```
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
```