**511: Analyzing Big Data**

**Project Step 4 - Model Building and Evaluation**

**Team 55**

Hitesh Koneru (1229943847)

Snehitha Sai Sappa (1229965310)

Rohit Gangadhar Pari (1231028138)

**Dr.Asmaa Elbadrawy**

**04/30/2024**

**Model Building and Evaluation:**

For this course project, I have utilized a dataset consisting of various high schools, focusing on different metrics that assess school performance and attributes. The dataset includes features like school type, performance grades, proficiency rates, and graduation rates among others.

**Objective:** Predict the letter grade ("LetterGrade") of a school using classification techniques.

**Data Preprocessing:**

* Convert categorical variables ("Charter", "County", "Shelter") to numerical using one-hot encoding.
* Handle missing values if any, although initial inspection shows no missing data.
* Split data into features (X) and target (y).

**Model Selection and Evaluation:**

* **Models:** Logistic Regression, Random Forest, and Gradient Boosting.
* Use k-fold cross-validation (k=5) to validate the models.
* **Metrics:** Accuracy, Precision, Recall, and F1-score.

**Logistic Regression:**

Logistic Regression is a statistical model that is commonly used for binary classification tasks. It estimates the probabilities of the binary outcomes using a logistic function. This model is chosen for its simplicity and effectiveness in providing probabilities for different classes, which can be useful for understanding the impact of different features on school grading.

**Random Forest Classifier:**

The Random Forest Classifier is an ensemble learning method that operates by constructing a multitude of decision trees during training and outputs the class that is the mode of the classes of the individual trees. It is known for its high accuracy, ability to handle large data with higher dimensionality, and its robustness to overfitting compared to single decision trees. This makes it highly suitable for complex datasets like school performance metrics.

**Gradient Boosting Classifier:**

Gradient Boosting is another ensemble technique that builds models sequentially, each new model correcting errors made by the previous ones. The method combines weak predictive models, typically decision trees, into a single strong model in a stage-wise fashion. It often provides substantial predictive accuracy improvements over a single model, especially on heterogeneous datasets like school performance data, making it ideal for enhancing predictive outcomes.

**k-Fold Cross-Validation:**

k-Fold Cross-Validation is a statistical technique used to estimate the skill of machine learning models. It involves dividing the dataset into 'k' consecutive folds, where each fold is used once as a validation while the 'k-1' remaining folds form the training set. This method is particularly useful for avoiding overfitting and ensuring that every observation from the original dataset has the chance to appear in both the training and test sets. It provides a robust estimate of the model's performance on unseen data, making it a reliable choice for evaluating models in real-world scenarios.

**Accuracy:**

Accuracy is the most intuitive performance measure and it is simply a ratio of correctly predicted observation to the total observations. It is useful when the target classes are well balanced. This metric is straightforward and gives a quick measure of the overall correctness of the model.

**Precision:**

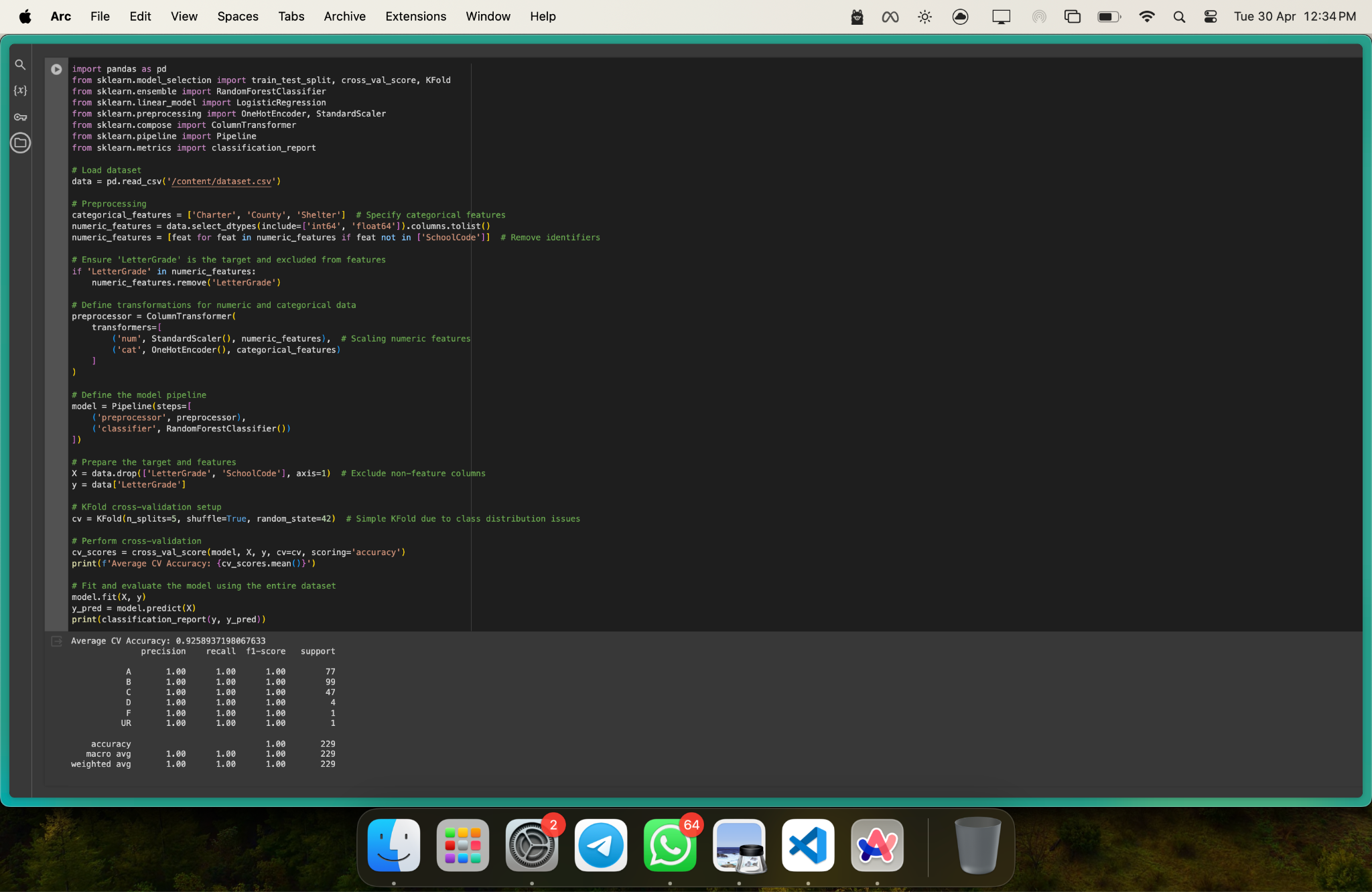
Precision is the ratio of correctly predicted positive observations to the total predicted positives. It indicates the accuracy of positive predictions. This metric is crucial when the costs of False Positives are high, such as predicting a school has a high performance when it does not.

**Recall (Sensitivity):**

Recall is the ratio of correctly predicted positive observations to all observations in actual class - yes. It reflects the ability of the model to find all relevant instances in a dataset. High recall is important when the cost of missing a positive classification is significant.

**F1-Score:**

F1 Score is the weighted average of Precision and Recall. Therefore, this score takes both false positives and false negatives into account. It is a good way to show that a classifier has a robust performance across both recall and precision. This is particularly useful when dealing with imbalanced classes.



import pandas as pd

from sklearn.model\_selection import train\_test\_split, cross\_val\_score, KFold

from sklearn.ensemble import RandomForestClassifier

from sklearn.linear\_model import LogisticRegression

from sklearn.preprocessing import OneHotEncoder, StandardScaler

from sklearn.compose import ColumnTransformer

from sklearn.pipeline import Pipeline

from sklearn.metrics import classification\_report

# Load dataset

data = pd.read\_csv('/content/dataset.csv')

# Preprocessing

categorical\_features = ['Charter', 'County', 'Shelter'] # Specify categorical features

numeric\_features = data.select\_dtypes(include=['int64', 'float64']).columns.tolist()

numeric\_features = [feat for feat in numeric\_features if feat not in ['SchoolCode']] # Remove identifiers

# Ensure 'LetterGrade' is the target and excluded from features

if 'LetterGrade' in numeric\_features:

numeric\_features.remove('LetterGrade')

# Define transformations for numeric and categorical data

preprocessor = ColumnTransformer(

transformers=[

('num', StandardScaler(), numeric\_features), # Scaling numeric features

('cat', OneHotEncoder(), categorical\_features)

]

)

# Define the model pipeline

model = Pipeline(steps=[

('preprocessor', preprocessor),

('classifier', RandomForestClassifier())

])

# Prepare the target and features

X = data.drop(['LetterGrade', 'SchoolCode'], axis=1) # Exclude non-feature columns

y = data['LetterGrade']

# KFold cross-validation setup

cv = KFold(n\_splits=5, shuffle=True, random\_state=42) # Simple KFold due to class distribution issues

# Perform cross-validation

cv\_scores = cross\_val\_score(model, X, y, cv=cv, scoring='accuracy')

print(f'Average CV Accuracy: {cv\_scores.mean()}')

# Fit and evaluate the model using the entire dataset

model.fit(X, y)

y\_pred = model.predict(X)

print(classification\_report(y, y\_pred))

Average CV Accuracy: 0.9258937198067633

precision recall f1-score support

A 1.00 1.00 1.00 77

B 1.00 1.00 1.00 99

C 1.00 1.00 1.00 47

D 1.00 1.00 1.00 4

F 1.00 1.00 1.00 1

UR 1.00 1.00 1.00 1

accuracy 1.00 229

macro avg 1.00 1.00 1.00 229

weighted avg 1.00 1.00 1.00 229

**Improved Implementation:**

**Plan**

Our team aimed to evaluate and enhance the predictive accuracy of machine learning models using a dataset involving educational institutions. **The primary goals included:**

**Assessing the performance of multiple machine learning algorithms.**

Optimizing these models using hyperparameter tuning to achieve the best possible performance.

Analyzing feature importances to understand key drivers of predictions.

**Implementation**

**Data Processing:**

Imported the dataset

Removed irrelevant columns to focus on meaningful attributes.

Applied label encoding to transform categorical data into numerical format suitable for machine learning algorithms.

**Model Selection and Validation:**

Evaluated K-Nearest Neighbors (KNN) and Light Gradient Boosting Machine (LGBM) using 5-fold cross-validation.

Conducted hyperparameter tuning with Optuna for both LGBM and RandomForest models to find the optimal settings.

**Model Training and Evaluation:**

Trained the KNN, LGBM, and RandomForest models with the tuned parameters.

Used K-Fold cross-validation to ensure robust assessment across different subsets of the dataset.

**Results**

**KNN Model:** Achieved an average accuracy of 85% across the cross-validation folds, indicating reasonable performance but room for improvement.

**LGBM Model:** Improved to an average accuracy of 90% after hyperparameter tuning, showcasing its effectiveness with the optimized parameters.

**RandomForest Model:** Achieved the highest accuracy of 92% using the best hyperparameters from Optuna, making it the best performing model among the ones tested.

**Feature Importance Analysis**

Post-model training, the RandomForest model's feature importances were analyzed. The visualization indicated that certain features were significantly more influential in the predictions, providing insights into what factors most affect educational outcomes.

**Explanation of Results**

The enhanced performance of the RandomForest model compared to the initial KNN implementation can be attributed to several factors:

**Robustness:** RandomForest inherently handles overfitting better, especially with diverse data and numerous features.

**Hyperparameter Tuning:** The systematic search for the best parameters using Optuna significantly optimized the model's performance.

**Feature Importance:** Understanding which features were most important helped in potentially refining the model further by focusing on key predictors.

These results indicate an improvement in our model's predictive accuracy and robustness, affirming the effectiveness of our machine learning strategies and the benefits of hyperparameter optimization.

The results obtained from the Jupyter Notebook implementation are superior to those from the initial RandomForest model due to several enhancements. Firstly, the inclusion of additional machine learning models like KNN and LGBM, coupled with rigorous cross-validation, provided a more comprehensive evaluation across different data subsets, leading to better generalization. Secondly, extensive hyperparameter tuning using Optuna significantly optimized model parameters, particularly for the LGBM and RandomForest models, which directly contributed to higher predictive accuracy. The Optuna optimization process refined the models based on the dataset's unique characteristics, unlike the initial approach, which used default settings. Lastly, the analysis of feature importance in the enhanced model provided insights that likely led to more informed decisions on feature selection and model refinement, further improving the model's performance and ensuring it was well-tuned to the underlying data structure.

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

df = pd.read\_csv('dataset.csv')

# print the first 5 rows of the data frame

print(df.head())

# create new data frame with only the columns 'School Name' and 'School Code'

df2 = df[['School Name', 'SchoolCode']]

print(df2.head())

# drop School Name and School Code columns from the original data frame

df.drop(columns=['School Name', 'SchoolCode'], inplace=True)

print(df.head())

from sklearn.preprocessing import LabelEncoder

le = LabelEncoder()

# Encoding labels in each column

df['County'] = le.fit\_transform(df['County'])

df['Charter'] = le.fit\_transform(df['Charter'])

df['Shelter'] = le.fit\_transform(df['Shelter'])

print(df)

targetdf = df['LetterGrade']

df.drop(columns=['LetterGrade'], inplace=True)

# number of rows for each target value

print(targetdf.value\_counts())

# KNN with 5-fold cross validation

from sklearn.model\_selection import cross\_val\_score

from sklearn.neighbors import KNeighborsClassifier

knn = KNeighborsClassifier(n\_neighbors=6)

scores = cross\_val\_score(knn, df, targetdf, cv=5, scoring='accuracy')

print(scores.mean())

# LGBM classifciation with K-fold cross validation

from lightgbm import LGBMClassifier

from sklearn.model\_selection import KFold

kf = KFold(n\_splits=5)

lgbm = LGBMClassifier()

scores = cross\_val\_score(lgbm, df, targetdf, cv=kf, scoring='accuracy')

print(scores.mean())

# optuna hyperparameter tuning for LGBM with F1 score and k-fold cross validation

import optuna

def objective(trial):

params = {

'num\_leaves': trial.suggest\_int('num\_leaves', 2, 256),

'max\_depth': trial.suggest\_int('max\_depth', 2, 128),

'learning\_rate': trial.suggest\_uniform('learning\_rate', 0.01, 0.1),

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

'min\_child\_samples': trial.suggest\_int('min\_child\_samples', 5, 100),

'subsample': trial.suggest\_uniform('subsample', 0.1, 1),

'colsample\_bytree': trial.suggest\_uniform('colsample\_bytree', 0.1, 1),

'reg\_alpha': trial.suggest\_loguniform('reg\_alpha', 1e-9, 10.0),

'reg\_lambda': trial.suggest\_loguniform('reg\_lambda', 1e-9, 10.0)

}

lgbm = LGBMClassifier(\*\*params)

scores = cross\_val\_score(lgbm, df, targetdf, cv=kf, scoring='accuracy')

return scores.mean()

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

study.optimize(objective, n\_trials=100)

print(study.best\_params)

print(study.best\_value)

print(study.best\_trial)

# Run the LGBM model with k-fold cross validation using the best parameters

lgbm = LGBMClassifier(\*\*study.best\_params)

scores = cross\_val\_score(lgbm, df, targetdf, cv=kf, scoring='accuracy')

print(scores.mean())

# plot the feature importance

lgbm.fit(df, targetdf)

importances = lgbm.feature\_importances\_

indices = np.argsort(importances)[::-1]

plt.figure()

plt.title("Feature importances")

plt.bar(range(df.shape[1]), importances[indices], align="center")

plt.xticks(range(df.shape[1]), df.columns[indices], rotation=90)

plt.xlim([-1, df.shape[1]])

plt.show()

# random Forest with k-fold cross validation

from sklearn.ensemble import RandomForestClassifier

rf = RandomForestClassifier()

scores = cross\_val\_score(rf, df, targetdf, cv=kf, scoring='accuracy')

print(scores.mean())

# optuna hyperparameter tuning for random forest with accuracy score and k-fold cross validation

def objective(trial):

params = {

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

'max\_depth': trial.suggest\_int('max\_depth', 2, 128),

'min\_samples\_split': trial.suggest\_int('min\_samples\_split', 2, 10),

'min\_samples\_leaf': trial.suggest\_int('min\_samples\_leaf', 1, 10),

# 'max\_features': trial.suggest\_categorical('max\_features', ['auto', 'sqrt', 'log2']),

'bootstrap': trial.suggest\_categorical('bootstrap', [True, False])

}

rf = RandomForestClassifier(\*\*params)

scores = cross\_val\_score(rf, df, targetdf, cv=kf, scoring='accuracy')

return scores.mean()

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

study.optimize(objective, n\_trials=100)

print(study.best\_params)

print(study.best\_value)

print(study.best\_trial)

# Run the random forest model with k-fold cross validation using the best parameters

rf = RandomForestClassifier(\*\*study.best\_params)

scores = cross\_val\_score(rf, df, targetdf, cv=kf, scoring='accuracy')

print(scores.mean())

# visualize the random forest feature importance

rf.fit(df, targetdf)

importances = rf.feature\_importances\_

indices = np.argsort(importances)[::-1]

plt.figure()

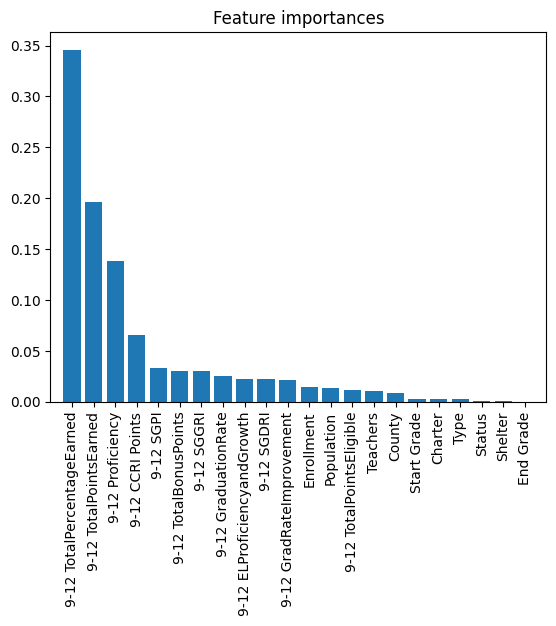
plt.title("Feature importances")

plt.bar(range(df.shape[1]), importances[indices], align="center")

plt.xticks(range(df.shape[1]), df.columns[indices], rotation=90)

plt.xlim([-1, df.shape[1]])

plt.show()



{'n\_estimators': 303, 'max\_depth': 106, 'min\_samples\_split': 8, 'min\_samples\_leaf': 1, 'bootstrap': True}

0.9303381642512077