7. Applications of Data Mining for Fraud Detection- Part 1: Banking and Finance

In this video, we will walk through a comprehensive process of applying machine learning techniques using real-life data. We will train test and evaluate from the following family of algorithms:

1. Supervised
2. Ensemble
3. Unsupervised

Import necessary libraries

In [2]:
```
# Import necessary libraries
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
from sklearn.naive_bayes import GaussianNB
from sklearn.neural_network import MLPClassifier
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, roc_auc_score, roc_curve
from sklearn.model_selection import GridSearchCV
import matplotlib.pyplot as plt

import warnings


import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.metrics import confusion_matrix, accuracy_score, f1_score, roc_auc_score, classification_report
from sklearn.ensemble import BaggingClassifier, RandomForestClassifier, AdaBoostClassifier, GradientBoostingCl
from xgboost import XGBClassifier

from sklearn.ensemble import StackingClassifier, VotingClassifier

# Ignore all warnings
warnings.filterwarnings("ignore")
```

Import the dataset

In [3]:
```
# Load the data
df = pd.read_csv('PS_20174392719_1491204439457_log.csv')
df = df.head(60000)
# Drop the 'type' and 'nameOrig' and 'nameDest' columns as they are not required for our prediction
df = df.drop(['type', 'nameOrig', 'nameDest'], axis=1)
df.head()
```

Out[3]:

	step	amount	oldbalanceOrg	newbalanceOrig	oldbalanceDest	newbalanceDest	isFraud	isFlaggedFraud
0	1	9839.64	170136.0	160296.36	0.0	0.0	0	0
1	1	1864.28	21249.0	19384.72	0.0	0.0	0	0
2	1	181.00	181.0	0.00	0.0	0.0	1	0
3	1	181.00	181.0	0.00	21182.0	0.0	1	0
4	1	11668.14	41554.0	29885.86	0.0	0.0	0	0

Split data into training and testing sets

In [8]:
```
X = df.drop('isFraud', axis=1)
y = df['isFraud']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Normalize the data
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

Supervised Learning Modelling Process

Define hyperparameters to tune for each algorithm

In [9]:
```
lr_params = {'C': [0.1, 1, 10], 'penalty': ['l1', 'l2'], 'solver': ['liblinear']}
dt_params = {'criterion': ['gini', 'entropy'], 'max_depth': [5, 10, 20]}
nb_params = {}
svm_params = {'C': [0.1, 1, 10], 'kernel': ['linear', 'poly', 'rbf', 'sigmoid']}
nn_params = {'hidden_layer_sizes': [(50,), (100,), (50, 50)], 'activation': ['relu', 'logistic'], 'solver': [
```

Train and Test the models

In [10]:
```
# Logistic regression
lr = LogisticRegression(max_iter=1000)
lr_gs = GridSearchCV(lr, lr_params, scoring='roc_auc', cv=5)
lr_gs.fit(X_train, y_train)
lr_preds = lr_gs.predict(X_test)

# Decision trees
dt = DecisionTreeClassifier()
dt_gs = GridSearchCV(dt, dt_params, scoring='roc_auc', cv=5)
dt_gs.fit(X_train, y_train)
dt_preds = dt_gs.predict(X_test)

# Naive Bayes
nb = GaussianNB()
nb_gs = GridSearchCV(nb, nb_params, scoring='roc_auc', cv=5)
nb_gs.fit(X_train, y_train)
nb_preds = nb_gs.predict(X_test)

# SVM
svm = SVC(probability=True,max_iter=1000)
svm_gs = GridSearchCV(svm, svm_params, scoring='roc_auc', cv=5)
svm_gs.fit(X_train, y_train)
svm_preds = svm_gs.predict(X_test)

# Neural Networks
nn = MLPClassifier(max_iter=1000)
nn_gs = GridSearchCV(nn, nn_params, scoring='roc_auc', cv=5)
nn_gs.fit(X_train, y_train)
nn_preds = nn_gs.predict(X_test)
```

Evaluation of algorithms

In [32]:
```
print("Logistic Regression Metrics:")
print("Accuracy:", accuracy_score(y_test, lr_preds))
print("Precision:", precision_score(y_test, lr_preds))
print("Recall:", recall_score(y_test, lr_preds))
print("F1 Score:", f1_score(y_test, lr_preds))
print("AUC Score:", roc_auc_score(y_test, lr_preds))
print("\n")

print("Decision Trees Metrics:")
print("Accuracy:", accuracy_score(y_test, dt_preds))
print("Precision:", precision_score(y_test, dt_preds))
print("Recall:", recall_score(y_test, dt_preds))
print("F1 Score:", f1_score(y_test, dt_preds))
print("AUC Score:", roc_auc_score(y_test, dt_preds))
print("\n")

print("Naive Bayes Metrics:")
print("Accuracy:", accuracy_score(y_test, nb_preds))
print("Precision:", precision_score(y_test, nb_preds))
print("Recall:", recall_score(y_test, nb_preds))
print("F1 Score:", f1_score(y_test, nb_preds))
print("AUC Score:", roc_auc_score(y_test, nb_preds))
print("\n")

print("SVM Metrics:")
print("Accuracy:", accuracy_score(y_test, svm_preds))
print("Precision:", precision_score(y_test, svm_preds))
print("Recall:", recall_score(y_test, svm_preds))
print("F1 Score:", f1_score(y_test, svm_preds))
print("AUC Score:", roc_auc_score(y_test, svm_preds))
print("\n")

print("Neural Networks Metrics:")
print("Accuracy:", accuracy_score(y_test, nn_preds))
print("Precision:", precision_score(y_test, nn_preds))
print("Recall:", recall_score(y_test, nn_preds))
print("F1 Score:", f1_score(y_test, nn_preds))
print("AUC Score:", roc_auc_score(y_test, nn_preds))
print("\n")
```

```
Logistic Regression Metrics:
Accuracy: 0.9982222222222222
Precision: 0.5833333333333334
Recall: 0.20588235294117646
F1 Score: 0.3043478260869565
AUC Score: 0.6028020247395406


Decision Trees Metrics:
Accuracy: 0.998
Precision: 0.0
Recall: 0.0
F1 Score: 0.0
AUC Score: 0.499944339307581


Naive Bayes Metrics:
Accuracy: 0.9813333333333333
Precision: 0.003289473684210526
Recall: 0.029411764705882353
F1 Score: 0.0059171597633136085
AUC Score: 0.5062732874514605


SVM Metrics:
Accuracy: 0.9981666666666666
Precision: 1.0
Recall: 0.029411764705882353
F1 Score: 0.05714285714285715
AUC Score: 0.5147058823529411


Neural Networks Metrics:
Accuracy: 0.9981111111111111
Precision: 0.0
Recall: 0.0
F1 Score: 0.0
AUC Score: 0.5
```

```
C:\Users\AMarkou\Anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1344: UndefinedMetricWarnin
g: Precision is ill-defined and being set to 0.0 due to no predicted samples. Use `zero_division` parameter
to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

Ensemble Learning Modelling Process

Define hyperparameters to tune for each algorithm

In [37]:
```
bag_params = {
    'n_estimators': [10, 50, 100, 200],
    'max_samples': [0.5, 1.0],
    'max_features': [0.5, 1.0],
    'bootstrap': [True, False],
    'bootstrap_features': [True, False]
}


rf_params = {
    'n_estimators': [100, 200, 500],
    'max_depth': [None, 10, 20, 30],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
    'bootstrap': [True, False]
}


ada_params = {
    'n_estimators': [50, 100, 200],
    'learning_rate': [0.001, 0.01, 0.1, 1.0]
}


gb_params = {
    'n_estimators': [100, 200, 500],
    'learning_rate': [0.001, 0.01, 0.1, 1.0],
    'subsample': [0.5, 1.0],
    'max_depth': [3, 5, 10]
}


xgb_params = {
    'n_estimators': [100, 200, 500],
    'learning_rate': [0.001, 0.01, 0.1, 1.0],
    'subsample': [0.5, 1.0],
    'max_depth': [3, 5, 10],
    'colsample_bytree': [0.5, 1.0],
    'gamma': [0, 0.1, 0.2]
```

```
}
```

Train and Test the models

In [39]:
```
# Bagging
bag = BaggingClassifier()
bag_gs = GridSearchCV(bag, bag_params, scoring='roc_auc', cv=5)
bag_gs.fit(X_train, y_train)
bag_preds = bag_gs.predict(X_test)

# Random Forest
rf = RandomForestClassifier()
rf_gs = GridSearchCV(rf, rf_params, scoring='roc_auc', cv=5)
rf_gs.fit(X_train, y_train)
rf_preds = rf_gs.predict(X_test)

# AdaBoost
ada = AdaBoostClassifier()
ada_gs = GridSearchCV(ada, ada_params, scoring='roc_auc', cv=5)
ada_gs.fit(X_train, y_train)
ada_preds = ada_gs.predict(X_test)

# Gradient Boosting
gb = GradientBoostingClassifier()
gb_gs = GridSearchCV(gb, gb_params, scoring='roc_auc', cv=5)
gb_gs.fit(X_train, y_train)
gb_preds = gb_gs.predict(X_test)

# XGBoost
xgb = XGBClassifier()
xgb_gs = GridSearchCV(xgb, xgb_params, scoring='roc_auc', cv=5)
xgb_gs.fit(X_train, y_train)
xgb_preds = xgb_gs.predict(X_test)
```

Evaluation of algorithms

In [40]:
```
print("Bagging Metrics:")
print("Accuracy:", accuracy_score(y_test, bag_preds))
print("Precision:", precision_score(y_test, bag_preds))
print("Recall:", recall_score(y_test, bag_preds))
print("F1 Score:", f1_score(y_test, bag_preds))
print("AUC Score:", roc_auc_score(y_test, bag_preds))
print("\n")

print("Random Forest Metrics:")
print("Accuracy:", accuracy_score(y_test, rf_preds))
print("Precision:", precision_score(y_test, rf_preds))
print("Recall:", recall_score(y_test, rf_preds))
print("F1 Score:", f1_score(y_test, rf_preds))
print("AUC Score:", roc_auc_score(y_test, rf_preds))
print("\n")

print("AdaBoost Metrics:")
print("Accuracy:", accuracy_score(y_test, ada_preds))
print("Precision:", precision_score(y_test, ada_preds))
print("Recall:", recall_score(y_test, ada_preds))
print("F1 Score:", f1_score(y_test, ada_preds))
print("AUC Score:", roc_auc_score(y_test, ada_preds))
print("\n")

print("Gradient Boosting Metrics:")
print("Accuracy:", accuracy_score(y_test, gb_preds))
print("Precision:", precision_score(y_test, gb_preds))
print("Recall:", recall_score(y_test, gb_preds))
print("F1 Score:", f1_score(y_test, gb_preds))
print("AUC Score:", roc_auc_score(y_test, gb_preds))
print("\n")

print("XGBoost Metrics:")
print("Accuracy:", accuracy_score(y_test, xgb_preds))
print("Precision:", precision_score(y_test, xgb_preds))
print("Recall:", recall_score(y_test, xgb_preds))
print("F1 Score:", f1_score(y_test, xgb_preds))
print("AUC Score:", roc_auc_score(y_test, xgb_preds))
print("\n")
```

```
Bagging Metrics:
Accuracy: 0.9981666666666666
Precision: 1.0
Recall: 0.029411764705882353
F1 Score: 0.05714285714285715
AUC Score: 0.5147058823529411


Random Forest Metrics:
Accuracy: 0.9982222222222222
Precision: 1.0
Recall: 0.058823529411764705
F1 Score: 0.1111111111111111
AUC Score: 0.5294117647058824


AdaBoost Metrics:
Accuracy: 0.9981666666666666
Precision: 1.0
Recall: 0.029411764705882353
F1 Score: 0.05714285714285715
AUC Score: 0.5147058823529411


Gradient Boosting Metrics:
Accuracy: 0.9984444444444445
Precision: 0.875
Recall: 0.20588235294117646
F1 Score: 0.33333333333333337
AUC Score: 0.6029133461243786


XGBoost Metrics:
Accuracy: 0.9992777777777778
Precision: 0.9565217391304348
Recall: 0.6470588235294118
F1 Score: 0.7719298245614036
AUC Score: 0.8235015814184964
```

Do the same process for Stacking

In [ ]:
```
#from sklearn.ensemble import StackingClassifier

# Base models
#lr = LogisticRegression()
#dt = DecisionTreeClassifier()
#svm = SVC(probability=True)

# Hyperparameters for the base models
#lr_params = {'logistic_regression__C': [0.001, 0.01, 0.1, 1, 10, 100, 1000]}
#dt_params = {'decision_tree__max_depth': [None, 5, 10, 15, 20]}
#svm_params = {'support_vector_machine__C': [0.001, 0.01, 0.1, 1, 10, 100, 1000],
#              'support_vector_machine__gamma': [1, 0.1, 0.01, 0.001, 0.0001],
#              'support_vector_machine__kernel': ['linear', 'rbf']}

#base_models = [
#    ('logistic_regression', GridSearchCV(lr, lr_params, cv=5)),
#    ('decision_tree', GridSearchCV(dt, dt_params, cv=5)),
#    ('support_vector_machine', GridSearchCV(svm, svm_params, cv=5))
#]

# Meta model
#rf = RandomForestClassifier()

# Hyperparameters for the meta model
#rf_params = {'randomforestclassifier__n_estimators': [50, 100, 150, 200],
#             'randomforestclassifier__max_depth': [None, 5, 10, 15, 20]}

# Define the stacking classifier
#stacking_model = StackingClassifier(estimators=base_models, final_estimator=GridSearchCV(rf, rf_params, cv=5)

# Fit the model to our training data
#stacking_model.fit(X_train, y_train)

# Predict the labels of the test set
#stack_preds = stacking_model.predict(X_test)

#print("Stacking Model Metrics:")
#print("Accuracy:", accuracy_score(y_test, stack_preds))
#print("Precision:", precision_score(y_test, stack_preds))
#print("Recall:", recall_score(y_test, stack_preds))
#print("F1 Score:", f1_score(y_test, stack_preds))
#print("AUC Score:", roc_auc_score(y_test, stack_preds))
```

```
#print("\n")
```

Unsupervised Learning Process

In [16]:
```
# Apply K-Means clustering
kmeans = KMeans(n_clusters=2, random_state=42)
kmeans.fit(X_train)

# Predict the clusters for test data
clusters = kmeans.predict(X_test)

# Append clusters and actual fraud labels to the test dataset
test_df = pd.DataFrame(X_test, columns=X.columns)
test_df['cluster'] = clusters
test_df['isFraud'] = y_test.values

# Calculate fraud rates for each cluster
cluster_fraud_rates = test_df.groupby('cluster')['isFraud'].mean()

print("Fraud Rate for each cluster:\n")
print(cluster_fraud_rates)

print("Silhouette Score: ", silhouette_score(X_test, clusters))
```

```
Fraud Rate for each cluster:

cluster
0    0.002113
1    0.000000
Name: isFraud, dtype: float64
Silhouette Score:  0.654287223507193
```

In [ ]: