows that are prone to hazards are anomalous from rest of the observations
  - Read more about the dataset ***here***

Load the dataset

- Load `paysim_transactions.csv` dataset and print the head

```
paysim = pd.read_csv("paysim_transactions.csv")
paysim.head()
```

```
   step      type      amount  ...
newbalanceDest  isFraud  isFlaggedFraud
0   308  CASH_OUT    94270.99  ...
486682.07        0               0
1   215  TRANSFER  1068883.00  ...
5165788.35       0               0
2   326  TRANSFER  2485281.21  ...
2663110.80       0               0
3   371   PAYMENT     2243.36  ...
0.00         0               0
4   283   PAYMENT     5845.82  ...
0.00         0               0

[5 rows x 11 columns]
```

```
paysim.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20220 entries, 0 to 20219
Data columns (total 11 columns):
step              20220 non-null int64
type              20220 non-null object
amount            20220 non-null float64
nameOrig          20220 non-null object
oldbalanceOrg     20220 non-null float64
newbalanceOrig    20220 non-null float64
nameDest          20220 non-null object
oldbalanceDest    20220 non-null float64
newbalanceDest    20220 non-null float64
isFraud           20220 non-null int64
isFlaggedFraud    20220 non-null int64
dtypes: float64(5), int64(3), object(3)
memory usage: 1.7+ MB
```

Understand the dataset

```
paysim.columns
```

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

- `step` - the unit of time
- `type` - type of transaction
- `amount` - amount of transaction in local currency
- `nameOrig` - customer who started the transaction
- `oldbalanceOrg` - initial balance before the transaction
- `newbalanceOrig` - new balance after the transaction
- `nameDest` - customer who is the recipient of the transaction
- `oldbalanceDest` - initial balance recipient before the transaction
- `newbalanceDest` - new balance recipient after the transaction
- `isFraud` - our target if it's a fraud or not
- `isFlaggedFraud` - tells whether the transaction is flagged for fraudulent activity

Target of the dataset

- `isFraud` is our target which tells if a particular transaction is fraud (`1`) or not (`0`)

```
paysim['isFraud'].value_counts()
```

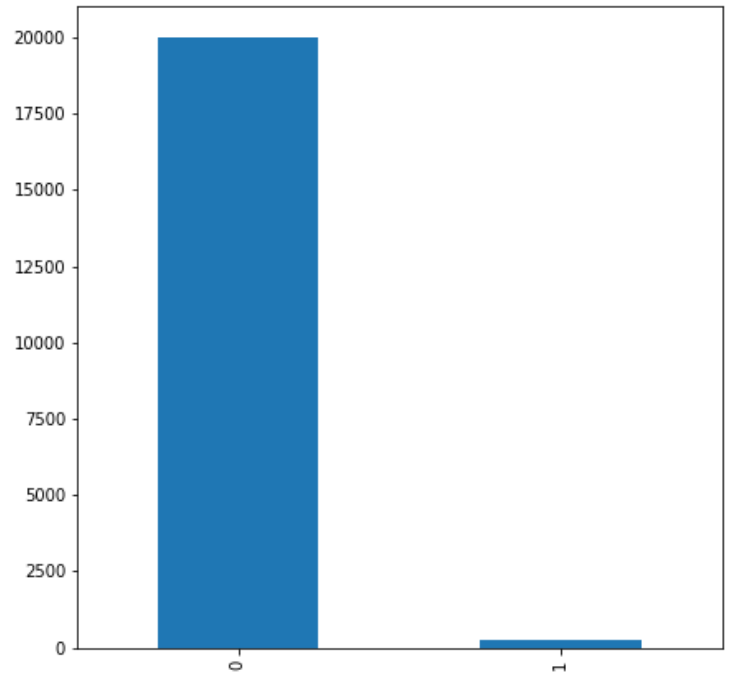
```
0    20000
1      220
Name: isFraud, dtype: int64
```

```
print(paysim['isFraud'].value_counts() / len(paysim))
```

```
0    0.98912
1    0.01088
Name: isFraud, dtype: float64
```

- Approximately 99% of the transactions are not fraud and only 1% of the transactions are fraud
- Anomalous behavior is rare and that is why we describe our dataset as highly imbalanced

```
paysim['isFraud'].value_counts().plot('bar')
```



- Our target is highly imbalanced between two classes

EDA of the dataset

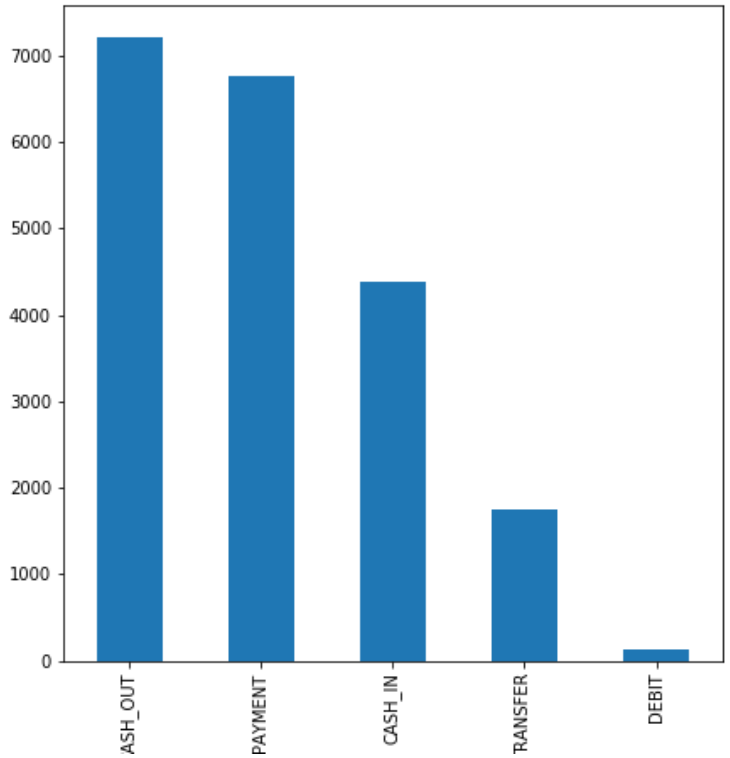
- Check for null values

```
paysim.isnull().sum()
```

```
step              0
type              0
amount            0
nameOrig          0
oldbalanceOrg     0
newbalanceOrig    0
nameDest          0
oldbalanceDest    0
newbalanceDest    0
isFraud           0
isFlaggedFraud    0
dtype: int64
```

- View the frequency of type of transactions

```
paysim['type'].value_counts().plot('bar')
```



Knowledge check 1

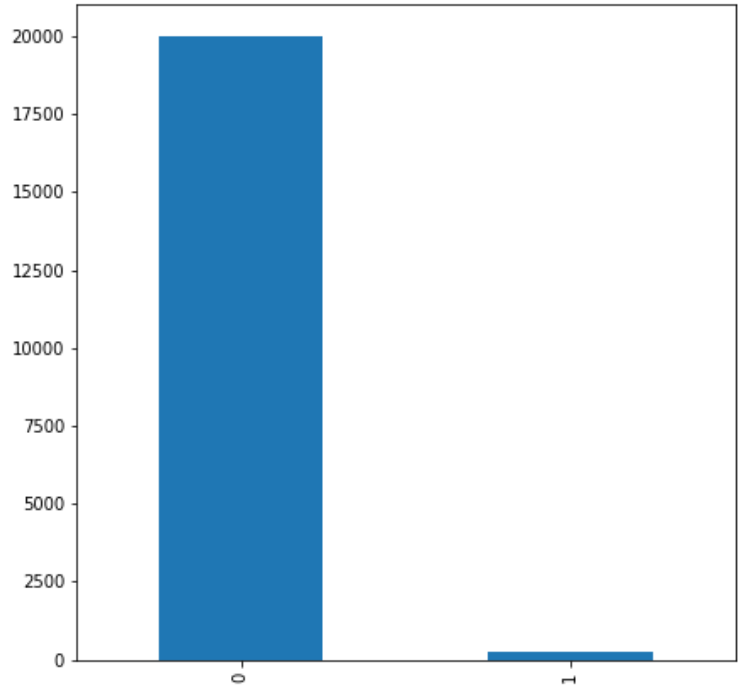
Exercise 1

Module completion checklist

Objective	Complete
Define anomaly concepts and uses	✔
Differentiate between types of anomalies	✔
Create a problem statement for our fraud dataset	✔
Examine why classification techniques are not useful for anomaly detection	
Summarize different techniques for anomaly detection	
Identify SMOTE analysis and its implementation	
Describe the isolation forest algorithm	
Implement isolation forest to detect credit fraud	

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



Prepare the dataset for modeling

- 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 today

```
# 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 `type` column to 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')
```

Logistic regression

- We will build a logistic regression model 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)
logreg = LogisticRegression()
logreg.fit(X_train, y_train)
```

```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='warn', n_jobs=None, penalty='l2',
                   random_state=None, solver='warn', tol=0.0001, verbose=0,
                   warm_start=False)
```

Predict the target

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

```
log_y_train_pred = logreg.predict(X_train)
log_y_test_pred = logreg.predict(X_test)
log_accuracy = metrics.accuracy_score(y_test, log_y_test_pred)
print("Accuracy of test data:\t", log_accuracy)
```

```
Accuracy of test data:   0.9874711506758984
```

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

```
0.9051135388236348
```

- We got 99% accuracy
- Do you think it is a good model for our fraud detection?

Confusion matrix of training data

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

```
Confusion Matrix - Training Dataset
```

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

```
Predicted      0    1    All
True
0          13798  203  14001
1             20  133    153
All        13818  336  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, log_y_test_pred,
rownames = ['True'], colnames = ['Predicted'],
margins = True))
```

```
Predicted     0    1   All
True
0          5935   64  5999
1            12   55    67
All        5947  119  6066
```

- Out of 67 fraud transactions, 55 are predicted as fraud

```
print('Percentage of accurate fraud cases is',
55/67)
```

```
Percentage of accurate fraud cases is
0.8208955223880597
```

- 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 - recap

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 - recap

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, log_y_test_pred).ravel()

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

```
0.9893315552592099
```

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

```
0.8208955223880597
```

- 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(['Logistic_regression_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  Logistic_regression_baseline  0.820896  0.989332
```

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

Module completion checklist

Objective	Complete
Define anomaly concepts and uses	✔
Differentiate between types of anomalies	✔
Create a problem statement for our fraud dataset	✔
Examine why classification techniques are not useful for anomaly detection	✔
Summarize different techniques for anomaly detection	
Identify SMOTE analysis and its implementation	
Describe the isolation forest algorithm	
Implement isolation forest to detect credit fraud	

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 datapoints
  - We need to convert this imbalanced data to balanced class data
  - This can make our classification models perform better
- **Validation metric**
  - We already saw 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

Anomaly detection - supervised methods

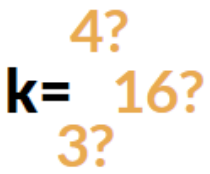
- Aside from conventional classification models, there are special statistical models to detect anomalies
- **Gaussian model**
  - This checks the distribution of the data and models any data point that falls outside the normal distribution
- **Isolation forest**
  - A tree-based approach, which we will learn more about later

Density based anomaly detection

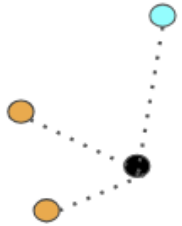
- Density based algorithms are mainly based on distance measure
- The general assumption is regular data points occur around a dense region and abnormalities are far away
- There are two algorithms for density based detection
- **K nearest neighbors**
  - It is based on the fact that regular datapoints are closer to each other than the anomalous data points
  - Any distance measure like Euclidean can be used to find the distance between the data points
- **Local outlier factor**
  - This method is based on relative density of data, which is called “reachability distance”

kNN - recap

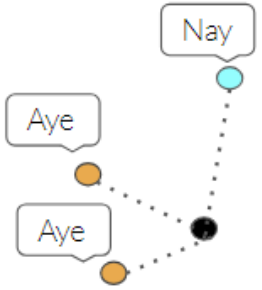
Step 1: Select K



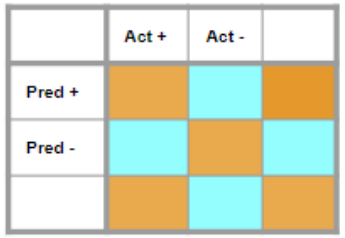
Step 2: Measure Distance



Step 3: Majority Vote



Step 4: Check Performance Metrics



Step 5: Find Best K

Clustering based anomaly detection

- Clustering is one of the most popular concepts in the area of anomaly detection
- It is based on the assumption that data points that are similar tend to belong to similar groups or cluster determined by the local centroid
- k means is the widely used clustering algorithm
- But DBScan is most commonly used for anomaly detection

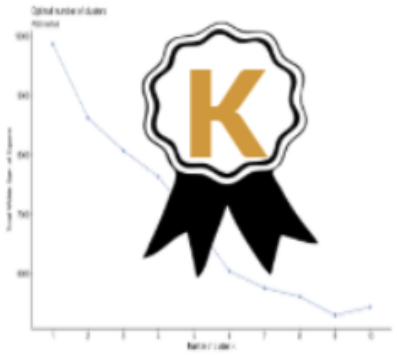
K-means clustering - recap

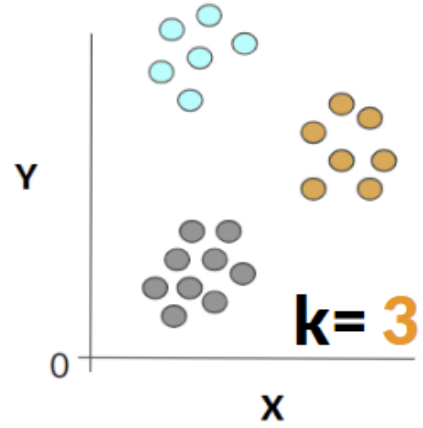
Step 1:
Find optimal k

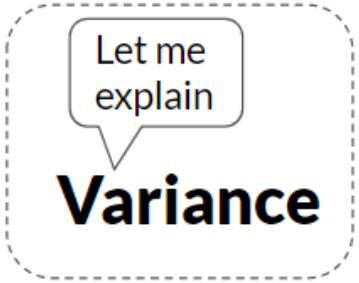
Step 2:
Use optimal k to cluster

Step 3:
Caculate metrics

Step 4:
Inspect clusters







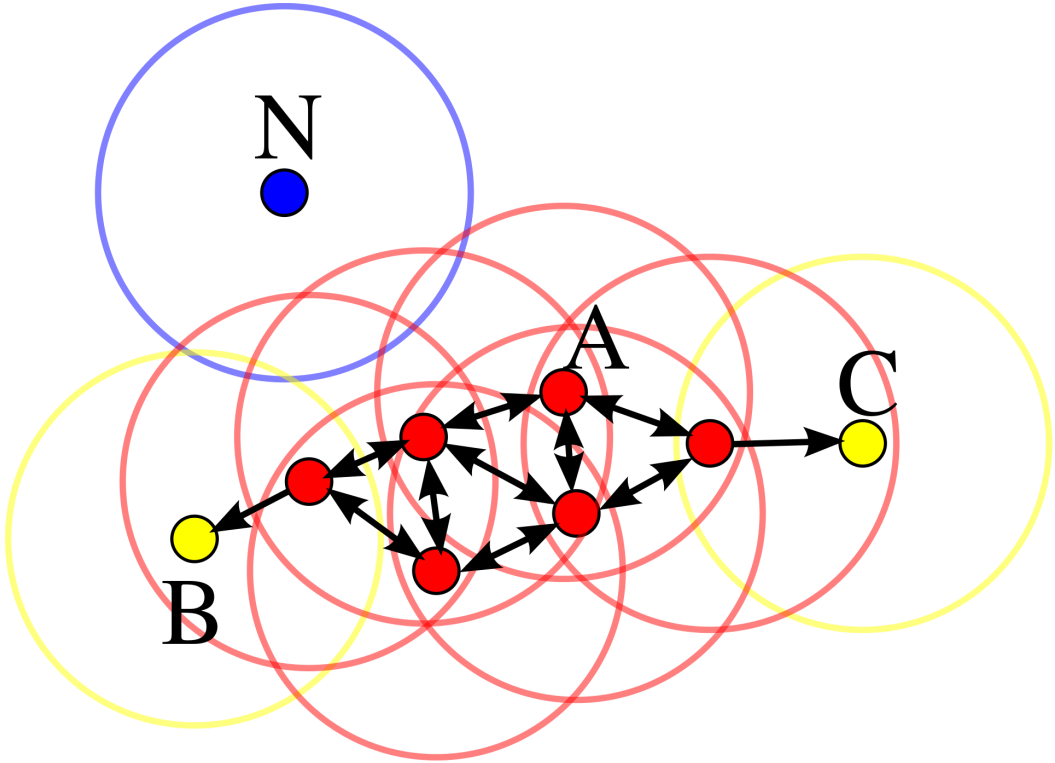
DBSCAN algorithm - recap

- The algorithm can be summarized into the following steps:

1. Pick a random point
2. Compute its neighborhood to determine if it is a core point or an outlier
3. If it's a core point, expand the cluster by adding points directly in reach
4. Add all density-reachable points by jumping neighborhoods of the points assigned to the cluster
5. If an outlier is added to a cluster, reassign it as a border point
6. Repeat the same steps until all data points are assigned to a cluster or labeled an outlier

DBSCAN algorithm - recap

- `MinPts` is set to `4` in the diagram
- The red points are **core points which form a single cluster** as they are all **reachable from one another**
- Yellow points `B` and `C` are not core points, but are reachable from them
- Hence, they belong to the cluster
- Blue point `N` is an outlier as it is neither a core point, nor reachable from them



Support vector machine based anomaly detection

- There is a special SVM algorithm for anomaly detection called as **one class SVM**
- This one class SVM is trained only on normal data
- The algorithm learns a soft boundary in order to cluster the normal data instances using training data
- When it sees the testing instance, it tunes itself to identify the abnormalities that fall outside the learned region

Anomaly detection techniques for today

- We saw many anomaly detection techniques
- Depending upon our use case, we should choose any technique useful for the data based on domain expertise
- Today for our fraud dataset, we will see how to:
  - Handle imbalanced data and using classification model
  - Build an isolation forest

Knowledge check 2

Exercise 2

Module completion checklist

Objective	Complete
Define anomaly concepts and uses	✔
Differentiate between types of anomalies	✔
Create a problem statement for our fraud dataset	✔
Examine why classification techniques are not useful for anomaly detection	✔
Summarize different techniques for anomaly detection	✔
Identify SMOTE analysis and its implementation	
Describe the isolation forest algorithm	
Implement isolation forest to detect credit fraud	

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
  - 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_sample(X_train, y_train)
```

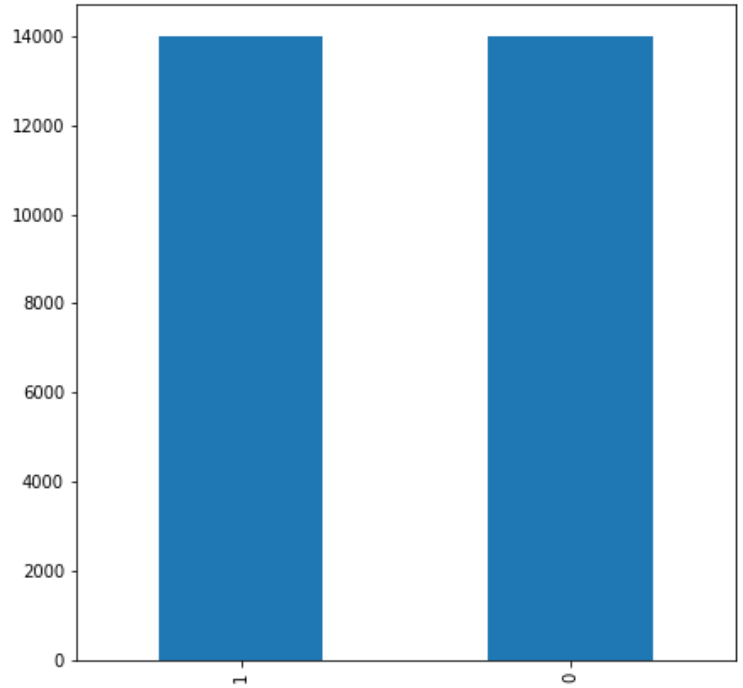
```
# Shape of X_train.
print(X_train.shape)
```

```
(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.
logreg.fit(X_train_new, y_train_new)

# Prediction for training data.
```

```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='warn', n_jobs=None, penalty='l2',
                   random_state=None, solver='warn', tol=0.0001, verbose=0,
                   warm_start=False)
```

```
train_pred_sm = logreg.predict(X_train_new)

# Prediction for the test data.
test_pred_sm = logreg.predict(X_test)
train_pred_sm = logreg.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          13485    516  14001
1            163  13838  14001
All        13648  14354  28002
```

- Out of 14,001 fraud transactions, 13,838 are predicted as fraud

```
print('Percentage of accurate fraud cases: ', 13838/14001)
```

```
Percentage of accurate fraud cases:  0.9883579744303979
```

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          5770  229  5999
1             1   66    67
All        5771  295  6066
```

- Out of 67 fraud transactions, 66 are predicted as fraud

```
print('Percentage of accurate fraud cases ',
66/67)
```

```
Percentage of accurate fraud cases
0.9850746268656716
```

- The SMOTE approach has improved our predictions from 0.82 on test data to 0.98 when compared to our previous logistic regression model which gave us 0.82
- 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)
```

```
0.9618269711618603
```

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

```
0.9850746268656716
```

```
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	Logistic_regression_baseline	0.820896	0.989332
1	SMOTE	0.985075	0.961827

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

Knowledge check 3

Exercise 3

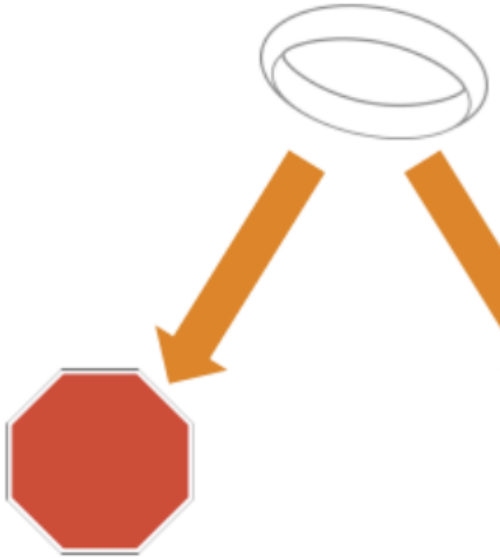
Module completion checklist

Objective	Complete
Define anomaly concepts and uses	✔
Differentiate between types of anomalies	✔
Create a problem statement for our fraud dataset	✔
Examine why classification techniques are not useful for anomaly detection	✔
Summarize different techniques for anomaly detection	✔
Identify SMOTE analysis and its implementation	✔
Describe the isolation forest algorithm	
Implement isolation forest to detect credit fraud	

Decision trees - recap

- Isolation forests are built on the basis of decision trees
- In these trees, partitions are created by first randomly selecting a feature and then selecting a random split value between the minimum and maximum value of the selected feature

Growing decision trees steps



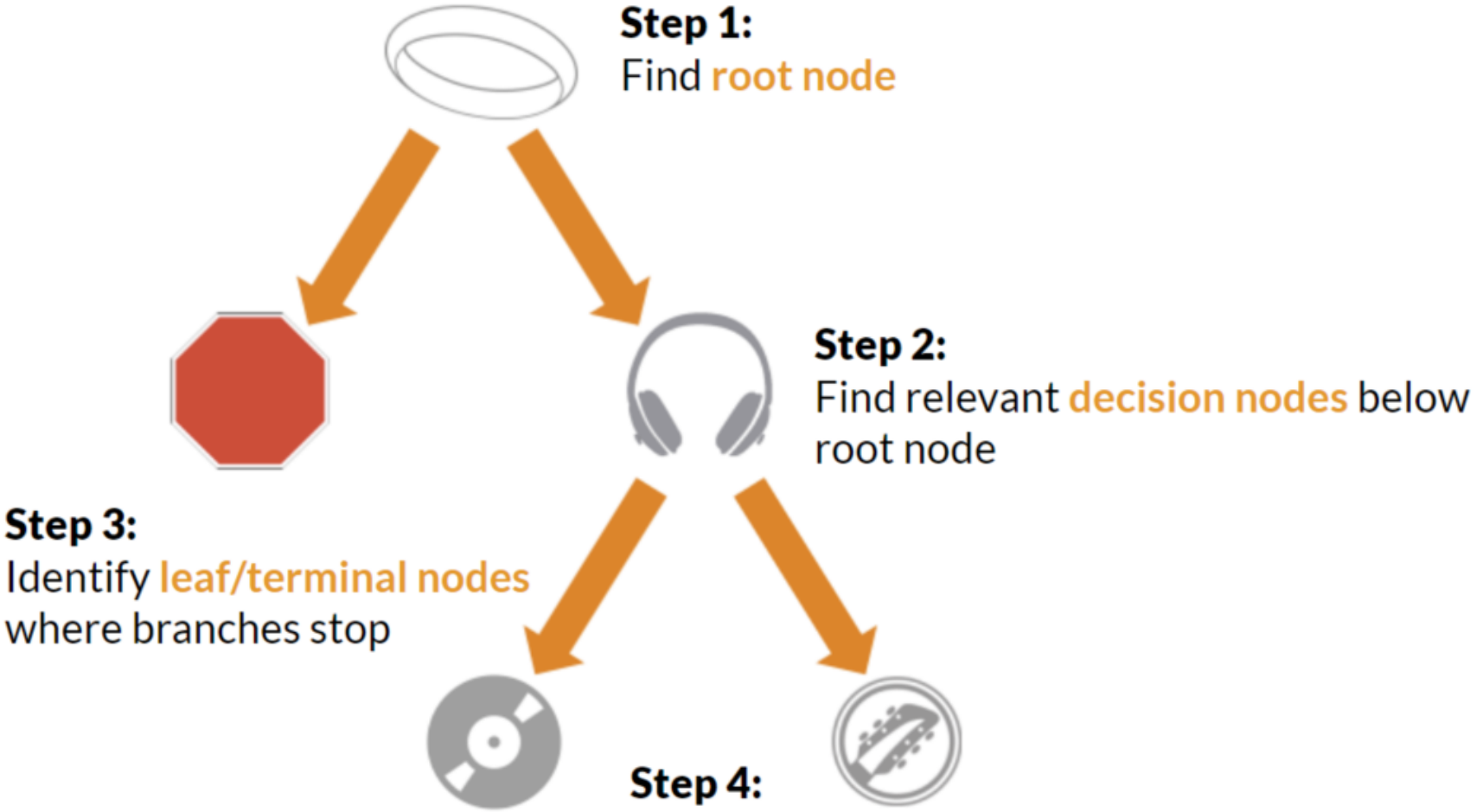
Step 1:
Ask the question with the most amount of information

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

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

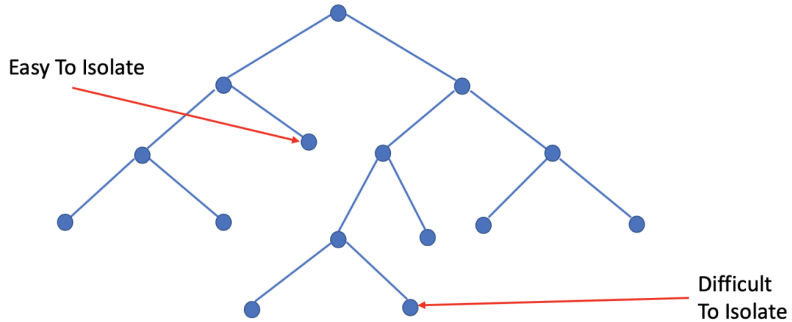
Step 4:
Repeat steps 2 and 3 for each question branch

Growing decision trees with vocabulary



Isolation forest

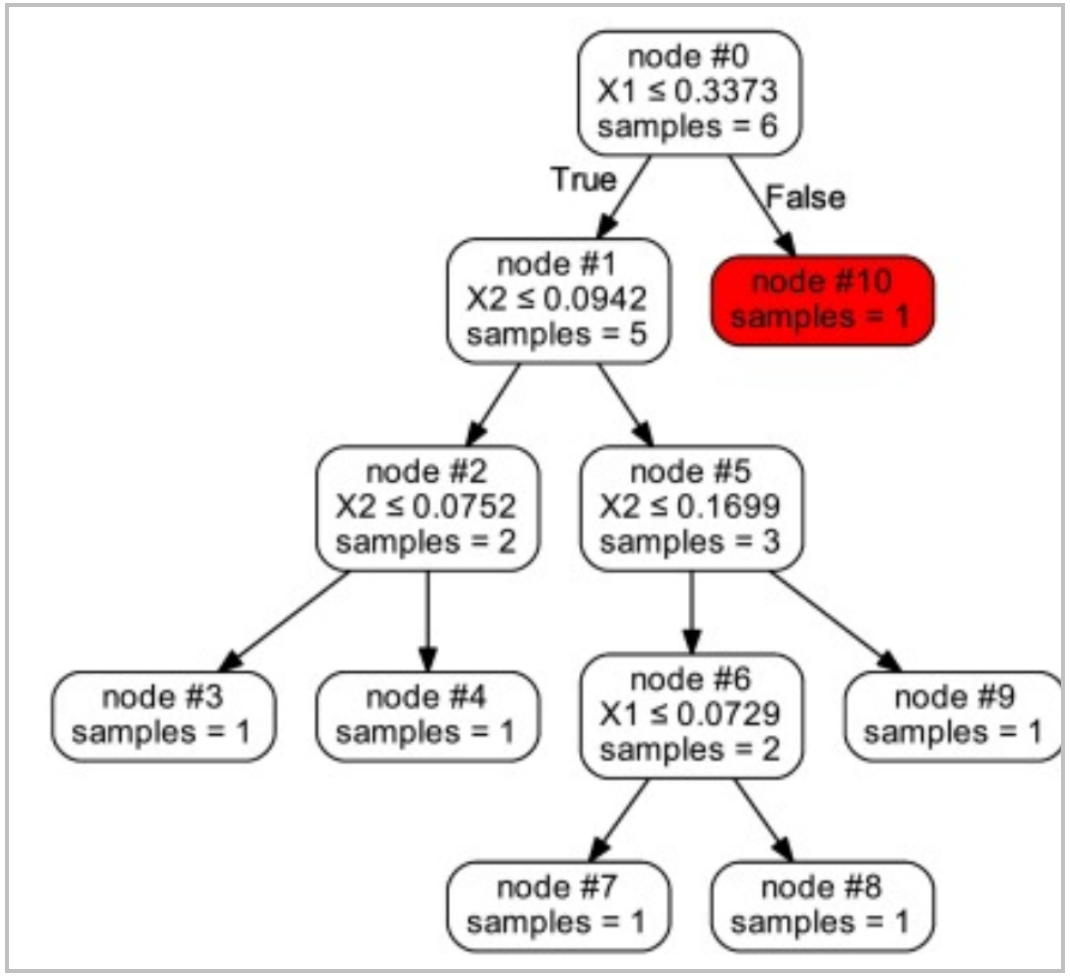
- We know that outliers are less frequent and differ from regular observations
- This means they will also be identified faster and closer to the root during the partitioning at each feature node
- An isolation forest algorithm calculates an anomaly score for each test observation which we want to classify based on the path length
- Path length is the number of nodes the observation travels down the decision tree
- Based on the anomaly score, it is classified either as an inlier (1) or outlier (-1)



Working of isolation forest

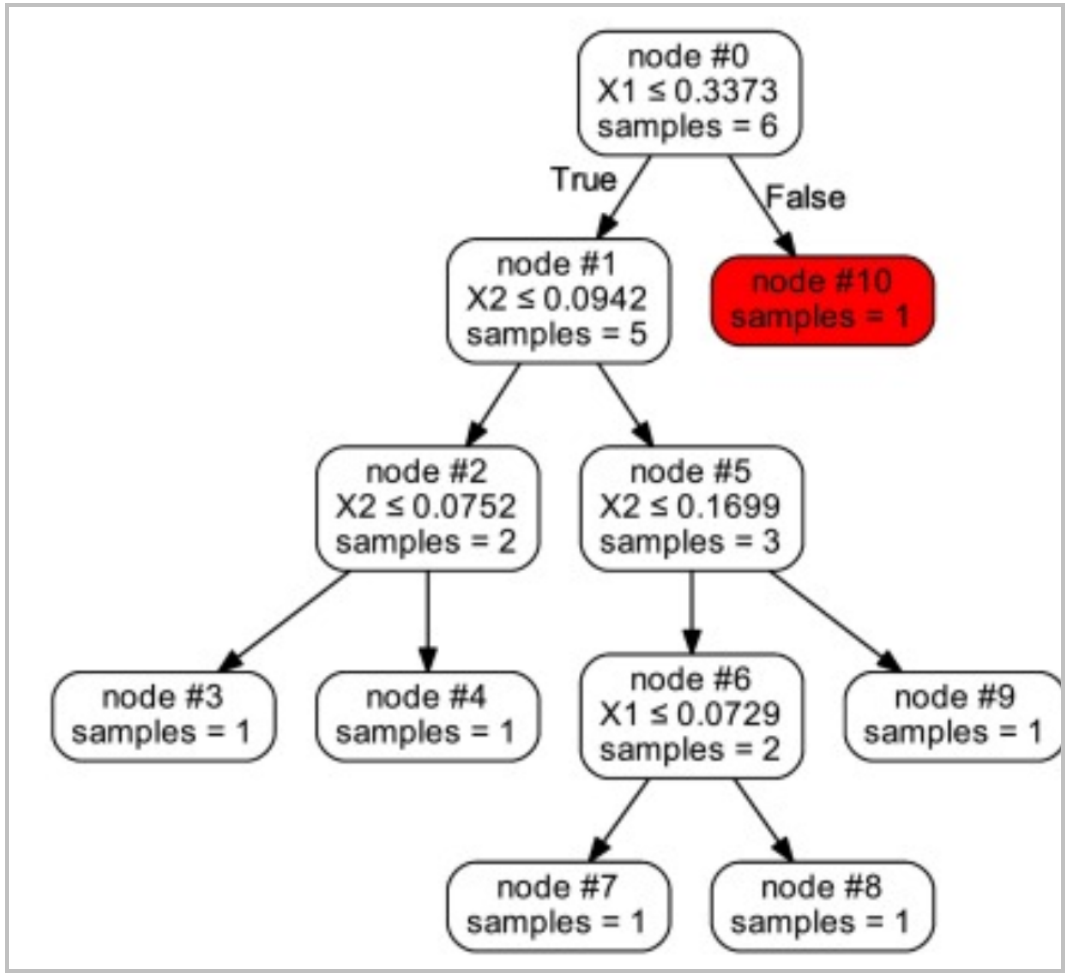
- In the isolation forest, the model is built only on regular observations
- When we get a new test data point, it travels through each node and gets classified as either
  - inlier or normal observation (+1)
  - outlier or anomaly (-1)
- Like random forest, isolation forest has multiple decision trees where the results are aggregated
- Let's say we have a new observation `a`, which is an outlier
- As the new observation travels down the tree, we note that none of its features have the same range as the regular observation
- The features of the new observation are very different from the tree model
- Hence, the new observation is classified as an anomaly and assigned label `-1`

Working of isolation forest with example 1



- Let's say we built our tree model with regular observation as shown
- We have a new observation which has a variable(X1) value greater than a threshold (0.3373), then it gets classified as an anomaly because all the observations in the tree model has X1 <= threshold (0.3373)

Working of isolation forest with example 2



- Let's say we have another new observation which is also an anomaly but its X1 <= threshold (0.3373)
- But its other variable (X2) is greater than threshold2 (0.0942)
- That observation gets classified as outlier at the second node (node #1) because all the observations within the tree model data have values with X2 <= threshold2 (0.0942)

Isolation forest in python

- Our SciKit library has a package for isolation forest
- Read more on it ***here***

sklearn.ensemble.IsolationForest

class sklearn.ensemble. **IsolationForest** (*n_estimators=100, max_samples='auto', contamination='legacy', max_features=1.0, bootstrap=False, n_jobs=None, behaviour='old', random_state=None, verbose=0, warm_start=False*) [source]

Isolation Forest Algorithm

Return the anomaly score of each sample using the IsolationForest algorithm

The IsolationForest 'isolates' observations by randomly selecting a feature and then randomly selecting a split value between the maximum and minimum values of the selected feature.

Since recursive partitioning can be represented by a tree structure, the number of splittings required to isolate a sample is equivalent to the path length from the root node to the terminating node.

This path length, averaged over a forest of such random trees, is a measure of normality and our decision function.

Random partitioning produces noticeably shorter paths for anomalies. Hence, when a forest of random trees collectively produce shorter path lengths for particular samples, they are highly likely to be anomalies.

Isolation forest on fraud dataset

- Our first step is to separate non fraud observations from the whole dataset
- We are going to train the isolation forest model on the regular observation as discussed

```
# Split fraud vs non fraud.
non_fraud = paysim[paysim['isFraud'] == 0]
fraud = paysim[paysim['isFraud'] == 1]

# Drop isFraud column from fraud dataframe.
fraud = fraud.drop(['isFraud'], axis = 1)

non_fraud_X = non_fraud.drop(['isFraud'], axis = 1)
non_fraud_y = non_fraud['isFraud']
```

Train and test split

- Split the non fraud dataset to train and test for modeling

```
non_fraud_X_train, non_fraud_X_test, non_fraud_y_train, non_fraud_y_test =
train_test_split(non_fraud_X, non_fraud_y, test_size = 0.30)

print(non_fraud_X_train.shape)
```

```
(14000, 9)
```

```
print(non_fraud_X_test.shape)
```

```
(6000, 9)
```

Create and fit the model

```
iforest = IsolationForest(random_state = 100)
iforest.fit(non_fraud_X_train)
```

```
IsolationForest(behaviour='old', bootstrap=False, contamination='legacy',
                max_features=1.0, max_samples='auto', n_estimators=100,
                n_jobs=None, random_state=100, verbose=0, warm_start=False)
```

Function to find metric

- We know that an isolation forest classifies the datapoints as -1 and +1 instead of 1 and 0
- Hence, we will create two functions to find our TPR and TNR metric

```
def nonfraud_TNR(Mat):
    Sum = 0
    for i in Mat:
        if(i == 1):
            Sum += 1.0
    return(Sum/len(Mat))
```

- In `non_fraud` dataset, all the observations are not frauds
- True negative rate is found by taking the count of all observations predicted as 1 (non fraud/regular) and divide by total number of observations

```
def fraud_TPR(Mat):
    Sum=0
    for i in Mat:
        if(i==-1):
            Sum+=1.0
    return(Sum/len(Mat))
```

- In `fraud` dataset all the observations are fraud and all are outliers
- True positive rate is found by taking all the count of all observations predicted as -1 (fraud/outlier) and divide by total number of observations

Prediction on non fraud cases

- Let's see how the model performs on non fraud cases and find true negative rate as well

```
nonfraud_test_pred = iforest.predict(non_fraud_X_test)
print(nonfraud_test_pred)
```

```
[1 1 1 ... 1 1 1]
```

```
print(np.unique(nonfraud_test_pred))
```

```
[-1  1]
```

```
non_fraud_eval = nonfraud_TNR(nonfraud_test_pred)
print('Accuracy of non fraud cases:\t', non_fraud_eval)
```

```
Accuracy of non fraud cases:     0.8968333333333334
```

Prediction on fraud cases

- Let's see how the model performs on fraud cases and find true positive rate

```
fraud_pred = iforest.predict(fraud)
print(fraud_pred)
```

```
[-1  1 -1 -1 -1 -1 -1  1 -1  1 -1 -1 -1 -1 -1  1  1  1 -1 -1  1 -1  1  1
  1  1  1 -1  1 -1  1  1 -1 -1  1  1 -1 -1  1 -1 -1  1 -1 -1 -1 -1 -1 -1
 -1 -1  1 -1 -1 -1 -1 -1  1  1 -1 -1 -1 -1  1 -1 -1  1 -1  1 -1  1 -1 -1
  1  1  1 -1 -1 -1 -1  1 -1 -1 -1 -1  1 -1  1  1 -1  1  1 -1  1 -1 -1 -1
  1 -1 -1  1 -1 -1 -1 -1  1 -1 -1  1  1  1 -1 -1 -1 -1 -1  1 -1 -1  1  1
  1 -1 -1  1 -1  1 -1 -1 -1 -1 -1  1  1 -1 -1 -1 -1 -1  1  1  1  1 -1  1
  1 -1  1 -1  1 -1 -1 -1  1 -1 -1 -1  1 -1  1 -1  1 -1 -1  1  1 -1  1 -1
 -1  1  1 -1 -1  1 -1 -1 -1 -1 -1 -1 -1  1  1  1  1 -1 -1  1 -1  1  1 -1
 -1 -1 -1  1  1  1  1 -1  1 -1 -1  1  1 -1  1 -1  1 -1 -1  1  1  1 -1  1
 -1 -1 -1  1]
```

```
print(np.unique(fraud_pred))
```

```
[-1  1]
```

```
fraud_eval = fraud_TPR(fraud_pred)
print('Accuracy of fraud cases:\t', fraud_eval)
```

```
Accuracy of fraud cases:     0.5954545454545455
```

Compare the results

- Append the results to performance dataframe

```
s = pd.Series(['Isolation forest', 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	Logistic_regression_baseline	0.820896	0.989332
1	SMOTE	0.985075	0.961827
2	Isolation forest	0.595455	0.896833

Interpreting the results

- Of the two anomaly models we built today, **SMOTE performs better** than our baseline logistic model and does a good job predicting the fraud cases
- Isolation forest on the other hand performs less well than the baseline model
- **What are some other steps we might try?**
- Tuning the hyperparameters might improve the results
- We could also implement other anomaly detection algorithms we discussed and see which performs better

Knowledge check 4

Exercise 4

Module completion checklist

Objective	Complete
Define anomaly concepts and uses	✔
Differentiate between types of anomalies	✔
Create a problem statement for our fraud dataset	✔
Examine why classification techniques are not useful for anomaly detection	✔
Summarize different techniques for anomaly detection	✔
Identify SMOTE analysis and its implementation	✔
Describe the isolation forest algorithm	✔
Implement isolation forest to detect credit fraud	✔

Workshop!

- Workshops are to be completed in the afternoon either with a dataset for a capstone project or with another dataset of your choosing
- Make sure to annotate and comment your code so that it is easy for others to understand what you are doing
- This is an exploratory exercise to get you comfortable with the content we discussed today
- Today you will:
  - Find a dataset with anomalies
  - Understand the data and build different anomaly models and interpret the results

This completes our module

Congratulations!