6219COMP
Advanced Topics in AI
Implementation Summary

Table of Contents

List of Tables

Table of Figures

1 Introduction

This report explores the use of ML techniques to classify hourly bike rental demand by using real-world transportation data. This is done by preprocessing a public dataset with the main goal of categorizing demand into 3 categories (low, medium and high) and applying a relevant AI technique to perform an effective classification. This will provide an accurate prediction of bike rental demand based on a number of factors. This allows for an improved customer experience and more efficient planning.

2 Dataset and Technique Justification

2.1 Dataset Selection

This report aims to predict outcomes from a transportation-related dataset using appropriate AI techniques. Two datasets were considered for this.

Dataset 1: The first dataset is the UCI Bike Sharing Dataset. This dataset contains hourly records of bike rental activity in Washington D.C. and contains weather and seasonal-based features.

Dataset 2: The second dataset considered was the Seoul Bike Sharing Dataset which also includes hourly records of bike rental activity over one year in Seoul. Features include weather conditions, holidays and rental counts.

A comparison of the two datasets is shown in ***Table 1*** below:

Feature	Dataset 1 (UCI Dataset)	Dataset 2 (Seoul Dataset)
Number of Records	17,379	8,760
Time Resolution	Daily and Hourly (Only hourly version considered)	Hourly
Target Outcome	cnt (rental count)	Rented Bike Count
Set of Features	17	14
Main Features	Hour, weekday, season, weather	Solar radiation, snow
Format	CSV	CSV

Table 2-1: Comparison of datasets

While both datasets are suitable for analysis, the UCI Bike Sharing Dataset was chosen. One reason was due to its large sample size which allows for more vigorous model training and evaluation. Also, its features are well-suited for classification tasks and include time-of-day and weather-based data that are commonly known to influence demand for transportation. Additionally, the dataset has been well-used in machine learning research which makes it a reliable choice for this coursework.

To structure the problem for a classification approach, the continuous hourly rental count (cnt) are grouped into three categories (low, medium and high demand). This was done based on their distribution in the dataset. This ensured that the class roughly contained an equal number of examples in order to avoid class imbalance during the model training process. Structuring it in this way allowed it to be approached as a supervised multi-class classification task, avoiding challenges of skewed data distribution whilst aligning with the strengths of traditional AI methods.

2.2 AI Technique Justification

A Random Forest Classifier was selected for this task. It is a commonly used machine learning algorithm that combines the output of multiple decision trees (non-parametric

supervised learning algorithm) to reach one conclusion. It can handle both regression and classification problems (IBM, n.d.).

It was selected due to its ability to model the complex relationships present in transportation data. The hourly demand for bike rental is impacted by a number of independent factors such as the temperature, time of day, day of the week and season. A Random Forest is perfect for capturing the data patterns through its collection of decision trees which are each trained on different parts of the data. This approach to the problem enhances accuracy whilst reducing the risk of overfitting (where a model performs well on training data but poorly on unseen data) (Google Developers, 2024).

Another advantage of using a Random Forest is its flexible handling of datasets with categorical and numerical features. This ensures that all variables such as season, weather condition and hour are all easily handled along with continuous data such as temperature and humidity. Data preprocessing is also simplified when using a Random Forest, compared to other models, as normalization is not required.

The final strength of the Random Forest algorithm is its ability to highlight which features are the most important for making predictions. This makes the model more transparent as it emphasizes the factors that have the biggest impact on bike rental demand. This is useful in the field of transportation where understanding demand changes can support more precise, efficient planning and decision making.

In summary, the use of a structured dataset paired with a high-performing model such as the Random Forest Classifier makes this approach well-suited for extracting meaningful insights and beneficial outcomes in the field of transportation.

3 Data Preparation

The dataset that was selected for this task was the UCI Bike Sharing Dataset. This dataset contains 17,379 hourly records of rental activity in Washington, D.C. Before the model could be trained on the dataset, preprocessing steps were required to prepare the data and define the classification task at hand.

The first step in data preparation was to convert the original prediction problem into a classification task. The target variable ***cnt*** indicates the number of bike rentals per hour and is a continuous numeric value. Since the goal was to classify the demand for bike rentals into three categories instead of predicting exact values, the variable was changed into a new categorical feature called ***demand_class***. The variable groups the rental counts into three categories: low, medium and high. This was done using a method that evenly split the data based on distribution and was implemented via the ***pandas.qcut()*** function, which assigns equal numbers of samples to each category (1/3 to each). This results in a balanced classification problem, reducing the risk of biased predictions during the training process, improving overall fairness of the evaluation across each class.

Binning target variable:

```
df['demand_class'] = pd.qcut(df['cnt'], q=3, labels=['Low', 'Medium', 'High'])
```

	cnt	demand_class
8680	202	Medium
11062	2	Low
9630	2	Low
1724	105	Medium
7428	6	Low
13638	251	High
2207	141	Medium
2336	58	Low
2556	98	Medium
7757	31	Low

Figure 3-1: Sample rows showing cnt values grouped into Low, Medium and High demand classes.

```
Class distribution:

demand_class
Low       5797
Medium    5823
High      5759
```

Figure 3-2: Distribution of demand classes showing very similar class sized after binning process.

A number of columns were removed from the dataset to avoid unnecessary repetition and data leakage. Data leakage occurs when the training process is influenced by information that wouldn't be available at the time of prediction (Mucci, 2024). In this case, the ***casual*** and ***registered*** columns were removed as the total sum of bikes rented per hour can be calculated using these two variables (***cnt = casual + registered***). Keeping these in would inflate model accuracy and invalidate the training process. Other columns that are dropped include ***cnt*** (***demand_class*** is being used instead), ***instant*** (row index) and ***dteday*** (date string that isn't used).

Dropping data leakage and unused columns:

```
df = df.drop(columns=['instant', 'dteday', 'casual', 'registered', 'cnt'])
```

```
Remaining columns:

['season', 'yr', 'mnth', 'hr', 'holiday', 'weekday', 'workingday', 'weathersit', 'temp', 'atemp', 'hum', 'windspeed', 'demand_class']

Data preview:

   season  yr  mnth  hr  holiday  ...  temp   atemp   hum  windspeed  demand_class
0       1   0     1   0        0  ...  0.24  0.2879  0.81        0.0           Low
1       1   0     1   1        0  ...  0.22  0.2727  0.80        0.0           Low
2       1   0     1   2        0  ...  0.22  0.2727  0.80        0.0           Low
3       1   0     1   3        0  ...  0.24  0.2879  0.75        0.0           Low
4       1   0     1   4        0  ...  0.24  0.2879  0.75        0.0           Low
```

Figure 3-3: Remaining columns and sample data after dropping data leakage and unnecessary fields from dataset.

After this, categorical variables were converted to the appropriate data type. Variables included ***season***, ***year***, ***month***, ***hour***, ***weekday***, ***holiday***, ***workingday*** and ***weathersit***. This allowed these to be one-hot encoded using the ***pandas.get_dummies()*** function. These were converted into binary indicators that enables the model to interpret them numerically. To stop multicollinearity from occurring, the first category from each feature was dropped. This expands the feature set to 55 columns all of which being in a machine-readable form (Murel & Kavlakoglu, 2023).

Converting to categorical and encoding:

```
# Convert categories

categories = ['season', 'yr', 'mnth', 'hr', 'holiday', 'weekday', 'workingday', 'weathersit']

df[categories] = df[categories].astype('category')

# One-hot encode

df_preprocessed = pd.get_dummies(df, drop_first=True)
```

```
First 10 encoded columns:
 ['temp', 'atemp', 'hum', 'windspeed', 'season_2', 'season_3', 'season_4', 'yr_1', 'mnth_2', 'mnth_3']
```

Figure 3-4: Dataset shape and after one-hot encoding.

Finally, the processed data was saved into two files. This was done to support transparency. The first version contains the cleaned dataset after column removal but before one-hot encoding. This version preserves the original categorical structure and acts as a reference for how the features appeared before being transformed for model

training. The second version contains the fully transformed dataset and is used directly in model training. This allows the data preparation process to be clearly documented and allows future review without having to repeat earlier steps. These steps are implemented in the ***preprocessing.py*** script.

Saving pre-processed datasets:

```
df.to_csv("original_bike_data.csv", index=False)
```

```
df_encoded.to_csv("preprocessed_bike_data.csv", index=False)
```

These steps ensured that the data was clean and ready for model training. By only using relevant and independent features, the final processed dataset allowed for effective classification while preserving the interpretability of data. This is an important consideration for real-world approaches where understanding the model decision process is as important as the model's accuracy.

4 Implementation and Evaluation

4.1 Model implementation and Hyperparameter Tuning

After the preprocessing stage, the classification task was implemented using the Random Forest Classifier from ***scikit-learn***. The target variable is ***demand_class*** and the pre-processed dataset was loaded from ***preprocessed_data.csv***. The target was reconstructed by checking which class column was active for each row. A custom function, ***class_reconstruction***, was used for this to map it back to their original labels.

To ensure model performance was evaluated reliably, the dataset was split into training and test sets using ***train_test_split()***. 20% of the data was reserved for testing and ***stratify=y*** was used to preserve class distribution. The Random Forest algorithm was chosen due to its strengths when dealing with numerical and categorical data and its strong performance with minimal preprocessing.

GridSearchCV was used to optimize the model by fine tuning the hyper parameters. A 5-fold cross-validation grid was used to explore combinations of key parameters. These parameters include ***n_estimators*** (number of trees), ***max_depth*** (maximum tree depth), ***min_samples_split*** and ***min_samples_leaf*** (minimum number of samples needed to split or arrive at a leaf), and ***max_features*** (number of features considered at each split). This process allowed model performance to be explored across different configurations, allowing for efficient implementation.

The results from the grid search identified the best parameter combination. This is shown in ***figure 5***:

GRID SEARCH:

```
# Set up GridSearchCV
grid_search = GridSearchCV(
    estimator=RandomForestClassifier(random_state=42),
    param_grid=parameter_grid,
    scoring='accuracy',
    cv=5,
    n_jobs=-1,
    verbose=1
)
# Train the model
print("Grid Search Started")
grid_search.fit(X_train, y_train)
```

```
Grid Search Started
Fitting 5 folds for each of 36 candidates, totalling 180 fits
```

Figure 4-1: Grid search process initiating.

```
Best Parameters: {'max_depth': None, 'max_features': 'sqrt', 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 200}
```

Figure 4-2: Best parameters obtained from grid search process.

These parameters allowed the model to grow trees to full depth, use a number of features at each split and built a large forest (relatively) for a more stable implementation.

4.2 Evaluation Results and Interpretation

Below is a table of the results obtained after running the model:

Class	Precision	Recall	F1-Score	Support
Low	0.924	0.867	0.895	1159
Medium	0.793	0.825	0.809	1165
High	0.883	0.901	0.892	1152
Accuracy			0.864	3476
Macro Average	0.866	0.864	0.865	3476
Weighted Average	0.866	0.864	0.865	3476

Table 4-1: Table of Random Forest results.

The fine-tuned model was evaluated on the test set using a number of performance indicators. It achieved an overall accuracy of 86.42% which indicates that it is effective in correctly predicting hourly bike rental demand across all three categories. This was a slight improvement over the baseline model which achieved an accuracy of 86.28%. Although the improvement in overall accuracy was minimal, fine tuning helped achieve better class balance. This was more evident in the ***Medium*** demand class.

The most successful configuration that was highlighted by ***GridSearchCV*** included 200 estimators, no maximum depth and the sqrt method for feature sampling. These settings allowed the model to construct a diverse set of decision trees whilst managing overfitting through ensemble averaging. The final model was evaluated using precision, recall and F1-score. A confusion matrix was also produced.

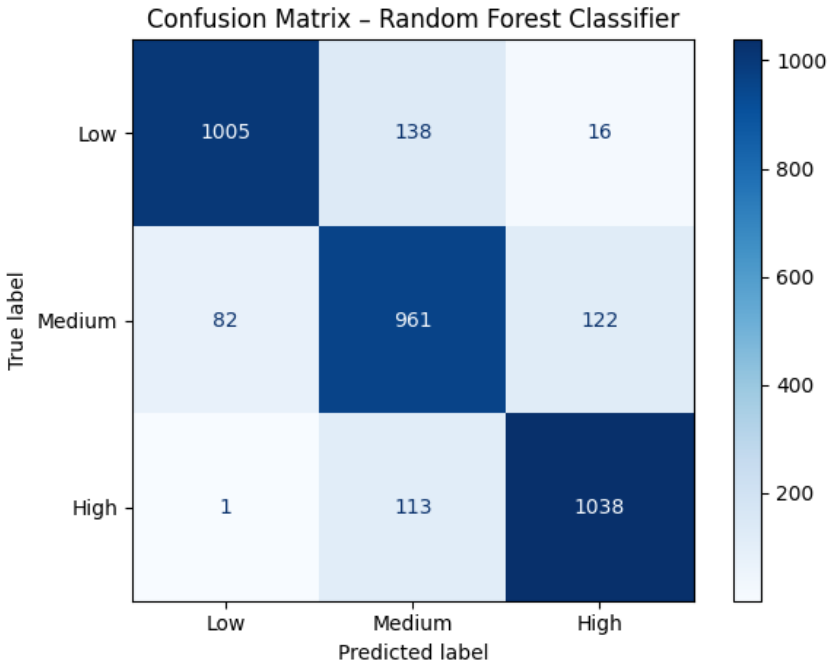

Figure 4-3: Confusion Matrix for Random Forest model

The model showed a balanced performance across all three levels of demand. Low demand had the highest precision (0.92) whereas high demand had the highest recall (0.90). The medium class had a slightly lower precision and recall. This is due to the difficulty in recognition due to the overlap between both low and high classes. However, it still produced an F1-score of 0.81 which is a respectable result for a middle class in an ordered classification problem.

A confusion matrix was plotted with most predictions calling along the diagonal. This confirms that the model has the ability to classify each demand level correctly. The misclassifications were mostly between adjacent classes. This is expected in real-world problems where the boundaries can be subtle.

The small difference in improvement seen between the tuned and baseline models is normal for Random Forest models as they often perform well with default parameters as long as the dataset is clean and balanced. Because of the highly informative feature set and well-structured classification problem, the baseline model managed to capture most of the key patterns without extensive tuning. Fine-tuning was implemented to confirm that the initial model was valid and suitable for this task.

5 Technique Comparison

To assess the effectiveness of the Random Forest Model, a decision tree was implemented using the dataset and evaluation metrics. Decision trees are simple and are highly interpretable which makes them a good for comparison. However, they are known to face issues with overfitting, especially when not paired with other machine learning algorithms.

The decision tree was implemented and achieved an overall accuracy of 80.1% compared to 86.4% that was obtained by the Random Forest. Both models performed very similarly on the low and high demand classes. F1-scores of approximately 0.84 for both classes was achieved by the decision tree, while the Random Forest achieved higher scores for both (0.89 each) and also showed improved performance on the medium class where it was more difficult to predict accurately due to the class overlap. This shows that the Random Forest was able to handle class overlap more easily compared to a singular decision tree. Below are the results of the decision tree along with the confusion matrix:

Class	Precision	Recall	F1-Score	Support
Low	0.839	0.845	0.842	1159
Medium	0.731	0.722	0.726	1165
High	0.834	0.839	0.836	1152
Accuracy			0.801	3476
Macro Average	0.801	0.802	0.801	3476
Weighted Average	0.801	0.801	0.801	3476

Table 5-1: Table of Decision tree results.

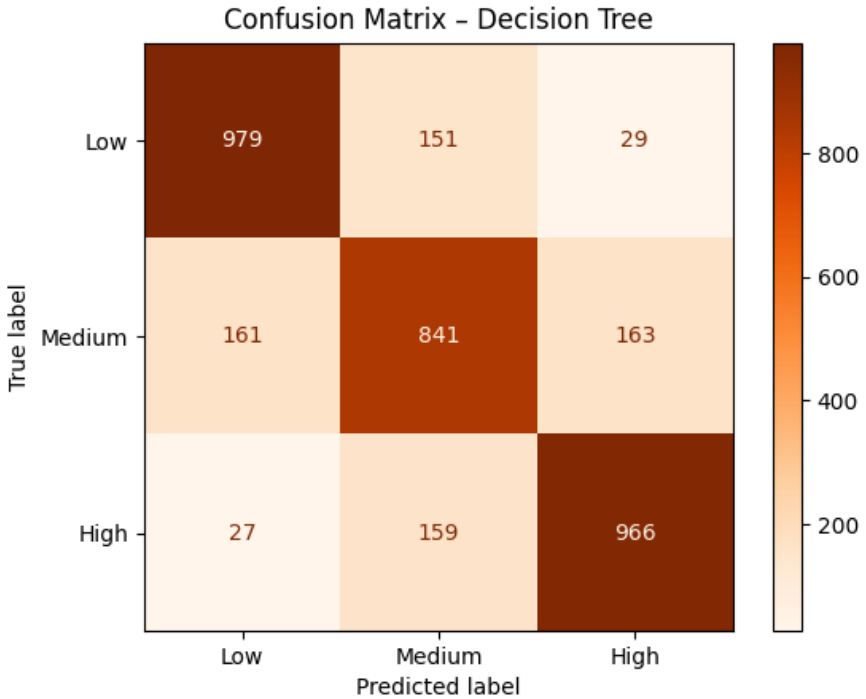

Figure 5-1: Confusion Matrix for Decision Tree

In relation to model training, the decision tree was quicker to train and implement. The Random Forest required slightly more effort in terms of programming, but it paid off through its improvement in accuracy and class balance. It's ability to handle overlapping classes (Medium class) effectively made it a better fit for this problem.

A number of other techniques, such as logistic regression, were looked into but the decision tree was chosen due to its similarity to Random Forest as both models are based on tree structures. In random forest models the data is split into decision rules which combine many trees to reduce the variance and improve generalisation. A single decision tree has to rely on individual splits and falls victim to overfitting. It was chosen as it makes an ideal baseline to highlight the advantages of ensemble methods. Linear regression wasn't chosen in the end as it lacks the ability to model more complex, non-linear relationships which are present in this dataset. As subtle, non-linear patterns are common in real-world transport data, a decision tree was the best technique to use for comparison.

6 Conclusion

This implementation successfully demonstrates how useful ML algorithms are in real-world scenarios, especially in transportation. A Random Forest model was implemented and achieved high accuracy in classifying bike rental demand and outperformed the singular decision tree baseline. By preprocessing the data, fine tuning parameters and evaluating the models, the approach taken was deemed an effective method for the task. Accurate predictions of bike rental demand can be produced by this model which can be used in a real-world scenario to better inform decision making highlighting the importance of AI techniques in transport applications.

7 References

Google Developers, 2024. *Overfitting*. [Online]
Available at: https://developers.google.com/machine-learning/crash-course/overfitting/overfitting
[Accessed 30 04 2025].

IBM, n.d. *What is random forest?*. [Online]
Available at: https://www.ibm.com/think/topics/random-forest
[Accessed 30 04 2025].

Mucci, T., 2024. *IBM | What is data leakage in machine learning?*. [Online]
Available at: https://www.ibm.com/think/topics/data-leakage-machine-learning
[Accessed 30 04 2025].

Murel, J. & Kavlakoglu, E., 2023. *IBM | What is multicollinearity?*. [Online]
Available at: https://www.ibm.com/think/topics/multicollinearity
[Accessed 30 04 2025].