Faculty of Engineering and Technology
Electrical and Computer Engineering Department
Intelligent Systems Laboratory
ENCS5141

Case Study #2

Comparative Analysis of Classification Techniques: Random Forest (RF), Extreme Gradient Boosting (XGBoost), and Multilayer Perceptron (MLP) on the Bike Sharing Dataset

Prepared by

Usama Shoora 1200796

Instructor

Dr. Mohammad Jubran

Teacher assistant

Eng. Hanan Awawdeh

Section **1**

BIRZEIT
August – 2024

Abstract

This case study uses the Bike Sharing Dataset to evaluate the prediction performance of three machine learning models: Random Forest (RF), Extreme Gradient Boosting (XGBoost), and Multilayer Perceptron (MLP). Based on rental counts, bike demand is divided into four categories: "low," "medium," "high," and "extreme." Each model is trained and tested against the preprocessed dataset. The results show various degrees of accuracy and classification capabilities across the models, showing their strengths and weaknesses in predicting categorized bike demand. The findings provide insight on each model's suitability for real-world bike-sharing systems.

Table of Contents

List of Listings

List of Tables

1. Introduction

The rise of bike-sharing systems has brought a handy and sustainable means of transportation to urban areas. Accurately predicting bike demand is critical for optimizing bike distribution and availability, which improves user experience and operational efficiency. This report compares the performance of three machine learning models—Random Forest (RF), Extreme Gradient Boosting (XGBoost), and Multilayer Perceptron (MLP)—in predicting bike rental demand using the Bike Sharing Dataset.

The dataset includes a variety of features, including weather conditions, temporal data, and user type, which all influence bike rental patterns. The target variable, representing bike demand, is divided into four categories: "low," "medium," "high," and "extreme." These categories allow for a more refined analysis of rental patterns, as well as the construction of models that can properly predict demand under various conditions.

This study's evaluation metrics are precision, recall, and f1-score, which provide a thorough assessment of model performance. Precision is the proportion of true positive predictions among all positive predictions, demonstrating how accurate the model is at detecting each demand category. Recall measures the fraction of true positive predictions among all actual positives, indicating the model's capacity to catch all instances of each category.

This report covers the steps involved in categorizing, training, and testing data. Each model is trained and tested on processed data, and its strengths and weaknesses are evaluated using evaluation metrics. This study's findings provide useful insights into the suitability of RF, XGBoost, and MLP models for predicting bike demand, which will help to create efficient and user-friendly bike-sharing systems.

2. Theory

2.1. Random Forest (RF)

Random Forest is an ensemble learning method that constructs a large number of decision trees during training and outputs their mode (classification) or mean prediction (regression). This method is known for its capability to resist overfitting and ability to handle massive datasets with higher dimensionality. It employs bagging (bootstrap aggregation). The algorithm divides the original dataset into subsets, replaces them, and trains a decision tree on each one. The final forecast is the average of all the tree predictions [1].

- **Hyperparameters (the ones used in this study)**:
  - **Max_depth**: The maximum depth of the tree. Limiting the depth of the tree helps control overfitting by ensuring that the model captures only the most important patterns in the data.
  - **Min_samples_split**: The minimum number of samples required to split an internal node. Higher values prevent the model from learning overly specific patterns, thereby reducing overfitting.
  - **N_estimators**: The number of trees in the forest. A larger number of trees usually improves the performance of the model but also increases computational cost.

2.2. Extreme Gradient Boosting (XGBoost)

XGBoost is an optimized distributed gradient boosting library that is highly efficient, adaptable, and portable. It uses ensemble modeling to boost the performance of weak learners, usually decision trees, by iteratively adding models that rectify prior failures. It is an ensemble model that differs from others by training multiple decision trees sequentially. The training process involves several steps. First, random samples are drawn with replacement from the training data, a technique known as bootstrapping. For each sample, a decision tree is built by randomly selecting a subset of features at each node and then splitting the node using the best feature among the selected subset. This process of creating individual trees is repeated multiple times. Finally, the predictions from all the trees are aggregated to produce the final prediction [2].

- **Hyperparameters (the ones used in this study)**:
  - **Learning_rate**: The step size shrinkage used in updating weights after each boosting iteration. It scales the contribution of each tree, preventing overfitting.
  - **Max_depth**: The maximum depth of the trees. Controls the complexity of the individual trees and helps in avoiding overfitting.
  - **N_estimators**: The number of boosting rounds. More rounds usually lead to better performance but require more computational resources.

2.3. Multilayer Perceptron (MLP)

MLP is a class of feedforward artificial neural network (ANN). It consists of at least three layers: an input layer, one or more hidden layers, and an output layer. Each neuron in one layer connects with a certain weight to every neuron in the next layer. The steps involved in using MLP include preprocessing the data, defining the network architecture, selecting the activation function, training the model using backpropagation, tuning the hyperparameters, and evaluating the model's performance [3].

- **Hyperparameters (the ones used in this study)**:
    - **Learning_rate**: The rate at which the model weights are updated. A higher learning rate might converge faster but can overshoot minima.
    - **Activation**: The activation function for the hidden layer. Common choices include 'relu' (Rectified Linear Unit), 'tanh', and 'logistic' (sigmoid). It defines the output of a node given an input or set of inputs.
    - **Hidden_layer_sizes**: The number and size of the hidden layers. For example, (100,) means one hidden layer with 100 neurons.
    - **Solver:** The optimization algorithm to use for weight optimization. Common solvers include 'adam', 'sgd', and 'lbfgs'.

2.4. Evaluation Metrics

The evaluation metrics that will be used in this study are the following:

- **Precision**: The ratio of true positive predictions to the total number of positive predictions. It measures the accuracy of the positive predictions made by the model.

$$Precision = \frac{True\ Positives}{True\ Positives + False\ Positives}$$

- **Recall**: The ratio of true positive predictions to the total number of actual positives. It indicates the model's ability to identify all relevant instances.

$$Recall = \frac{True\ Positives}{True\ Positives + False\ Negatives}$$

- **F1-Score**: The harmonic mean of precision and recall, providing a single metric that balances both concerns.

$$F1 - Score = 2 \times \frac{Precision \times Recall}{Precision + Recall}$$

- **Accuracy**: The ratio of correctly predicted instances to the total instances.

$$Accuracy = \frac{True\ Positives + True\ Negatives}{Total\ Instances}$$

- **Macro Average**: The average of the precision, recall, and f1-score calculated for each class independently, without considering class imbalance.

$$Macro\ Average = \frac{\sum Precision_{class(i)}}{Number\ of\ Classes}$$

- **Weighted Average**: The average of the precision, recall, and f1-score calculated for each class, weighted by the number of instances in each class.

$$Weighted\ Average = \frac{\sum(Support_{class} \times Metrics_{class})}{Total\ Support}$$

3. Procedure and Discussion

3.1. Importing the Necessary Dependencies

For this case study, libraries for the models, metrics, label-encoding, splitting the data, cross-validation using Grid Search, and pandas will be imported.

```
import pandas as pd

from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.metrics import classification_report
from sklearn.preprocessing import LabelEncoder

from sklearn.neural_network import MLPClassifier
from sklearn.ensemble import RandomForestClassifier
from xgboost import XGBClassifier
```

Listing 3.1: Importing Dependencies

These are the necessary libraries to complete this procedure.

3.2. Loading the Dataset

After completing Case Study #1 and saving the data after encoding and the final cleaned data to “encoded_data.csv” and “PCA_data.csv” respectively, they’ll be used to split the dataset and train the models.

```
PCA_data = pd.read_csv("./PCA_data.csv")
encoded_data = pd.read_csv("./encoded_data.csv")
```

Listing 3.2: Loading Datasets

3.3. Categorizing and Splitting the Data

The aim is to train the three models to categorize/classify bike demand as "extreme," "high," "medium," or "low" based on rental bike counts. To do this, the data after PCA needs to be categorized into bins in the ranges of *[0, 0.25, 0.5, 0.75, 1.0]* and labels *['low', 'medium', 'high', 'extreme']*. Then the data can be split for training and testing.

```
PCA_X = PCA_data
PCA_y = encoded_data['count']

bins = [0, 0.25, 0.5, 0.75, 1.0]
labels = ['low', 'medium', 'high', 'extreme']

label_encoder = LabelEncoder()
PCA_y_binned = label_encoder.fit_transform(pd.cut(PCA_y, bins=bins, labels=labels))

PCA_X_train, PCA_X_test, PCA_y_train, PCA_y_test = train_test_split(PCA_X, PCA_y_binned,
                                                            test_size=0.2, random_state=42)
```

Listing 3.3: Categorizing and Splitting the Data

3.4. Hyperparameter Tuning and Training

To find the best hyperparameters to use for each model, cross validation using Grid Search will be utilized with pre-defined lists of parameters for each model. **The Random Forest Classifier** will be tuned for the parameters *[n_estimators, max_depth, min_samples_split]*. **The XGBoost** model will be tuned for the following *[n_estimators, max_depth, learning_rate]*. **The MLP** model will be tuned for *[hidden_layer_sizes, activation, solver, learning_rate]*.

```
rf = RandomForestClassifier()
xgb = XGBClassifier()
mlp = MLPClassifier()

rf_params = {
    'n_estimators': [50, 100, 200],
    'max_depth': [None, 10, 20, 30],
    'min_samples_split': [2, 5, 10]
}
xgb_params = {
    'n_estimators': [50, 100, 200],
    'max_depth': [3, 6, 9],
    'learning_rate': [0.01, 0.1, 0.2]
}
mlp_params = {
    'hidden_layer_sizes': [(50,), (100,), (100, 50)],
    'activation': ['relu', 'tanh'],
    'solver': ['adam', 'sgd'],
    'learning_rate': ['constant', 'adaptive']
}
```

Listing 3.4: Defining Models and Hyperparameters Grids

To perform Grid Search cross-validation and train the data on the best hyperparameters for each model:

```
def perform_grid_search(model, params, X_train, y_train):
    grid_search = GridSearchCV(estimator=model, param_grid=params,
                               cv=5, scoring='accuracy', n_jobs=-1)
    grid_search.fit(X_train, y_train)
    return grid_search.best_estimator_, grid_search.best_params_

best_rf, best_rf_params = perform_grid_search(rf, rf_params, PCA_X_train, PCA_y_train)
best_xgb, best_xgb_params = perform_grid_search(xgb, xgb_params, PCA_X_train, PCA_y_train)
best_mlp, best_mlp_params = perform_grid_search(mlp, mlp_params, PCA_X_train, PCA_y_train)

print("Best Random Forest Parameters:", best_rf_params)
print("Best XGBoost Parameters:", best_xgb_params)
print("Best MLP Parameters:", best_mlp_params)
```

Listing 3.5: Performing GridSearch and Training the Models

After running for 45.22 mins for searching and training, the results were the following:

```
Best Random Forest Parameters: {'max_depth': 30, 'min_samples_split': 5, 'n_estimators': 100}
Best XGBoost Parameters: {'learning_rate': 0.2, 'max_depth': 6, 'n_estimators': 200}
Best MLP Parameters: {'activation': 'tanh', 'hidden_layer_sizes': (100, 50), 'learning_rate': 'adaptive',
                      'solver': 'adam'}
```

Listing 3.6: Best Hyperparameters for Each Model Based on GridSearch

3.5. Testing

Now that each model is trained on the data using the best hyperparameters from the given grids, testing them will be conducted through the following code.

```
# Evaluate the models and print the best parameters
def evaluate_model(model, X_test, y_test):
    y_pred = model.predict(X_test)
    unique_labels = label_encoder.inverse_transform(sorted(set(y_test)))
    report = classification_report(y_test, y_pred, target_names=unique_labels)
    print(report)

print("Random Forest Evaluation:")
evaluate_model(best_rf, PCA_X_test, PCA_y_test)

print("XGBoost Evaluation:")
evaluate_model(best_xgb, PCA_X_test, PCA_y_test)

print("MLP Evaluation:")
evaluate_model(best_mlp, PCA_X_test, PCA_y_test)
```

Listing 3.7: Testing the Models

After training and testing the Random Forest, XGBoost, and MLP models, the results for categorizing bike rental demands as low, medium, high, and extreme based on count are as follows:

- **Random Forest Evaluation**

Table 3.1: Random Forest Evaluation Metrics

	Precision	Recall	F1-Score	Support
High	0.88	0.83	0.85	290
Low	0.99	0.99	0.99	2420
Medium	0.91	0.92	0.91	743
Overall				
Accuracy			0.96	3453
Macro avg	0.93	.091	0.92	3453
Weighted avg	0.96	0.96	0.96	3453

- **XGBoost Evaluation**

Table 3.2: XGBoost Evaluation Metrics

	Precision	Recall	F1-Score	Support
High	0.89	0.84	0.86	290
Low	0.99	0.99	0.99	2420
Medium	0.91	0.93	0.92	743
Overall				
Accuracy			0.97	3453
Macro avg	0.93	0.92	0.93	3453
Weighted avg	0.97	0.97	0.97	3453

- **MLP Evaluation**

Table 3.3: MLP Evaluation Metrics

	Precision	Recall	F1-Score	Support
High	0.85	0.83	0.84	290
Low	1.00	1.00	1.00	2420
Medium	0.93	0.93	0.93	743
Overall				
Accuracy			0.97	3453
Macro avg	0.93	0.92	0.92	3453
Weighted avg	0.97	0.97	0.97	3453

All three models show high accuracy, with **RF** at 96%, **XGBoost** at 97%, and **MLP** at 97%. This suggests that the models are generally suitable for predicting bike rental demand categories based on the PCA-transformed features. The high *precision* and *recall* for the 'low' category across all models indicate that the models are especially good at predicting low bike rental demand, which is also the majority class (support = 2420).

The 'high' and 'medium' categories have slightly lower *precision* and *recall* than the 'low' category. This is anticipated, given the lower support (number of instances) for these classes (290 for high and 743 for medium). The *F1-scores* for the 'high' and 'medium' categories remain quite high, demonstrating that the models perform well even with fewer examples.

Random Forest performs well with high accuracy and balanced *precision* and *recall* across categories. However, it has somewhat worse *precision* and *recall* in the 'high' category than **XGBoost** and **MLP**. **XGBoost** slightly surpasses **Random Forest** in terms of *accuracy* and *precision*/*recall* in the 'high' and 'medium' categories. This could be attributed to **XGBoost's** capacity to handle complicated relationships and interactions between features. **MLP** performs similarly to **XGBoost**, with flawless *precision* and *recall* for the 'low' category and high scores in the 'medium' and 'high' categories. The **MLP's** performance demonstrates that neural networks can effectively detect patterns in data post PCA transformation.

It is worth mentioning that the ‘extreme’ category did not show up in the evaluation metrics, and this is due to it having no instances in the data based on the selected ranges. Which can be checked by running the following code.

```
binned_counts = pd.cut(encoded_data['count'], bins=bins, labels=labels)
print(binned_counts.value_counts())
```

Listing 3.8: Checking Number of Instances for Each Category

```
count
low          12105
medium        3782
high          1375
extreme          0
Name: count, dtype: int64
```

Listing 3.9: Number of Instances for Each Category

4. Conclusion

In this case study, three machine learning models—Random Forest (RF), Extreme Gradient Boosting (XGBoost), and Multilayer Perceptron (MLP)—were tested for predicting bike demand using the Bike Sharing Dataset from Case Study #1. The bike rental counts were divided into four categories: "low," "medium," "high," and "extreme." This classification is based on the distribution of rental bike counts divided into quartiles.

The models are trained on preprocessed data, and their performance is measured using classification metrics like precision, recall, f1-score, and accuracy. Each model's ability to predict the various demand categories is evaluated, revealing its strengths and weaknesses in this context.

The study found that all three models were highly accurate, with the Random Forest and XGBoost models performing similarly in terms of precision, recall, and f1-score. MLP also performed well but showed slight variations in precision for certain categories. Additionally, the "extreme" category was underrepresented in the dataset, limiting the model's ability to predict it accurately.

The Random Forest model showed a balanced performance across all categories, rendering it an excellent choice for this classification task. XGBoost demonstrated a slight boost in precision and recall for the "medium" and "high" categories, proving its ability to handle more detailed demand levels. MLP, while effective, identified potential overfitting concerns that might need additional tuning.

These results highlight the significance of dataset balancing and selecting appropriate evaluation metrics when comparing models. This study lays the foundation for selecting and fine-tuning models for bike-sharing demand predictions, eventually leading to the development of efficient and user-friendly bike-sharing systems.