7. Applications of Data Mining for Fraud Detection - Part 3: Healthcare

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 [5]:
```
# 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 [6]:
```
# Load the data
df = pd.read_csv('healthcare.csv')

df.head()
```

Out[6]:

	Provider	PotentialFraud	InscClaimAmtReimbursed	DeductibleAmtPaid	AdmitForDays	NoOfMonths_PartACov	NoOfMonths_PartBCo
0	PRV51001	0	104640	5340.0	30.0	300	30
1	PRV51003	1	605670	66286.0	382.0	1560	156
2	PRV51004	0	52170	310.0	0.0	1768	178
3	PRV51005	1	280910	3700.0	0.0	13872	1391
4	PRV51007	0	33710	3264.0	19.0	852	85

5 rows × 28 columns

Split data into training and testing sets

In [3]:
```
X = df.drop(axis=1,columns=['Provider','PotentialFraud'])
y = df['PotentialFraud']

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 [8]:
```
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 [9]:
```
# 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 [10]:
```
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: 0.933456561922366
Precision: 0.8072289156626506
Recall: 0.42138364779874216
F1 Score: 0.553719008264463
AUC Score: 0.7052273430250542


Decision Trees Metrics:
Accuracy: 0.9254467036352434
Precision: 0.6666666666666666
Recall: 0.4779874213836478
F1 Score: 0.5567765567765568
AUC Score: 0.7260155686153211


Naive Bayes Metrics:
Accuracy: 0.9038817005545287
Precision: 0.5098039215686274
Recall: 0.49056603773584906
F1 Score: 0.5
AUC Score: 0.7196682647695638


SVM Metrics:
Accuracy: 0.929143561306223
Precision: 0.8055555555555556
Recall: 0.36477987421383645
F1 Score: 0.5021645021645021
AUC Score: 0.6776085163418909


Neural Networks Metrics:
Accuracy: 0.933456561922366
Precision: 0.7684210526315789
Recall: 0.4591194968553459
F1 Score: 0.5748031496062993
AUC Score: 0.7220460872254872
```

Ensemble Learning Modelling Process

Define hyperparameters to tune for each algorithm

In [14]:
```
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 [12]:
```
# 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 [13]:
```
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: 0.9297597042513863
Precision: 0.7848101265822784
Recall: 0.389937106918239
F1 Score: 0.5210084033613445
AUC Score: 0.6891625425301577


Random Forest Metrics:
Accuracy: 0.9285274183610598
Precision: 0.7362637362637363
Recall: 0.42138364779874216
F1 Score: 0.536
AUC Score: 0.7024951025878957


AdaBoost Metrics:
Accuracy: 0.9322242760320394
Precision: 0.8266666666666667
Recall: 0.389937106918239
F1 Score: 0.5299145299145299
AUC Score: 0.6905286627487369


Gradient Boosting Metrics:
Accuracy: 0.9303758471965496
Precision: 0.7875
Recall: 0.39622641509433965
F1 Score: 0.5271966527196653
AUC Score: 0.692307196618208


XGBoost Metrics:
Accuracy: 0.9322242760320394
Precision: 0.810126582278481
Recall: 0.4025157232704403
F1 Score: 0.5378151260504203
AUC Score: 0.6961349108155479
```

Unsupervised Learning Process

In [4]:
```
# 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    0.088584
1    0.850000
Name: isFraud, dtype: float64
Silhouette Score:  0.8953841660292245
```