
Do we need more bikes?

Project in Statistical Machine Learning

Erik Björk

Simona Stoyanoska

Olov Rahm

Jakob Hanson

Abstract

In this project we develop, and study different statistical machine learning models for predicting whether the number of available bikes at a given hour should be increased, a project by the District Department of Transportation in Washington D.C. The training data set consists of 1600 instances of hourly bike rentals, and a test set of 400 instances. The models for prediction we have used are: *Logistic regression*, *Discriminant methods: LDA, QDA*, *k- Nearest Neighbour*, and *Tree Based Methods*. We have found that *k- Nearest Neighbour* gives best prediction, with accuracy 92%.
The group consists of 4 students.

1 Introduction

Statistical machine learning is a subject that aims to build and train algorithms, that analyse large amount of data, and make predictions for the future, which are computed by using established statistical models, and tools from functional analysis. This is a project in supervised, statistical machine learning, where several models were created, and trained, in order to analyse which one of them gives best prediction for the project "Do we need more bikes", where we want to understand, and predict if there is a high, or low demand of city bikes in the public transportation of Washington, a project by the District Department of Transportation in Washington D.C..

The data set used for training our models, consist of 15 variables, containing quantitative/qualitative data. We developed several models, and evaluated them with cross-validation, in order to understand which algorithm gives the best prediction.

2 Theoretical Background

Here we review the theoretical background of the models. We follow mostly [1].

Let $\{\mathbf{x}_i, y_i\}_{i \in \mathbb{N}}$ be the training data set, where \mathbf{x}_i is a matrix representing the input, and y_i is the output.

2.1 Mathematical Overview of the Models

2.1.1 Logistic Regression

The backbone of logistic regression is linear regression, i.e. finding the least-squares solution to an equation system

$$X\theta = y$$

given by the normal equations

$$X^T X \theta = X^T y$$

where X is the training data matrix, θ is the coefficient vector and y is the training output. The parameter vector is then used in the sigmoid function $\sigma(z) : \mathbb{R} \rightarrow [0, 1]$

$$\sigma(z) = \frac{e^z}{1 + e^z} :$$

where $z = x^T \theta$, and x is the testing input. This gives a statistical interpretation of the input vector. In the case of a binary True/False classification, the value of the sigmoid function then determines the class. The model was tuned using the `lbfgs` algorithm, which is standard in *SKLearn*. It optimizes the weights θ_i to minimize the log-loss function:

$$J(\theta) = -\frac{1}{m} \sum_{i=1}^m [y_i \log(h_\theta(x_i)) + (1 - y_i) \log(1 - h_\theta(x_i))] + \frac{\lambda}{2} \|\theta\|^2, \quad (1)$$

where λ is the regularization strength (default for *SKLearn*: $\lambda = 1.0$), h_θ is the sigmoid function for each of the m input vectors x_i , and θ is the parameter vector being optimized. It is being optimized through a gradient method.

2.1.2 Random forest

The random forest method is a based upon decision trees, i.e. dividing the data point into binary groups based on Gini-impurity, entropy or classification error, Gini being the most common. These divisions are then used to create a binary tree shown in figure ??Tree) and where the leaf-nodes are used to classify the target variables based on the input. As of itself the decision tree tends to have unsatisfying results which leads to methods like random forest and sandbagging that boost its accuracy. Sandbagging is a way to sample the data in order to get multiple datasets from the same data. One then creates a decision-tree for every subset data to then combine them into one model. This lessens the variance of the model but increases bias. This means that sandbagging can increase false negatives which in this application makes it nonviable. Random forest on the otherhand is viable, it creates multiple trees while discarding random input variable this randomness decreases overfitting creating a more robust model.

2.1.3 Non-parametric method: k-Nearest Neighbour

k-Nearest Neighbour (k -NN) is a distance based method that takes a k amount of points from the training data set, called *neighbours*, computes the distance between them, then assumes that the predicted value $\hat{y}(x_*)$ follows the trend of the k -nearest neighbours. Since k -NN uses the training data explicitly it is also called a *nonparametric* method.

The k -NN method can be divided into several subcategories, inter alias *classification* k -NN method, *regression* k -NN method. In this project, we are using the classification method, since we are trying to predict in which of the two classes low, or high demand, the given, and predicted data points belong.

The classification k -NN algorithm evaluates $\hat{y}(x_*)$ by computing the most frequently occurring class among the k nearest neighbours. Here, we try to identify whether a data point belong to the high demand-class. Denote c = high demand class. For simplicity, assume Euclidean ambience. Then

$$\hat{y}(x_*) = \arg \max_c \sum_{n \in \mathbb{N}} \chi_{(y_n=c)},$$

where y_i is the class of the nearest neighbour, χ is the characteristic function

$$\chi_{(y_i=c)} = \begin{cases} 1 & \text{if } y_n = c, \\ 0 & \text{otherwise.} \end{cases}$$

It is very common to use a weighted sum to predict the next value, i.e.

$$\hat{y}(x_*) = \arg \max_c \sum_{n \in \mathbb{N}} \frac{\chi_{(y_n=c)}}{d(x, x_n)},$$

where d is the standard Euclidean metric, computing the distance between an input x , and a neighbour x_n .

When using this model it is important to choose an optimal k -value. There are several tests for this, here we implement *uniform weighting*, and *distance weighting*. The first algorithm creates a k -NN model for each new $k \in [1, 500]$, and trains the model with uniform weights, i.e. the contribution of all neighbours is equal. Similarly, the latter trains a k -NN classifier for each $k \in [1, 500]$, with the difference that it uses distance based weighting, i.e. closer neighbours have greater influence. After testing different upper boundaries for k , the two models gave good results in the interval $[1, 500]$, see Figure 1. From the figures, we can see that the second test gives a better value for k , since the plot follows smoother trend, in comparison to the uniform weighting test, which makes it easier to identify an optimal k value ($k = 120$). Moreover, the distance weighting algorithm is providing results for larger values of k , that is for $k \in [1, 400)$ before the curve converges, while the uniform weighting algorithm converges earlier, when $k = 120$. This means that for large k , both test algorithms make prediction based on the most common class in the data set, instead of making prediction based on the behaviour of the neighbours. Thus for sufficiently large k , for any given data point, the model will consider unnecessarily large amount of neighbours, and the prediction will be evaluated to belong to the most frequent class. Since the distance weighting has a larger range of k -value, it should be more trustworthy.

When $k = 120$, the accuracy of the model is 92%.

2.1.4 Discriminant analysis: LDA and QDA

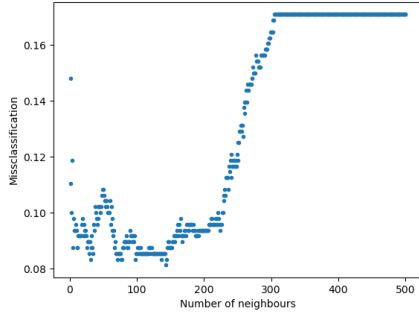
Linear Discriminant Analysis is a generative model, which means it is a model that's creating and using a probability distribution $P(\mathbf{x}, y)$ to create an estimation for the probability $P(y = m | \mathbf{x})$ using bayes theorem.

Bayes theorem is:

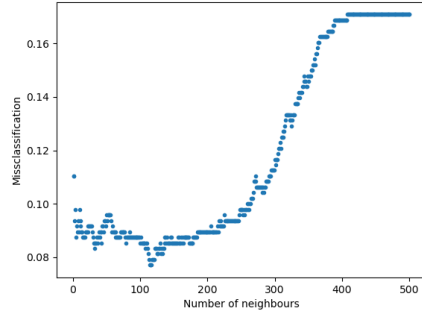
$$p(y | \mathbf{x}) = \frac{p(y, \mathbf{x})}{p(\mathbf{x})} = \frac{p(y)p(\mathbf{x} | y)}{\int_y p(y, \mathbf{x})}$$

For the discrete version it is obtained:

$$p(y = m | \mathbf{x}) = \frac{p(y = m)p(\mathbf{x} | y = m)}{\sum_{m=1}^M p(y = m)p(\mathbf{x} | y = m)}$$



(a) Uniform distance test for k .



(b) Weighted distance test for k .

Figure 1: Test for choosing an optimal k -value.

For this form of the equation to be useful, it is necessary to obtain an accurate estimation of $p(y = m)$ and $p(\mathbf{x}|y = m)$ for all classes m .

In LDA, $p(y = m)$ is estimated by counting the percentage of data points (in the training data) being in each of the classes and using that percentage as the probability of a data point being in that class. In mathematical terms:

$$p(y = m) = \frac{1}{n} \sum_{i=1}^n \mathbb{I}\{y_i = m\} = \frac{n_m}{n}$$

To estimate the probability distribution $p(\mathbf{x}|y = m)$, a multi-dimensional gaussian distribution is used:

$$\mathcal{N}(\mathbf{x}|\mu, \Sigma) = \frac{1}{(2\pi)^{d/2} |\Sigma|^{1/2}} \exp\left(-\frac{1}{2}(\mathbf{x} - \mu)^T \Sigma^{-1}(\mathbf{x} - \mu)\right)$$

Where \mathbf{x} is the d -dimensional data point, μ is the (d -dimensional) mean of the random variable. Σ is the symmetric, positive definite covariance matrix defined by:

$$\Sigma = \frac{1}{n - M} \sum_{m=1}^M \sum_{i:y_i=m} (\mathbf{x}_i - \mu_m)(\mathbf{x}_i - \mu_m)^T$$

Using these estimations results in an expression for the quantity $p(y = m|\mathbf{x}) \forall m$. LDA then uses maximum likelihood to categorize an input \mathbf{x} into a class m .

Quadratic discriminant analysis (QDA) is heavily based of LDA with the sole difference being how the covariance matrix Σ is created. In LDA, the covariance matrix is assumed to be the same for data in each and every class. In QDA however, the covariance matrix is calculated for each class as follows:

$$\Sigma_m = \frac{1}{n_m - 1} \sum_{i:y_i=m} (\mathbf{x}_i - \mu_m)(\mathbf{x}_i - \mu_m)^T$$

To optimize LDA and QDA, it's all about estimating Σ_m and μ_m for all classes m . The mean μ is estimated as the mean of all the input data $\mu = \frac{1}{n} \sum_{i=1}^n \mathbf{x}_i$. The covariance matrix Σ_m is defined previously in the section.

One thing to note about LDA and QDA is that the use of a multi-variable gaussian distribution benefits normally distributed variables. In this project however, there is a dependance on positive definite values which are not normally distributed by nature. This is an issue when using QDA since in the class of *high_bike_demand*, all data points have a snow depth of 0 and has hence no variance. This results in this class having a undefined inverse for the covariance matrix. The solution used was to exclude this variable from this model.

2.2 Input Data Modification

By plotting the data and analyzing the .csv file, some observations were made. The different inputs were then changed accordingly:

- *Kept as-is*: weekday, windspeed, visibility, temp
- *Modified*:
 - month - split into two inputs, one cosine and one sine part. This make the new inputs linear and can follow the fluctuations of the year. The original input was discarded.
 - hour_of_day - split into three boolean variables: demand_day, demand_evening, and demand_night, reflecting if the time was between 08-14, 15-19 or 20-07 respectively. This was done because plotting the data showed three different plateaus of demand for the different time intervals. The original input was discarded.
 - snowdepth, precip were transformed into booleans, reflecting if it was raining or if there was snow on the ground or not. This was done as there was no times where demand was high when it was raining or when there was snow on the ground.
- *Removed*: cloudcover, day_of_week, snow, dew, holiday, summertime. These were removed due to being redundant (e.g. summertime), not showing a clear trend (e.g. cloudcover), giving a worse score when used, or all three (e.g. day_of_week).

3 Data Analysis

In the given data, there are some numerical and categorical features:

- *Numerical*: temp, dew, humidity, precip, snow, snowdepth, windspeed, cloudcover and visibility.
- *Categorical*: hour_of_day, day_of_week, month, holiday, weekday, summertime, and increase_stock

There are some trends seen in the data when it comes to time and weather. From figure 2, one can see a periodic relationship for the months, where there is a higher demand during the warmer months, loosely following a trigonometric curve. Over the week, the demand is rather stable, with a peak on the weekend, especially Saturdays.

Looking at the weather (figure 3); if there is rain or if there is snow on the ground, there is close to always low demand. Cloudcover did not make a big impact, which is also intuitive, as a cloudy day does not make biking more difficult. Dew point also does not have a clear trend, while humidity however has a clear trend downwards as the humidity increases. Temperature had a more clear impact, where more people wanted to bike the warmer it got.

The overall trend is that about one eighth of observations correspond to a high bike demand. During the night, or in bad weather, the demand is (intuitively) low. But during rush hour (figure 2c), the demand is very high, and should probably be increased in order to minimize excessive CO₂ emissions.

4 Result

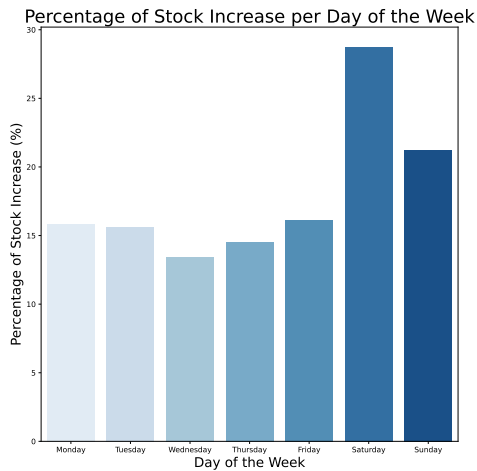
The method used to evaluate the different models where chosen to be the accuracy as well as the precision and recall of the class "high bike demand". The accuracy is defined simply as:

$$\text{Accuracy} = \frac{n_{\text{correct}}}{n_{\text{tot}}}$$

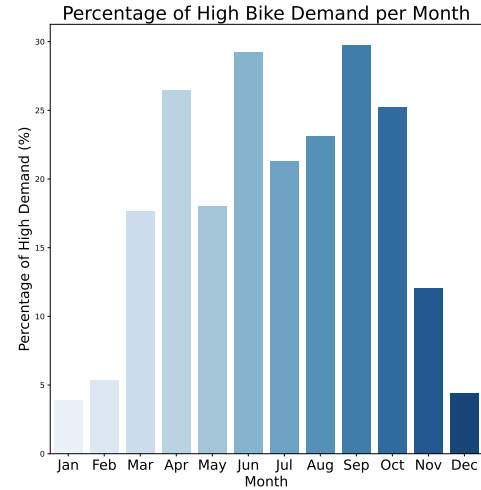
And the precision and recall is defined as:

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

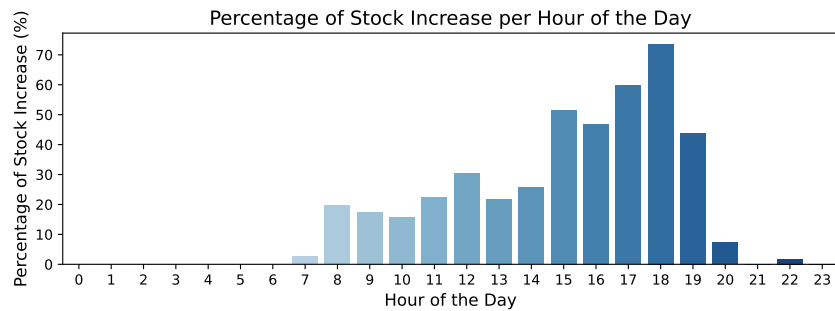
Furthermore, a naive model that only guessed there is a low demand was compared to the rest of the models. The different models were tested and the accuracy where:



(a) Demand per day of week.



(b) Demand per month.



(c) Demand per hour of day.

Figure 2: Bike demand vs. day of week and month.

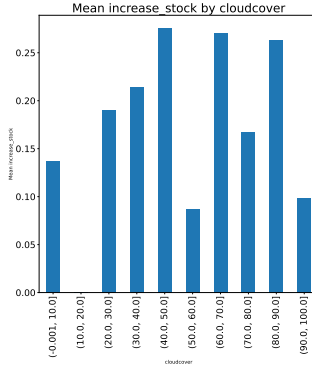
Accuracy of the models			
Model	Accuracy	Precision	Recall
LDA	85%	53%	50%
QDA	87%	67%	36%
k-nearest neighbour	92%	81%	70%
Random Forest	91%	77%	71%
Logistic Regression	90%	73%	63%
Naive	83%	0%	0%

Here you can clearly see random forest and k-nearest neighbour are the best classifiers both outperforming linear and quadratic regression on accuracy, precision and recall. Out of random forest and kNN the group would proceed with the kNN method, its higher accuracy and precision score outweighing the slightly better recall score of random forest. This will mean a slight loss in income caused by increasing false negatives but is thought to be covered by fewer false positives.

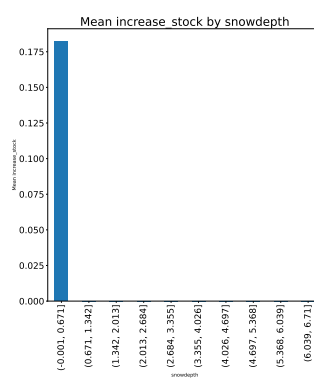
5 Conclusion

From the evaluation the models, k-nn performed the best with the highest accuracy and precision. As for the recall, k-nn did not perform the best but is considered adequate and hence this method is chosen as the best one.

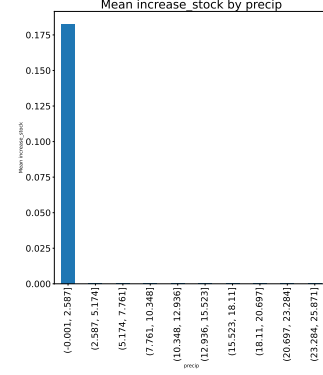
One reason for the discriminant analysis falling short of the other models is likely due to



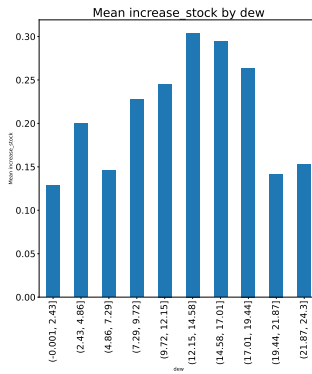
(a) Demand per cloudcover (percentage).



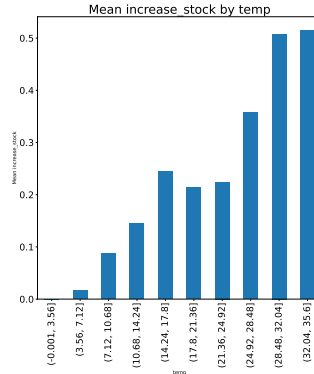
(b) Demand per snowdepth (mm).



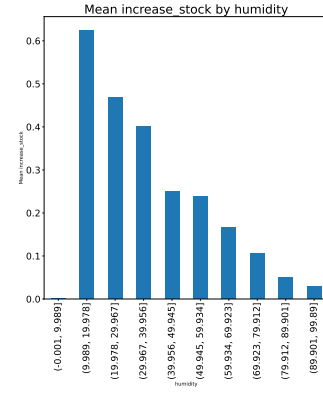
(c) Demand per precipitation (mm/hour).



(d) Demand per dew point ($^{\circ}$ C).



(e) Demand per temperature ($^{\circ}$).



(f) Demand per humidity level (percentage).

Figure 3: Bike demand vs. various weather parameters.

these models being designed with the assumption of variables being normally distributed. This is not the case for this particular data set.

References

- [1] A. Lindholm, N. Wahlström, F. Lindsten, and T. B. Schön. *Machine learning: a first course for engineers and scientists*. Cambridge University Press, 2022.

A Appendix

```
1 import pandas as pd
2 import numpy as np
3 from sklearn.model_selection import train_test_split
4 from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
5 from sklearn.linear_model import LogisticRegression
6 from sklearn.metrics import accuracy_score
7 from sklearn.metrics import classification_report
8
9 df = pd.read_csv('training_data_vt2025.csv')
10
11 # modify the month to represent the periodicity that is observed in
    data.
12 df['month_cos'] = np.cos(df['month']*2*np.pi/12)
13 df['month_sin'] = np.sin(df['month']*2*np.pi/12)
14
15 # time of day, replaced with 3 bool values: is_night, is_day and
    is_evening,
16 # adding the new categories back in the end.
17 def categorize_demand(hour):
18     if 20 <= hour or 7 >= hour:
19         return 'night'
20     elif 8 <= hour <= 14:
21         return 'day'
22     elif 15 <= hour <= 19:
23         return 'evening'
24
25 df['time_of_day'] = df['hour_of_day'].apply(categorize_demand)
26 df_dummies = pd.get_dummies(df['time_of_day'], prefix='is', drop_first
    =False)
27 df = pd.concat([df, df_dummies], axis=1)
28
29 # Create bool of snowdepth and percipitation
30 df['snowdepth_bool'] = df['snowdepth'].replace(0, False).astype(bool)
31 df['precip_bool'] = df['precip'].replace(0, False).astype(bool)
32
33 # Seperate training data from target
34 X=df[['#holiday',
35     'weekday',
36     '#summertime',
37     'temp',
38     '#dew',
39     '#humidity',
40     '#visibility',
41     '#windspeed',
42     '#month',
43     'month_cos',
44     'month_sin',
45     '#hour_of_day',
46     'is_day',
47     'is_evening',
48     'is_night',
49     '#hour_cos',
50     '#hour_sin',
51     'snowdepth_bool',
52     'precip_bool'
53     ]]
54
55 y=df['increase_stock']
56
57 # Split dataset into training and test sets
58 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size
    =0.2, random_state=42)
59
```



```

60 # Apply Linear Discriminant Analysis (LDA)
61 lda = LinearDiscriminantAnalysis(n_components=1)
62 X_train_lda = lda.fit_transform(X_train, y_train)
63 X_test_lda = lda.transform(X_test)
64
65 # Train a classifier (Logistic Regression)
66 clf = LogisticRegression()
67 clf.fit(X_train_lda, y_train)
68
69 # Make predictions
70 y_pred = clf.predict(X_test_lda)
71
72 # Evaluate accuracy
73 accuracy = accuracy_score(y_test, y_pred)
74 print(f"Model Accuracy: {accuracy:.2f}")
75
76 print(classification_report(y_test, y_pred))

```

Listing 1: Code for LDA

```

1 import pandas as pd
2 import numpy as np
3 from sklearn.model_selection import train_test_split
4 from sklearn.discriminant_analysis import
    QuadraticDiscriminantAnalysis
5 from sklearn.metrics import accuracy_score
6 from sklearn.metrics import classification_report
7
8 df = pd.read_csv('training_data_vt2025.csv')
9
10 # modify the month to represent the periodicity that is observed in
    data.
11 df['month_cos'] = np.cos(df['month']*2*np.pi/12)
12 df['month_sin'] = np.sin(df['month']*2*np.pi/12)
13
14 # time of day, replaced with 3 bool values: is_night, is_day and
    is_evening,
15 # adding the new categories back in the end.
16 def categorize_demand(hour):
17     if 20 <= hour or 7 >= hour:
18         return 'night'
19     elif 8 <= hour <= 14:
20         return 'day'
21     elif 15 <= hour <= 19:
22         return 'evening'
23
24 df['time_of_day'] = df['hour_of_day'].apply(categorize_demand)
25 df_dummies = pd.get_dummies(df['time_of_day'], prefix='is', drop_first
    =False)
26 df = pd.concat([df, df_dummies], axis=1)
27
28 # Create bool of snowdepth and percipitation
29 df['snowdepth_bool'] = df['snowdepth'].where(df['snowdepth'] == 0, 1)
30 df['precip_bool'] = df['precip'].where(df['precip'] == 0, 1)
31
32 # Seperate training data from target
33 X=df[['holiday',
34     'weekday',
35     '#summertime',
36     'temp',
37     '#dew',
38     '#humidity',
39     '#visibility',
40     '#windspeed',
41     '#month',

```

```

42         'month_cos',
43         'month_sin',
44         #'hour_of_day',
45         'is_day',
46         'is_evening',
47         'is_night',
48         #'snowdepth_bool',
49         'precip_bool'
50     ]]
51
52 y=df['increase_stock']
53
54 # Split dataset into training and test sets
55 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size
56                                                    =0.2, random_state=42)
57
58 # Apply Quadratic Discriminant Analysis (QDA)
59 qda = QuadraticDiscriminantAnalysis()
60 X_train_lda = qda.fit(X_train, y_train)
61
62 # Make predictions
63 y_pred = qda.predict(X_test)
64
65 # Evaluate accuracy
66 accuracy = accuracy_score(y_test, y_pred)
67 print(f"Model Accuracy: {accuracy:.2f}")
68 print(classification_report(y_test, y_pred))

```

Listing 2: Code for QDA

```

1  import pandas as pd
2  import numpy as np
3  import matplotlib
4  import matplotlib.pyplot as plt
5  from sklearn import tree
6  from sklearn.ensemble import BaggingClassifier, RandomForestClassifier
7  import graphviz
8  from sklearn.model_selection import GridSearchCV, RandomizedSearchCV
9  from sklearn.metrics import classification_report
10
11 df = pd.read_csv('training_data_vt2025.csv')
12 #df.info()
13
14 # Modify the dataset, emphasizing different variables
15 df.iloc[:,12]=df.iloc[:,12]**2
16 df.iloc[:,13]=np.sqrt(df.iloc[:,13])
17 df.iloc[:,11] = df.iloc[:,11]**2
18
19 df['month_cos'] = np.cos(df.month*np.pi/12)
20 df['month_sin'] = np.sin(df.month*np.pi/12)
21
22 # time of day, replaed with low,medium and high demand,
23 # adding the new categories back in the end.
24 def categorize_demand(hour):
25     if 20 <= hour or 7 >= hour:
26         return 'night'
27     elif 8 <= hour <= 14:
28         return 'day'
29     elif 15 <= hour <= 19:
30         return 'evening'
31
32 df['demand_category'] = df['hour_of_day'].apply(categorize_demand)
33 df.dummies = pd.get_dummies(df['demand_category'], prefix='demand',
34                             drop_first=False)

```

```

34 df = pd.concat([df, df_dummies], axis=1)
35
36 # converting to bools
37 def if_zero(data):
38     if data == 0:
39         return True
40     else:
41         return False
42
43 # temperature
44
45 df['snowdepth_bool'] = df['snowdepth'].replace(0, False).astype(bool)
46 df['precip_bool'] = df['precip'].replace(0, False).astype(bool)
47
48 # Split into train and test:
49
50 #df.iloc[:,15]=df.iloc[:,15].replace('low_bike_demand',False)
51 #df.iloc[:,15]=df.iloc[:,15].replace('high_bike_demand',True)
52 np.random.seed(0)
53
54 df_modified=df[ ['#holiday',
55                  'weekday',
56                  #'summertime',
57                  'temp',
58                  #'dew',
59                  #'humidity',
60                  'visibility',
61                  'windspeed',
62                  'month_cos',
63                  'month_sin',
64                  'demand_day',
65                  'demand_evening',
66                  'demand_night',
67                  'snowdepth_bool',
68                  'precip_bool',
69                  'increase_stock']]
70
71 N = df_modified.shape[0]
72 n = round(0.7*N)
73 trainI = np.random.choice(N,size=n,replace=False)
74 trainIndex = df_modified.index.isin(trainI)
75 train = df_modified.iloc[trainIndex]
76 test = df_modified.iloc[~trainIndex]
77
78 X_train = train.drop(columns=['increase_stock'])
79 # Need to transform the qualitative variables to dummy variables
80
81 y_train = train['increase_stock']
82
83 model = RandomForestClassifier(random_state=42)
84 param_grid = {
85     'n_estimators': [100, 200, 300],
86     'max_depth': [10, 20, None],
87     'min_samples_split': [2, 5, 10],
88     'min_samples_leaf': [1, 2, 4]
89 }
90
91 # Set up Grid Search
92 random_search = RandomizedSearchCV(model, param_grid, cv=5, scoring='
    accuracy', n_jobs=-1, verbose=2)
93
94 # Fit on training data
95 random_search.fit(X_train, y_train)
96
97 # Get the best hyperparameters

```

```

98 print("Best Parameters: ", random_search.best_params_)
99 print("Best Accuracy: %.2f" % random_search.best_score_)
100
101 # Update the model with the best parameters
102 best_model = random_search.best_estimator_
103
104 # Fit the best model on the training data
105 best_model.fit(X_train, y_train)
106
107 # Make predictions using the optimized model
108
109
110
111
112 ###
113 #dot_data = tree.export_graphviz(model, out_file=None, feature_names =
114     X_train.columns, class_names = model.classes_,
115     filled=True, rounded=True,
116     leaves_parallel=True, proportion=True)
117 #graph = graphviz.Source(dot_data)
118 #graph.render("decision_tree", format="pdf")
119 X_test = test.drop(columns=['increase_stock'])
120 y_test = test['increase_stock']
121 y_predict = best_model.predict(X_test)
122
123 print(classification_report(y_test, y_predict))

```

Listing 3: Code for Random Forest

```

1 import numpy as np
2 import pandas as pd
3 import matplotlib.pyplot as plt
4 import sklearn.linear_model as skl_lm
5 import sklearn.preprocessing as pp
6 import sklearn.metrics as skl_m
7
8 import sklearn.neighbors as skl_nb
9
10 df = pd.read_csv('training_data_vt2025.csv')
11 #df.info()
12
13 # Modify the dataset, emphasizing different variables
14 #df.iloc[:,12]=df.iloc[:,12]**2
15 #df.iloc[:,13]=np.sqrt(df.iloc[:,13])
16 #df.iloc[:,11] = df.iloc[:,11]**2
17
18 df['month_cos'] = np.cos(df.month*np.pi/12)
19 df['month_sin'] = np.sin(df.month*np.pi/12)
20
21 # time of day, replaed with low,medium and high demand,
22 # adding the new categories back in the end.
23 def categorize_demand(hour):
24     if 20 <= hour or 7 >= hour:
25         return 'night'
26     elif 8 <= hour <= 14:
27         return 'day'
28     elif 15 <= hour <= 19:
29         return 'evening'
30
31 df['demand_category'] = df['hour_of_day'].apply(categorize_demand)
32 df_dummies = pd.get_dummies(df['demand_category'], prefix='demand',
33     drop_first=False)
34 df = pd.concat([df, df_dummies], axis=1)

```

```

34
35 # converting to bools
36 def if_zero(data):
37     if data == 0:
38         return True
39     else:
40         return False
41
42 # temperature
43
44 df['snowdepth_bool'] = df['snowdepth'].replace(0, False).astype(bool)
45 df['precip_bool'] = df['precip'].replace(0, False).astype(bool)
46
47 # Split into train and test:
48
49 df.iloc[:,15]=df.iloc[:,15].replace('low_bike_demand',False)
50 df.iloc[:,15]=df.iloc[:,15].replace('high_bike_demand',True)
51 np.random.seed(0)
52
53 df_modified=df[['holiday',
54                 'weekday',
55                 '#summertime',
56                 'temp',
57                 'dew',
58                 'humidity',
59                 'visibility',
60                 'windspeed',
61                 'month_cos',
62                 'month_sin',
63                 'demand_day',
64                 'demand_evening',
65                 'demand_night',
66                 'snowdepth_bool',
67                 'precip_bool',
68                 'increase_stock']]
69
70 N = df_modified.shape[0]
71 n = round(0.7*N)
72 trainI = np.random.choice(N,size=n,replace=False)
73 trainIndex = df_modified.index.isin(trainI)
74 train = df_modified.iloc[trainIndex]
75 test = df_modified.iloc[~trainIndex]
76
77 # Set up X,Y
78
79 # Train data
80 X = train.iloc[:,0:-2]
81 Y = train['increase_stock']
82
83 # Test data
84 X_test = test.iloc[:,0:-2]
85 Y_test = test['increase_stock']
86
87
88 """
89 # Tests for k-value
90 # TEST 1 - uniform distance
91 missclassification = []
92 for k in range(500): # Try n_neighbours = 1, 2, ...,
93
94     #kNN method
95     scaler = pp.StandardScaler().fit(X)
96     model = skl_nb.KNeighborsClassifier(n_neighbors = k+1, weights = '
uniform')
97     model.fit(scaler.transform(X),Y)

```

```

98
99     # Prediction
100     y_hat = model.predict(scaler.transform(X_test))
101     missclassification.append(np.mean(y_hat != Y_test))
102
103 K = np.linspace(1, 500, 500)
104 plt.plot(K, missclassification, '.')
105 plt.ylabel('Missclassification')
106 plt.xlabel('Number of neighbours')
107 plt.show()
108
109 #TEST 2
110 missclassification = []
111 for k in range(500): # Try n_neighbours = 1, 2, ....,
112
113     #kNN method
114     scaler = pp.StandardScaler().fit(X)
115     model = skl_nb.KNeighborsClassifier(n_neighbors = k+1, weights = '
distance')
116     model.fit(scaler.transform(X),Y)
117
118     # Prediction
119     y_hat = model.predict(scaler.transform(X_test))
120     missclassification.append(np.mean(y_hat != Y_test))
121
122 K = np.linspace(1, 500, 500)
123 plt.plot(K, missclassification, '.')
124 plt.ylabel('Missclassification')
125 plt.xlabel('Number of neighbours')
126 plt.show()
127 """
128
129
130
131 # creating the model
132 model = skl_nb.KNeighborsClassifier(n_neighbors = 120, weights = '
distance')
133
134
135 # Scaling the data, otherwise
136 scaler = pp.StandardScaler().fit(X)
137 model.fit(scaler.transform(X),Y)
138 y_hat = model.predict(scaler.transform(X_test))
139
140
141
142 '''
143 # oskalad data
144 model.fit(X,Y)
145 y_hat = model.predict(X_test)'''
146
147 # Get confusion matrix
148 diff = pd.crosstab(y_hat, Y_test)
149 print(f'Confusion matrix: \n {diff}')
150
151 # No. of TP,TN,FP,FN
152 '''TP = diff.iloc[0,0]
153 TN = diff.iloc[1,1]
154 FP = diff.iloc[1,0]
155 FN = diff.iloc[0,1]'''
156
157 # Get metrics:
158 print(skl_m.classification_report(Y_test, y_hat))

```

Listing 4: Code for K- nearest neighbours

```

1 import numpy as np
2 import pandas as pd
3 import matplotlib.pyplot as plt
4 import sklearn.linear_model as skl_lm
5 import sklearn.preprocessing as pp
6 import sklearn.metrics as skl_m
7
8 df = pd.read_csv('training_data_vt2025.csv')
9 #df.info()
10
11 # Modify the dataset, emphasizing different variables
12 #df.iloc[:,12]=df.iloc[:,12]**2
13 #df.iloc[:,13]=np.sqrt(df.iloc[:,13])
14 #df.iloc[:,11] = df.iloc[:,11]**2
15
16 df['month_cos'] = np.cos(df.month*2*np.pi/12) # period of 12 months
17 df['month_sin'] = np.sin(df.month*2*np.pi/12)
18
19 # time of day, replaed with low,medium and high demand,
20 # adding the new categories back in the end.
21 def categorize_demand(hour):
22     if 20 <= hour or 7 >= hour:
23         return 'night'
24     elif 8 <= hour <= 14:
25         return 'day'
26     elif 15 <= hour <= 19:
27         return 'evening'
28
29 # Adding the categories back, but creating three new categories
30 # for the different times
31 df['demand_category'] = df['hour_of_day'].apply(categorize_demand)
32 df_dummies = pd.get_dummies(df['demand_category'], prefix='demand',
33                             drop_first=False)
34 df = pd.concat([df, df_dummies], axis=1)
35
36 # converting to bools
37 df['snowdepth_bool'] = df['snowdepth'].replace(0, False).astype(bool)
38 df['precip_bool'] = df['precip'].replace(0, False).astype(bool)
39
40 # Split into train and test:
41 np.random.seed(0)
42
43 # Can try different combinations, which inputs give worse perf etc
44 df_modified=df[['holiday',
45                 'weekday',
46                 'summertime',
47                 'temp',
48                 'dew',
49                 'humidity',
50                 'visibility',
51                 'windspeed',
52                 'month_cos',
53                 'month_sin',
54                 'demand_day',
55                 'demand_evening',
56                 'demand_night',
57                 'snowdepth_bool',
58                 'precip_bool',
59                 'increase_stock']]
60
61 N = df_modified.shape[0]
62 n = round(0.7*N)
63 trainI = np.random.choice(N,size=n,replace=False)
64 trainIndex = df_modified.index.isin(trainI)
65 train = df_modified.iloc[trainIndex]

```

```

65 test = df_modified.iloc[~trainIndex]
66
67 # Set up X,Y
68
69 # Train data
70 X = train.iloc[:,0:-2]
71 Y = train['increase_stock']
72
73 # Test data
74 X_test = test.iloc[:,0:-2]
75 Y_test = test['increase_stock']
76
77 model = skl_lm.LogisticRegression()
78
79 # Scaling the data, otherwise
80 scaler = pp.StandardScaler().fit(X)
81 model.fit(scaler.transform(X),Y)
82 y_hat = model.predict(scaler.transform(X_test))
83
84 '''
85 # oskalad data
86 model.fit(X,Y)
87 y_hat = model.predict(X_test)'''
88
89 # Get confusion matrix
90 diff = pd.crosstab(y_hat, Y_test)
91 print(f'Confusion matrix: \n {diff}')
92
93 # No. of TP,TN,FP,FN
94 '''TP = diff.iloc[0,0]
95 TN = diff.iloc[1,1]
96 FP = diff.iloc[1,0]
97 FN = diff.iloc[0,1]'''
98
99 # Get metrics:
100 print(skl_m.classification_report(Y_test, y_hat))

```

Listing 5: Code for Logistic Regression