project/

│

├── data/

│ └── data.csv

│

├── models/

│ ├── svm\_model.py

│ ├── dt\_model.py

│ ├── blda\_model.py

│ ├── knn\_model.py

│ └── xgb\_model.py

│

├── preprocessing/

│ └── preprocess.py

│

├── main.py

└── requirements.txt

from sklearn.svm

import SVC

import optuna

import torch

import scipy.io

import os

import data

def svm\_objective(trial, X\_train, y\_train):

C = trial.suggest\_loguniform('C', 1e-3, 1e3)

kernel = trial.suggest\_categorical('kernel', ['linear', 'poly', 'rbf', 'sigmoid'])

model = SVC(C=C, kernel=kernel)

return cross\_val\_score(model, X\_train, y\_train, n\_jobs=-1, cv=3).mean()

def get\_best\_svm\_model(X\_train, y\_train):

study = optuna.create\_study(direction='maximize')

study.optimize(lambda trial: svm\_objective(trial, X\_train, y\_train), n\_trials=50)

best\_params = study.best\_trial.params

best\_model = SVC(\*\*best\_params)

best\_model.fit(X\_train, y\_train)

return best\_model, study.best\_trial

from sklearn.svm

import SVC

import optuna

import data

def svm\_objective(trial, X\_train, y\_train):

C = trial.suggest\_loguniform('C', 1e-3, 1e3)

kernel = trial.suggest\_categorical('kernel', ['linear', 'poly', 'rbf', 'sigmoid'])

model = SVC(C=C, kernel=kernel)

return cross\_val\_score(model, X\_train, y\_train, n\_jobs=-1, cv=3).mean()

def get\_best\_svm\_model(X\_train, y\_train):

study = optuna.create\_study(direction='maximize')

study.optimize(lambda trial: svm\_objective(trial, X\_train, y\_train), n\_trials=50)

best\_params = study.best\_trial.params

best\_model = SVC(\*\*best\_params)

best\_model.fit(X\_train, y\_train)

return best\_model, study.best\_trial

from sklearn.tree import DecisionTreeClassifier

import optuna

import data

def dt\_objective(trial, X\_train, y\_train):

max\_depth = trial.suggest\_int('max\_depth', 1, 32)

model = DecisionTreeClassifier(max\_depth=max\_depth)

return cross\_val\_score(model, X\_train, y\_train, n\_jobs=-1, cv=3).mean()

def get\_best\_dt\_model(X\_train, y\_train):

study = optuna.create\_study(direction='maximize')

study.optimize(lambda trial: dt\_objective(trial, X\_train, y\_train), n\_trials=50)

best\_params = study.best\_trial.params

best\_model = DecisionTreeClassifier(\*\*best\_params)

best\_model.fit(X\_train, y\_train)

return best\_model, study.best\_trial

from sklearn.discriminant\_analysis import LinearDiscriminantAnalysis

def get\_best\_blda\_model(X\_train, y\_train):

model = LinearDiscriminantAnalysis()

model.fit(X\_train, y\_train)

return model

from sklearn.neighbors import KNeighborsClassifier

import optuna

import data

def knn\_objective(trial, X\_train, y\_train):

n\_neighbors = trial.suggest\_int('n\_neighbors', 1, 50)

model = KNeighborsClassifier(n\_neighbors=n\_neighbors)

return cross\_val\_score(model, X\_train, y\_train, n\_jobs=-1, cv=3).mean()

def get\_best\_knn\_model(X\_train, y\_train):

study = optuna.create\_study(direction='maximize')

study.optimize(lambda trial: knn\_objective(trial, X\_train, y\_train), n\_trials=50)

best\_params = study.best\_trial.params

best\_model = KNeighborsClassifier(\*\*best\_params)

best\_model.fit(X\_train, y\_train)

return best\_model, study.best\_trial

from xgboost import XGBClassifier

import optuna

import data

def xgb\_objective(trial, X\_train, y\_train):

params = {

'n\_estimators': trial.suggest\_int('n\_estimators', 50, 500),

'learning\_rate': trial.suggest\_loguniform('learning\_rate', 0.01, 0.3),

'max\_depth': trial.suggest\_int('max\_depth', 3, 10),

'eval\_metric': 'mlogloss'

}

model = XGBClassifier(\*\*params, use\_label\_encoder=False)

return cross\_val\_score(model, X\_train, y\_train, n\_jobs=-1, cv=3).mean()

def get\_best\_xgb\_model(X\_train, y\_train):

study = optuna.create\_study(direction='maximize')

study.optimize(lambda trial: xgb\_objective(trial, X\_train, y\_train), n\_trials=50)

best\_params = study.best\_trial.params

best\_model = XGBClassifier(\*\*best\_params, use\_label\_encoder=False)

best\_model.fit(X\_train, y\_train)

return best\_model, study.best\_trial

import pandas as pd

from sklearn.preprocessing import StandardScaler, LabelEncoder

from sklearn.model\_selection import train\_test\_split

def preprocess\_data(file\_path):

df = pd.read\_csv(file\_path)

df = df.dropna()

label\_encoders = {}

for column in df.select\_dtypes(include=['object']).columns:

le = LabelEncoder()

df[column] = le.fit\_transform(df[column])

label\_encoders[column] = le

X = df.drop('target', axis=1)

y = df['target']

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

scaler = StandardScaler()

X\_train = scaler.fit\_transform(X\_train)

X\_test = scaler.transform(X\_test)

return X\_train, X\_test, y\_train, y\_test, label\_encoders

from preprocessing.preprocess import preprocess\_data

from models.svm\_model import get\_best\_svm\_model

from models.dt\_model import get\_best\_dt\_model

from models.blda\_model import get\_best\_blda\_model

from models.knn\_model import get\_best\_knn\_model

from models.xgb\_model import get\_best\_xgb\_model

# Cargar y preprocesar datos

X\_train, X\_test, y\_train, y\_test, label\_encoders = preprocess\_data('data/data.csv')

# Obtener y evaluar el mejor modelo SVM

svm\_model, svm\_trial = get\_best\_svm\_model(X\_train, y\_train)

print(f'SVM Best Params: {svm\_trial.params}')

print(f'SVM Test Accuracy: {svm\_model.score(X\_test, y\_test)}')

# Obtener y evaluar el mejor modelo DT

dt\_model, dt\_trial = get\_best\_dt\_model(X\_train, y\_train)

print(f'DT Best Params: {dt\_trial.params}')

print(f'DT Test Accuracy: {dt\_model.score(X\_test, y\_test)}')

# Obtener y evaluar el mejor modelo BLDA

blda\_model = get\_best\_blda\_model(X\_train, y\_train)

print(f'BLDA Test Accuracy: {blda\_model.score(X\_test, y\_test)}')

# Obtener y evaluar el mejor modelo KNN

knn\_model, knn\_trial = get\_best\_knn\_model(X\_train, y\_train)

print(f'KNN Best Params: {knn\_trial.params}')

print(f'KNN Test Accuracy: {knn\_model.score(X\_test, y\_test)}')

# Obtener y evaluar el mejor modelo XGB

xgb\_model, xgb\_trial = get\_best\_xgb\_model(X\_train, y\_train)

print(f'XGB Best Params: {xgb\_trial.params}')

print(f'XGB Test Accuracy: {xgb\_model.score(X\_test, y\_test)}')

pandas

scikit-learn

xgboost

optuna