Jetwyn Wilson

Extra Credit Link: https://youtu.be/AYfcrX7Ya2U

```
from ucimlrepo import fetch_ucirepo

cervical_cancer_risk_factors = fetch_ucirepo(id=383)

X = cervical_cancer_risk_factors.data.features
y = cervical_cancer_risk_factors.data.targets

print(cervical_cancer_risk_factors.metadata)

print(cervical_cancer_risk_factors.variables)
```

```
from ucimlrepo import fetch_ucirepo
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, LabelEncoder
from imblearn.over_sampling import SMOTE
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, accuracy_score
cervical_cancer_risk_factors = fetch_ucirepo(id=383)

X = cervical_cancer_risk_factors.data.features
y = cervical_cancer_risk_factors.data.targets


data = pd.concat([X, y], axis=1)

print("Missing values in the dataset:")
print(data.isnull().sum())


y = X['Biopsy']
X = X.drop(columns=['Biopsy'])

columns_to_drop = X.columns[X.isnull().mean() > 0.5]
X.drop(columns=columns_to_drop, inplace=True)
```

```
X = X.fillna(X.median())

categorical_columns = X.select_dtypes(include=['object']).columns
for col in categorical_columns:
    X[col] = X[col].fillna(X[col].mode()[0])

label_encoder = LabelEncoder()
for col in ['Smokes', 'Hormonal Contraceptives', 'IUD']:
    if col in X.columns:
        X[col] = label_encoder.fit_transform(X[col])

scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

X_train, X_test, y_train, y_test = train_test_split(X_scaled, y,
test_size=0.2, random_state=42)

smote = SMOTE(random_state=42)
X_train_smote, y_train_smote = smote.fit_resample(X_train, y_train)


print(f"Training data shape (after SMOTE): {X_train_smote.shape}")
print(f"Testing data shape: {X_test.shape}")


model = RandomForestClassifier(random_state=42)
model.fit(X_train_smote, y_train_smote)


y_pred = model.predict(X_test)


print("\nModel Evaluation:")
print(f"Accuracy: {accuracy_score(y_test, y_pred)}")
print("Classification Report:")
print(classification_report(y_test, y_pred))
```

```
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, accuracy_score

param_grid = {
    'n_estimators': [100, 200, 300],
    'max_depth': [None, 3, 5, 7],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
    'bootstrap': [True, False],
    'class_weight': [None, 'balanced']
}


grid_search_rf = GridSearchCV(
    RandomForestClassifier(random_state=42),
    param_grid=param_grid,
    scoring='accuracy',
    cv=3,
    n_jobs=-1
)

grid_search_rf.fit(X_train_smote, y_train_smote)

print("Best hyperparameters for Random Forest:")
print(grid_search_rf.best_params_)

best_rf_model = grid_search_rf.best_estimator_
y_pred_rf = best_rf_model.predict(X_test)

print("Random Forest with Best Hyperparameters Evaluation:")
print(f"Accuracy: {accuracy_score(y_test, y_pred_rf)}")
print("Classification Report:")
print(classification_report(y_test, y_pred_rf))


from xgboost import XGBClassifier
from sklearn.metrics import classification_report, accuracy_score
```

```
class_0_count = sum(y_train_smote == 0)
class_1_count = sum(y_train_smote == 1)


xgb_model = XGBClassifier(scale_pos_weight=class_0_count/class_1_count,
random_state=42)


xgb_model.fit(X_train_smote, y_train_smote)


y_pred_xgb = xgb_model.predict(X_test)


print("XGBoost Model Evaluation:")
print(f"Accuracy: {accuracy_score(y_test, y_pred_xgb)}")
print("Classification Report:")
print(classification_report(y_test, y_pred_xgb))

from sklearn.model_selection import GridSearchCV

param_grid = {
    'learning_rate': [0.01, 0.05, 0.1],
    'max_depth': [3, 5, 7],
    'n_estimators': [100, 200, 300],
    'subsample': [0.8, 0.9, 1.0],
}


grid_search = GridSearchCV(XGBClassifier(random_state=42), param_grid,
scoring='accuracy', cv=3)
grid_search.fit(X_train_smote, y_train_smote)


print("Best parameters found: ", grid_search.best_params_)

xgb_model_best = XGBClassifier(
    learning_rate=0.05,
    max_depth=3,
```

```
    n_estimators=200,
    subsample=1.0,
    scale_pos_weight=class_0_count/class_1_count,
    random_state=42
)


xgb_model_best.fit(X_train_smote, y_train_smote)

y_pred_xgb_best = xgb_model_best.predict(X_test)


print("XGBoost Model Evaluation with Best Parameters:")
print(f"Accuracy: {accuracy_score(y_test, y_pred_xgb_best)}")
print("Classification Report:")
print(classification_report(y_test, y_pred_xgb_best))

y_prob = xgb_model_best.predict_proba(X_test)[:, 1]
threshold = 0.3
y_pred_threshold = (y_prob > threshold).astype(int)

print("XGBoost Model with Adjusted Threshold Evaluation:")
print(f"Accuracy: {accuracy_score(y_test, y_pred_threshold)}")
print("Classification Report:")
print(classification_report(y_test, y_pred_threshold))



from imblearn.over_sampling import ADASYN
from sklearn.metrics import classification_report, accuracy_score

adasyn = ADASYN(sampling_strategy=0.7, random_state=42)

X_train_adasyn, y_train_adasyn = adasyn.fit_resample(X_train, y_train)

print("Class distribution after ADASYN:")
print(pd.Series(y_train_adasyn).value_counts())

from xgboost import XGBClassifier
```

```
xgb_model_adasyn = XGBClassifier(
    learning_rate=0.05,
    max_depth=3,
    n_estimators=200,
    subsample=1.0,
    scale_pos_weight=class_0_count/class_1_count,
    random_state=42
)


xgb_model_adasyn.fit(X_train_adasyn, y_train_adasyn)

y_prob_adasyn = xgb_model_adasyn.predict_proba(X_test)[:, 1]
threshold = 0.3
y_pred_threshold_adasyn = (y_prob_adasyn > threshold).astype(int)

print("XGBoost Model with ADASYN and Further Adjusted Threshold
Evaluation:")
print(f"Accuracy: {accuracy_score(y_test, y_pred_threshold_adasyn)}")
print("Classification Report:")
print(classification_report(y_test, y_pred_threshold_adasyn))

import matplotlib.pyplot as plt
import numpy as np
models = ['XGBoost', 'Random Forest', 'ADASYN + XGBoost']
precision_class_1 = [0.53, 0.60, 0.56]
recall_class_1 = [0.73, 0.55, 0.82]
f1_class_1 = [0.62, 0.57, 0.67]
fig, ax = plt.subplots(figsize=(10, 6))
# 1 (cancer)
ax.plot(models, precision_class_1, label='Precision (Class 1)',
color='r', marker='o')
ax.plot(models, recall_class_1, label='Recall (Class 1)', color='y',
marker='o')
ax.plot(models, f1_class_1, label='F1-score (Class 1)', color='m',
marker='o')
ax.set_xlabel('Models')
ax.set_ylabel('Scores')
```

```
ax.set_title('Performance Comparison of Models for Cancer Prediction
(Class 1)')
ax.legend()
plt.tight_layout()
plt.show()
```