Do we need more bikes?
Project in Statistical Machine Learning

Chen Gu, Saba Yousefinasabnobari, Seyedehmoniba Ravan, Nora Derner
Group 29

Abstract

In this project, we delve into the binary classification challenge of predicting bicycle demand in Washington D.C. Utilizing a robust dataset of 1600 instances, our analysis encompasses the training and fine-tuning of various classifiers, including Logistic Regression, KNN, QDA, LDA, Random Forest, and Bagging. We emphasize understanding the mathematical underpinnings of these models and strategically tuning their hyperparameters. Each model is rigorously evaluated using metrics such as accuracy, F1-score, and ROC, within a cross-validation framework. Our comparative analysis identifies Random Forest as the most effective classifier. **Number of Group Members: 4**

1 Introduction

The District Department of Transportation in Washington D.C. wants to find out whether the number of available rental bikes needs to be increased under certain weather and time conditions. Based on the provided training dataset, a binary classifier is trained that best predicts whether the stock should be increased or not.

2 Data analysis

The data set consists of 1600 data samples, with each sample containing 16 different categorical or numerical characteristics. The **numerical variables** represent the quantitative data on the time of day (time, day of the week, month) and quantitative data describing the weather (temperature, dew, humidity, precipitation, snowdepth, windspeed, cloudcover, visibility). The **categorical variables** indicate in binary form whether it is a public holiday, weekday or summer day, whether there is snow or not and whether there is high or low demand for bicycles.

Table 1: Statistical values for numerical weather-dependent data

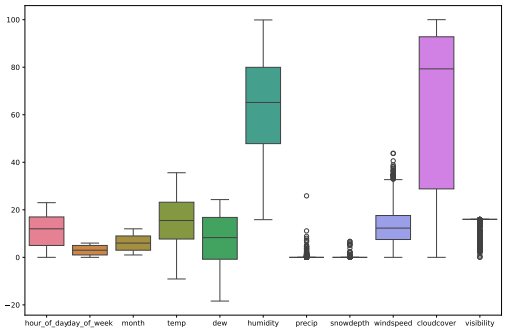
	temp	**dew**	**humidity**	**precip**	**snowdepth**	**windspeed**	**cloudcover**	**visibility**
count	1600.0	1600.0	1600.0	1600.0	1600.0	1600.0	1600.0	1600.0
mean	15.21	7.75	63.93	0.12	0.04	13.08	64.32	15.34
std	9.26	10.03	19.08	**0.92**	**0.42**	7.76	**32.75**	2.32
min	-9.1	-18.4	15.85	0.0	0.0	0.0	0.0	0.1
50%	15.5	8.3	65.18	0.0	0.0	12.3	79.3	16.0
max	35.6	24.3	99.89	25.87	6.71	43.8	100.0	16.0

Table 1 shows a part of the **basic statistics** for the **numerical features**. All our features show complete data sets. There are no missing values in the data, as can be seen in the first row of tables. Also, it shows high values in the standard deviation of `temp` and `cloudcover`, among others. For example, the temperature values vary widely around the mean value, so there is a large fluctuation in temperature values. The standard deviation of `snowdepth` is 0.04, which means that there is no

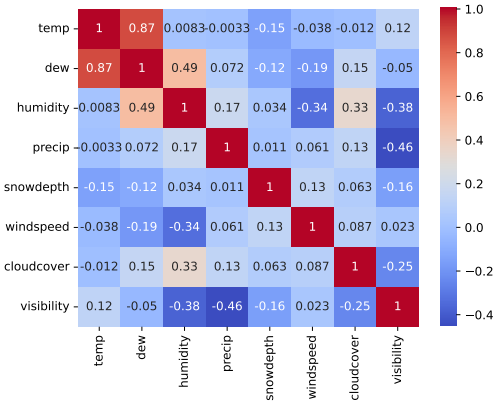
variation in the data. This observation can be supported by the findings from the boxplot in Figure 1a. Here the distribution of all numerical data is shown in comparison. We recognize the spread in the width of the box as with `cloudcover` with simultaneous asymmetrical distribution of the measurements. Furthermore, we can detect the outliers in the data, e.g. in the precipitation feature.

Figure 1b shows the correlation matrix of the weather-dependent numerical features to investigate the linear relationship between the variables. The correlation coefficient of 0.87 shows a strong positive linear relationship between `temp` and `dew`. Furthermore, there is a moderate dependency between `humidity` and `dew`, characterized by a correlation coefficient of 0.49. All other coefficients are close to zero. This means that there is little dependence between them.

Figure 1: Analysis of the numerical features



(a) Boxplot of the numerical features



(b) Correlation matrix of the weather-dependent numerical features

For the **categorical features**, we examined the distribution and identified that class imbalances are present for all five categorical features, as shown in Table 2. In particular, for our target variable `increase_demand`, we find that only 18% of the data measurements predict high demand for bicycles. Consequently, we have to make sure that the classifiers are not specialized to the more frequent class later on during modeling.

Table 2: Frequencies for categorical data

	holiday	**weekday**	**summertime**	**snow**	**increase_stock**
0	1547	464	570	1600	**1312**
1	53	1136	1030	0	**288**

To identify **trends** in the data on the increase in bicycle demand, we group the data to recognize patterns. Two possible perspectives are shown in Figure 3a and 3b in the Appendix. First, we summarize time-dependent data, such as time of day and days of the week. We recognize an increase in demand during the day and on weekends the demand is already higher before noon than on weekdays. Figure 3b shows the bicycle demand as a function of the weather-specific characteristics of temperature and cloud cover. There is a significant increase in demand as the temperature rises, but no real pattern when looking at the cloud cover. By performing a chi-square test, we investigated whether there is a significant **association** between the feature `holiday` or `weekday` and `increase_stock`. We came to the conclusion that `holiday` and `increase_stock` are independent of each other (chi-square value: 0.00021, p-value: 0.98) and that there is a significant association between `weekday` and `increase_stock` (chi-square value: 21.03, p-value: 4.51).

3 Model development

3.1 Classification methods

Based on the data analysis, we decided to remove the features `snow` and `snowdepth` due to their low relevance. Furthermore, we performed a label coding to assign the numerical value 0 for the category `low_bike_demand` and 1 for `high_bike_demand` to the target variable. Then we normalized the feature using the `standard scaler` class from the sklean library. To enlarge the dataset and to remove the class imbalance, we tried different resampling methods, over- and undersampling, but they did not have a significant positive impact on the performance of our classifiers. We established a benchmark by implementing a model called the naive model, which consistently predicts high demand.

3.1.1 Logistic Regression

Mathematical principle of Logistic Regression The logistic regression uses the logistic sigmoid function to calculate the probability that the given input instance x_i belongs to a certain class (high_bike_demand or low_bike_demand). It is a modified version of the linear regression model by adding the logistic sigmoid function (1), which only takes values between 0 and 1. The output is a value in the interval $[0, 1]$. If the output value is above the decision threshold (e.g. $r = 0.5$), the class m = 1 (high_bike_demand) is predicted.

$$g(x_i) = \frac{1}{1 + e^{-\theta^T x_i}} \tag{1}$$

Using the training dataset, the parameters are trained so that the model best reflects the given data. We have used 13 input features to retain all available information and avoid underfitting. Each feature is weighted differently until the best linear decision boundary between the two classes is found. Mathematically, we look for those parameters $\hat{\theta}$ that minimize the cost function, the negative log-likelihood function. (2) This optimization problem is solved using a numerical method such as gradient descent or Newton's method. In order to prevent overfitting to the training data, we added an L2-regularization term to the log-likelihood function, which also ensures that there is a unique solution to the optimization problem.

$$\hat{\theta} = \arg\min_{\theta} \frac{1}{n} \sum_{i=1}^{n} \ln\left(1 + \exp\left(-y_i \theta^T x_i\right)\right) + \lambda \|\theta\|_2^2 \tag{2}$$

Hyperparameter tuning Using the `GridSearchCV` function from the Python library scikit-learn, we found the optimal hyperparameters for model generation by trying out different combinations of parameters through cross-validation. We tried the different regularization terms L1-, L2- in combination with 20 different logarithmic distributed regularization parameter values from 0.0001 to 1000. Eventually, we selected the L2-regularization term with the inverse regularization strength of $C = 0.336$. The small C as a quantitative input value ensures that the regularization is strong and by adding the penalty term to the cost function, the θ parameters are kept as small as possible.(2) We also varied between different optimization algorithms, i.e. stochastic mean gradient descent, linear SVC and the Newton gradient method. We chose the solver liblinear as qualitative input as it works well in combination with L2 regularization, especially for input data with many features and small data sets. The optimal number of iterations for the solver was selected using the `GridSearchCV` method as $max\_iter = 1000$ from the range of [1000, 1500, 2000]. The solver could not converge to a solution within the iterations with a smaller number of possible maximum iterations. The last parameter we tried out was whether the classes should have different weights based on the class frequencies when training the model. In our use case, this did not lead to an improvement in the results.

3.1.2 Discriminant analysis: LDA, QDA

Mathematical principle Both Linear Discriminant Analysis (LDA) and Quadratic Discriminant Analysis (QDA), are derived from Bayes Theorem. LDA and QDA assume the predictor variables X of each class follow a multivariate Gaussian distribution, $x \sim \mathcal{N}(\mu, \Sigma)$, where Σ_k is a covariance

matrix for the kth class. So, based on these assumption, the Bayes classification in LDA and QDA becomes for different class values of C_k, find the maximum of the product of the prior probability and the probability of observing the features given that class.

$$\hat{C} = \arg\max_{C_k} P(C_k) \prod_{i=1}^{n} P(x_i|C_k) = \arg\max_{C_k} f_k(x)\pi_k \tag{3}$$

For LDA: The covariance matrices $\Sigma_1 = \cdots = \Sigma_{|k|}$ are assumed to be equal for each of the k classes in LDA. Based on the covariance matrices assumption of LDA and (3), we may derive an approximated LDA discriminant score $\hat{\delta}_k(x)_{lda}$. thus we can determine whether the likelihood belongs to class k according to the prediction variable. And as we can see it is a linear function of x, which matches the linear decision boundaries produced by LDA.

$$\hat{\delta}_k(x)_{lda} = x \cdot \frac{\hat{\mu}_k}{\hat{\sigma}^2} - \frac{\hat{\mu}_k^2}{2\hat{\sigma}^2} + \log(\hat{\pi}_k) = \omega_k^t x + \omega_{k0} + Cst. \tag{4}$$

For QDA: We make the assumption that each of the k classes may have different variance between each class, $\Sigma_1 \neq \cdots \neq \Sigma_{|i|} \neq \cdots \neq \Sigma_{|k|}$. Than we can obtain estimated QDA discriminant score $\delta_k(x)_{qda}$ from (3) and the assumption of QDA, which leads to the quadratic decision surface that QDA produces.

$$\delta_k(x)_{qda} := -\frac{1}{2}\ln(|\Sigma_k|) - \frac{1}{2}(x-\mu_k)^T\Sigma_k^{-1}(x-\mu_k) + \ln(\pi_k) + Cst. \tag{5}$$

After computing the discriminant score with (4) and (5) for each class based on the model's parameters and the input features of an observation, the class with the highest discriminant score is then chosen as the predicted class for that particular observation.

Hyperparameter tuning We decided to use the **LDA** model in conjunction with a support vector classifier (SVC) and tuned the hyperparameters for both the LDA and SVC models using the grid search method. The first hyperparameter in LDA is 'SVD' (Singular Value Decomposition) as qualitative input, which is used for dimensional reduction and for determining the linear combinations of features that best separate the classes. The second hyperparameter is 'tol' (tolerance). Grid search method suggests 0.00001, since a lower value of 'tol' will potentially capture more information based on our small data set. To use the support vector classifier method, we first change the train data by fitting it with the best LDA hyperparameters. Then, we use that to fit the SVC model. SVC focuses on finding a decision boundary that best separates different classes by maximizing the margin between the sample points and the hyperplane. The first hyperparameter for SVC is C, which is a quantitative input and can be interpreted as a penalty parameter of the error term. Grid search method chooses the high value of $C = 100$. This will allow the model to prioritize correctly classifying as many training samples as possible. The second hyperparameter is γ, which is also a quantitative input. The selected value is $\gamma = 1$, which is quite high due to the small size of our data set, resulting in an accurate fit to our training data. Finally, the grid search method selected the qualitative 'kernel' parameter as 'rbf', which is used to separate the data correctly. **QDA:** For the first hyperparameter 'reg_para', we selected a value of 0.89 for our model using the gridsearch method. A high value of this parameter can increase the stability of the model. The second hyperparameter is 'tol' (tolerance). We select a quite small tolerance 0.0001 for the QDA model in order to get the majority of the information in the covariance matrix.

3.1.3 K-nearest neighbor

Mathematical principle The k-Nearest Neighbors (k-NN) algorithm measures the distance between a test data point and all training data points using one of several distance metrics, such as Euclidean, Manhattan (6), or Minkowski. The Manhattan Metric is often known as the taxicab or city block distance. It focuses on the sum of the absolute differences between the coordinates. This approach is frequently representing the path one would take from one location to another. For two points the Manhattan distance is given by:

$$D_{\text{Manhattan}}(P,Q) = \sum_{i=1}^{n} |p_i - q_i| \tag{6}$$

Hyperparameter tuning A crucial aspect of k-NN is the selection of 'k', the number of closest training data points or 'neighbors', which significantly influences the model's performance. A small 'k' value may render the model sensitive to noise, leading to overfitting, whereas a larger 'k' could oversimplify the model, risking underfitting. Furthermore, the algorithm's performance deteriorates with increasing feature count, which complicates accurate distance measurement in high-dimensional spaces. Preprocessing steps, such as data normalization, are crucial to ensure the uniform contribution of all features to distance calculations. In optimizing our k-NN model we applied a multifaceted approach. Initially, we used a grid search to determine the optimal 'k' value as a quantitative input and distance metric, finding that a 'k' of 7 with the Manhattan metric was most suitable for our dataset. Further enhancing accuracy, we integrated Principal Component Analysis (PCA), which effectively reduced our dataset's dimensionality. PCA concentrated on the most informative features, thereby not only speeding up computation but also improving the k-NN model's accuracy by eliminating noise and irrelevant features.

3.1.4 Bagging and Random Forest

Mathematical principle Bagging, short for bootstrap aggregating, is a powerful ensemble learning technique that aims to enhance predictive performance by training multiple models on different subsets of the training data. The final prediction is obtained by aggregating the predictions of the trained model on B bootstrap samples. In the context of this project, we employ bagging with the decision tree as our base model. The prediction associated with each region is determined through a majority vote mechanism:

$$y_\ell = \text{MajorityVote}(y_i : x_i \in R_\ell) \tag{7}$$

The objective is to identify the split that minimizes impurity or disorder within the child nodes:

$$\arg\min(n_1 Q_1 + n_2 Q_2) \tag{8}$$

Impurity, measured using a criterion such as Gini impurity or entropy, is denoted as Q. Random Forest, an extension of the bagging technique, introduces an additional layer of randomness by considering only a random subset of features at each split point. This feature randomness helps Random Forest to be more robust and less prone to overfitting compared to a standard bagging approach with decision trees.

Hyperparameter tuning The hyperparameters for the Decision Tree are: 1. class_weight: (Qualitative) Mitigates class imbalance by adjusting weights based on class frequencies. 2. criterion: (Qualitative) Dictates the impurity measure for split quality. 3. min_samples_leaf: (Quantitative) Specifies the minimum samples for a leaf node 4. min_samples_split: (Quantitative) Sets the minimum samples required to split an internal node. and for Random Forest are 5. bootstrap: (Qualitative) Determines if bootstrap samples are used. 6.n_estimators: (Quantitative) Specifies the number of trees. The selected hyperparameters (`'class_weight': 'balanced', 'criterion': 'entropy', 'min_samples_leaf': 3, 'min_samples_split': 2`) for decision trees, and (`'bootstrap': True, 'n_estimators': 100`) for Random Forests, reflect our considerations for addressing class imbalance, impurity measurement, and control over tree structure. 'Balanced' class weight is chosen to handle imbalanced data, 'entropy' is selected for its suitability to specific datasets, and a low value like 3 for 'min_samples_leaf' captures finer details, balancing with the risk of overfitting.

We underscore the importance of **feature selection**, strategically removing less significant columns or using the most important ones based on insights gained from a feature importance analysis using a Random Forest. In the Appendix Figure 4, we provide a visual representation of the most important features identified through the Random Forest analysis. We identified `hour_of_day` as the most crucial feature, exerting a high impact on model predictions. Following closely, `temp` demonstrated moderate importance, significantly contributing to predictive power. While `humidity` played a role of lower importance, it still contributed to overall model performance. Other features, `precip` and `holiday` were identified as the least important for our model.

3.2 Evaluation of performance

In assessing the performance of our classifiers, we employed cross-validation with $k = 5$ to ensure robustness in our results.

Logistic Regression The logistic regression classifier classifies on average 84.75% of the instances correctly. Furthermore, on average, about 67.57% of the 'high_demand' predicted instances actually have a high demand. The classifier has difficulty correctly recognizing the instances of high demand (33.05% Recall). Overall, the logistic classifier has an acceptable but not excellent performance. This could be due to the fact that the classifier assumes a linear relationship between the features and the `increase_stock` feature, which is not predominant in our data set.

LDA & QDA The mean accuracy of QDA 83.06% is slightly less than LDA 84.75%, which suggests that QDA is less effective on training data; The recall of QDA 51.14% is higher than LDA 31.71%, indicating that QDA is better at finding actual positive cases. However, the overall score of these two methods is not good enough. The suboptimal performance of LDA and QDA could be due to the mismatch between model assumptions and the actual data characteristics, so these two methods are not our perfect choice as well.

KNN In evaluating the performance of our k-NN model, we observed a mean accuracy of 84.31%, indicating a capability to classify instances correctly. However, the model exhibited a moderate recall rate of 41.58%, suggesting it missed a significant number of actual high-demand instances. The precision of the model was recorded at 59.64%, reflecting that while it is fairly reliable in predicting high-demand instances, there is still a noticeable proportion of false positives.

Bagging and Random forest In the domain of ensemble methods, both Random Forest and Bagging with Decision Tree deliver robust performances. Random Forest achieves an impressive 89.63% accuracy and the F1-Score of 71.53% reflects its balanced performance, making Random Forest a robust and top-performing algorithm for this dataset. Bagging with Decision Tree achieves an 89.06% accuracy, with 69.24% precision and 70.70% recall, demonstrating reliable classification. It merges as a strong performer, rivaling Random Forest and outperforming other evaluated methods.

3.3 Model selection and comparison

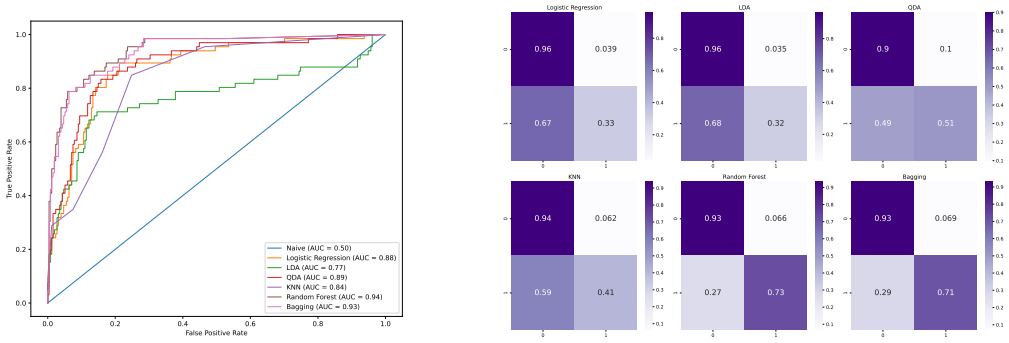
Table 3: Comparison of performance metrics

Model	Accuracy	Precision	Recall	F1-Score
Naive	0.180000	0.180000	1.000000	0.304778
KNN	0.843125	0.596490	0.415834	0.488278
Logistic Regression	0.847500	0.675795	0.330560	0.437861
LDA	0.848750	0.679552	0.317125	0.428619
QDA	0.830625	0.528221	0.511445	0.515346
Random Forest	**0.896250**	**0.719086**	**0.724407**	**0.715306**
Bagging	0.890625	0.692423	0.707044	0.698177

As shown in Table 3 and Figure 5 in the Appendix, Random Forest has the highest accuracy of 89.62%, so we expect this classifier to be the most reliable at distinguishing between high and low bike demand. All other models also show solidly similar accuracy values in the range of 83.06% to 89.06%. Random Forest has the highest precision with 71.91%, which means that this classifier is also particularly accurate in predicting high bike demand and minimizes false predictions. QDA has the lowest precision value, which could mean that it could potentially make incorrect predictions for high bike demand. Random Forest is also promising because of its very high recall value of 72.44%. This large recall value is important to ensure that a high demand for bicycles is actually recognized. Logistic regression and LDA in particular are therefore less sensitive. Our naive model, which always predicts high demand, i.e. recognizes all actual high demand, correctly has a recall of 100%. Random Forest also has the highest value for the metric F1 Score, which means that a good balance between precision and recall is achieved for predictions of high bike demand. The Logistic Classifier and LDA do not have a robust performance due to the low values of 43.78% and 42.86%.

The **ROC curve** is another way of comparing the performance of the classifiers shown in Figure 2a. The larger the area under the curve, the better the performance due to the high sensitivity and high specificity. The Random forest has the highest value with 94.00% and the Bagging classifier the second highest with 93.00%. What is remarkable about the LDA curve is that it shows a bend at the

Figure 2: Comparison Confusion Metrics and ROC-Curves



(a) ROC-Curves

(b) Confusion Matrix

threshold value 77.00%, i.e. it shows weaknesses in this area, and that KNN shows a very constant performance, visible through fewer bends in the straight line.

The **confusion matrix** is another very useful way to describe the performance of classification models, as seen in Figure 2b. Logistic Regression, LDA has higher false negative values (67.00% and 68.00%), which suggests that they are more likely to predict low bicycle demand than high demand. The QDA has a more a balanced score between false negative values 49.00% and true positive values 51.00%, but it still missed about half of the high demand predictions. KNN and Bagging methods have a better balanced overall score compared to the previous three methods. But we noticed that the Random Forest model had the highest true negative rate 73.00%, and the lowest false negative rate, this indicates that Random Forest is relatively more efficient at correctly identifying instances of low demand and is less likely to miss high-demand instances compared to the other models.

4 Conclusions

In order to make a statement about the required bicycle stock under certain time and weather conditions based on the given data set, six different classifiers were modeled and trained after comprehensive pre-processing of the data. This involved systematically analyzing the data set and optimizing the classifiers by selecting hyperparameters. After training the classifiers together under cross-validation with the identical training dataset, the classifiers were compared using metrics such as accuracy, precision, recall, and F1-score. The results show that Random Forest performs excellently across all metrics and is particularly strong in identifying times with high cycling demand. We decided to send this classifier 'into production' for the test dataset. LDA, KNN, Logistic Regression, and Bagging also show solid performance in the evaluation but still have the potential for improvement.

References

[1] Lindholm, Andreas & Niklas Wahlström & Fredrik Lindsten & Thomas B. Schön. (2022) Machine Learning: A First Course for Engineers and Scientists. 1st ed. Cambridge University Press, 2022. DOI: 10.1017/9781108919371 .

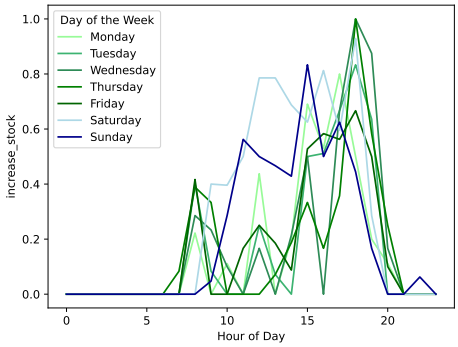
[2] Ghojogh, B. & Crowley, M. (2019) Linear and quadratic discriminant analysis: Tutorial. arXiv preprint arXiv:1906.02590.

[3] Rosipal, R. & Trejo, L.J. & Matthews, B. (2003) Kernel PLS-SVC for linear and nonlinear classification. In Proceedings of the 20th International Conference on Machine Learning (ICML-03) (pp. 640-647).

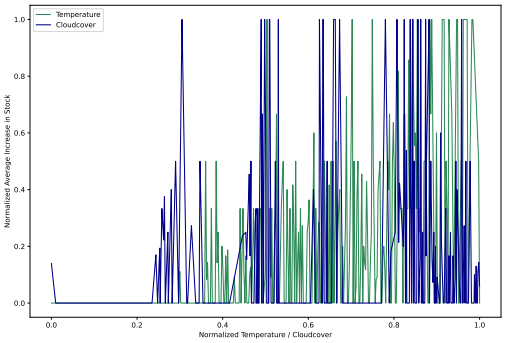
A Appendix

A.1 Appendix Data Analysis

Figure 3: Trends in bicycle demand



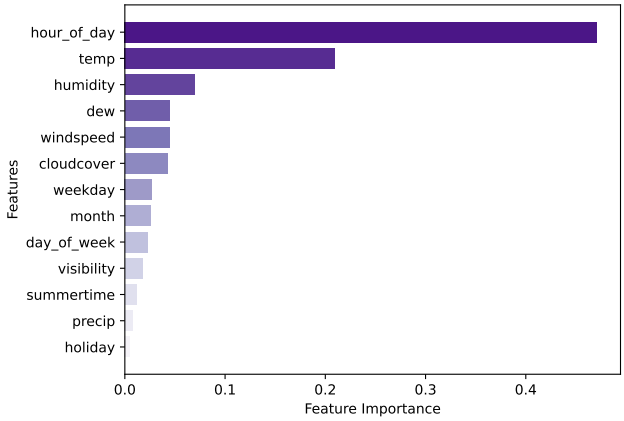
(a) Average Increase Stock by Hour of Day, Day of Week, and Month



(b) Average Increase Stock by Temperature and Cloud cover

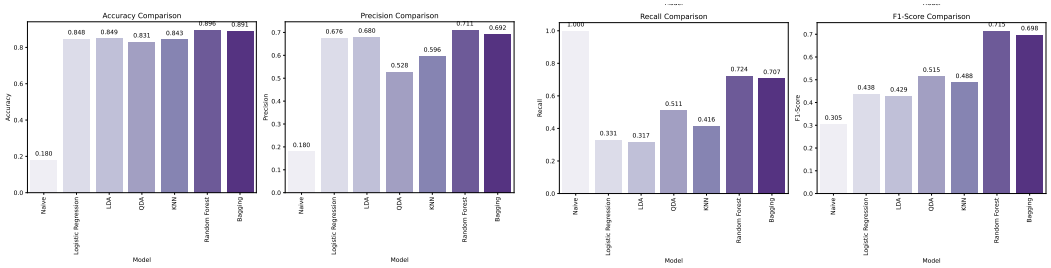
A.2 Appendix Random Forest

Figure 4: Feature Importance



A.3 Appendix Model Selection and Comparison

Figure 5: Comparison of evaluation metrics



(a) Accuracy and Precision

(b) Recall and F1-Score