```
#outlier detection using isolation forest
```

```
#part a : implementing isolation forest on a randomly generated dataset
```

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

```
#generating the dataset
X=0.3*np.random.randn(100,2)
X.shape
```

```
(100, 2)
```

```
X_train_normal=np.r_[X+2,X-2]#Concanted the first axis ie row
```

```
print(X.shape,X_train_normal.shape)
```

```
(100, 2) (200, 2)
```

```
#generate a dataset for testing
X=0.3*np.random.randn(50,2)
X_test_normal=np.r_[X+2,X-2]
```

```
print(X.shape,X_test_normal.shape)
```

```
(50, 2) (100, 2)
```

```
#generate outliers for training
X_train_outliers=np.random.uniform(low=-4,high=4,size=(20,2))
```

```
X_train_outliers.shape
```

```
(20, 2)
```

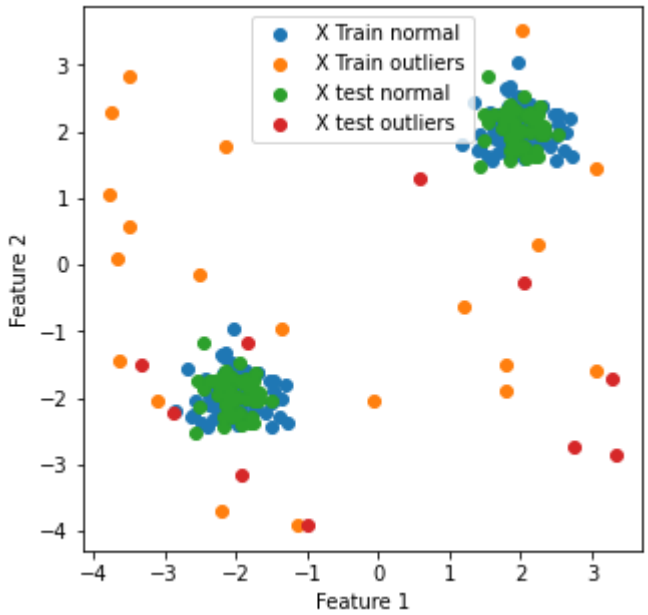
```
#generate outlier for testing
X_test_outliers=np.random.uniform(low=-4,high=4,size=(10,2))
```

```
X_test_outliers.shape
```

```
(10, 2)
```

```
#visualising the data
plt.figure(figsize=(5,5))
plt.scatter(X_train_normal[:,0],X_train_normal[:,1],label="X Train normal")
plt.scatter(X_train_outliers[:,0],X_train_outliers[:,1],label="X Train outliers")
plt.scatter(X_test_normal[:,0],X_test_normal[:,1],label="X test normal")
plt.scatter(X_test_outliers[:,0],X_test_outliers[:,1],label="X test outliers")
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.legend()
```

```
<matplotlib.legend.Legend at 0x7f8fde0b3ad0>
```



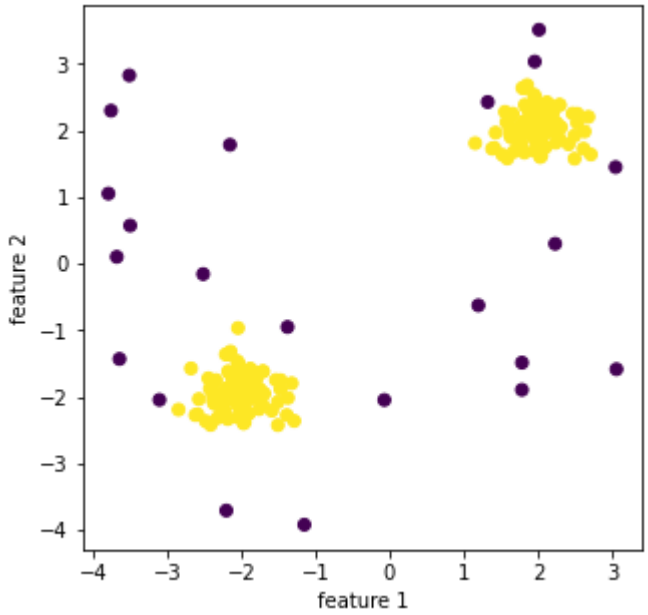

```
X_train=np.append(X_train_normal,X_train_outliers,axis=0)

X_test=np.append(X_test_normal,X_test_outliers,axis=0)


#training with isolation forest
from sklearn.ensemble import IsolationForest
model=IsolationForest(random_state=1,contamination =0.1)



model.fit(X_train)
```

```
IsolationForest(behaviour='deprecated', bootstrap=False, contamination=0.1,
                max_features=1.0, max_samples='auto', n_estimators=100,
                n_jobs=None, random_state=1, verbose=0, warm_start=False)
```

```
#prediction
#since the model is unsupervised well test on train as well as test data
pred_train=model.predict(X_train)
pred_test=model.predict(X_test)
```

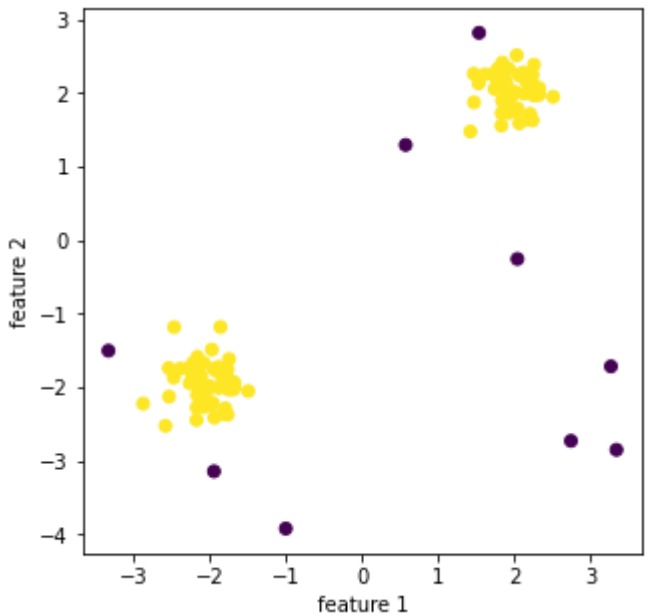
```
#visualise the predictions
plt.figure(figsize=(5,5))
plt.scatter(X_train[:,0],X_train[:,1],c=pred_train)
plt.xlabel('feature 1')
plt.ylabel('feature 2')
```

```
Text(0, 0.5, 'feature 2')
```




```
#visualise the predictions
plt.figure(figsize=(5,5))
plt.scatter(X_test[:,0],X_test[:,1],c=pred_test)
plt.xlabel('feature 1')
plt.ylabel('feature 2')
```

```
Text(0, 0.5, 'feature 2')
```




```
#Part B : Isolation forest on the creditcard dataset


#df=pd.read_csv('/drive/MyDrive/Colab Notebooks/creditcard.csv')
df=pd.read_csv('creditcard.csv')
```

```
df.shape
```

```
(5974, 31)
```

```
normal=df[df['Class']==0]
fraud=df[df['Class']==1]
```

```
print(normal.shape,fraud.shape)
```

```
(5970, 31) (3, 31)
```

```
data=df.sample(frac=0.2,random_state=1)
```

```
data.shape
```

```
(1195, 31)
```

```
normal_frac=data[data['Class']==0]
fraud_frac=data[data['Class']==1]
```

```
normal_frac.shape
```

```
(1193, 31)
```

```
fraud_frac.shape
```

```
(2, 31)
```

```
anomaly_fraction=len(fraud_frac)/float(len(data)) #going to be used for contaminati
```

```
#train the model
model=IsolationForest(n_estimators=100,contamination=anomaly_fraction,random_state=
```

```
model.fit(data[['Class']])
```

```
IsolationForest(behaviour='deprecated', bootstrap=False,
                contamination=0.0016736401673640166, max_features=1.0,
                max_samples='auto', n_estimators=100, n_jobs=None,
                random_state=1, verbose=0, warm_start=False)
```

```
#decision boundary for class 0 or 1
data['scores']=model.decision_function(data[['Class']])
```

```
data['anomaly_score']=model.predict(data[['Class']])
```

```
data.shape
```

```
(1195, 33)
```

```
data[data['anomaly_score']==-1].head() #is in fradulent class
```

	Time	V1	V2	V3	V4	V5	V6	V7	
623	472	-3.043541	-3.157307	1.088463	2.288644	1.359805	-1.064823	0.325574	-0.
4920	4462	-2.303350	1.759247	-0.359745	2.330243	-0.821628	-0.075788	0.562320	-0.

```
#checking the accuracy of the model
anomaly_count=data[data['Class']==1]
anomaly_count=anomaly_count.shape[0]
```

```
accuracy = 100*list(data['anomaly_score']).count(-1)/(anomaly_count)
```

```
anomaly_count
```

```
2
```

```
accuracy
```

```
100.0
```