Do we need more bikes?
Project in Statistical Machine Learning

Anonymous Author(s)
Affiliation
Address
`email`

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 $\{\boldsymbol{x}_i, y_i\}_{n\in\mathbb{N}}$ be the training data set, where $\boldsymbol{x_i}$ 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 b is the training output. The parameter vector is then used in the sigmoid function $\sigma(z) : \mathbb{R} \to [0, 1]$

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

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:

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

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 thee leaf-nodes are used to classify the target variables bases on the input. As of itself the disition tree tends to have unsatisfying results which leads to methodes like random forest and sandbagging that boost its accuracy. Sandbagging is a way to sampel the data in order to get multiple datasets from the same data. One then creates a desition-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 theis aplication makes i nonviable. Random forest on the otherhand is viable, it creates mutiple trees whilse disrecarding random input variable this randomnes 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 ambiance. Then

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

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%.

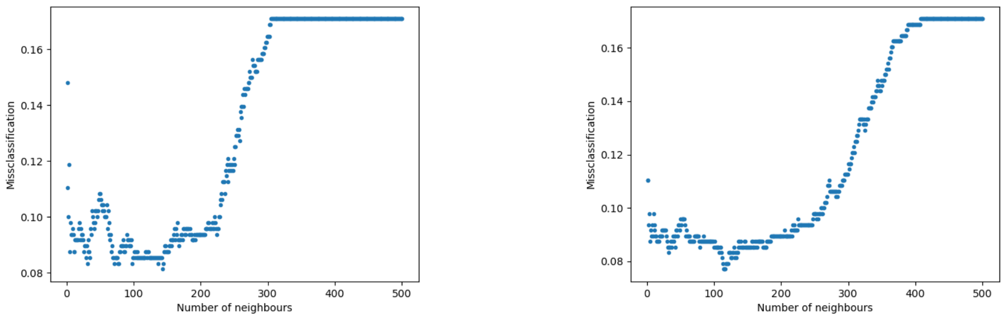
2.1.4 Discriminant analysis: LDA and QDA

Linear Discriminant Analysis is a generative model, which means it is a model that's creating and using a probaility 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 neccesary 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 estimete the probability distribution $p(\mathbf{x}|y = m)$, a multi-dimensional gaussian distribution is used:

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

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

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

Using these estimations results in an expression for the quantity $p(y = m|\mathbf{x})\forall m$. LDA then uses maximum likelyhood 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 $\mathbf{\Sigma}$ 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:

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

One thing to note about LDA and QDA is that the use of a multi-variable gaussian distribution benefints 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:

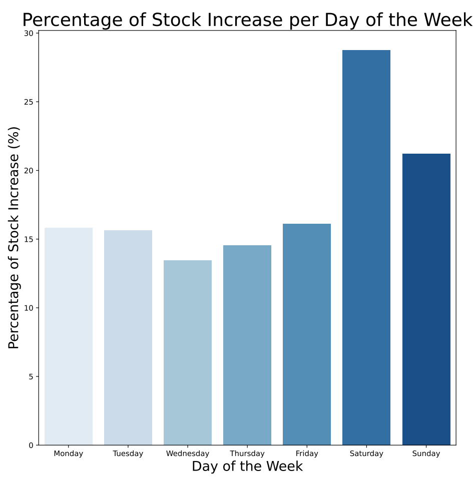
- *Kept as-is*: `weekday`, `windspeed`, `visibility`, `temp`
- *Modified*:

  - `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 plateaues 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`).

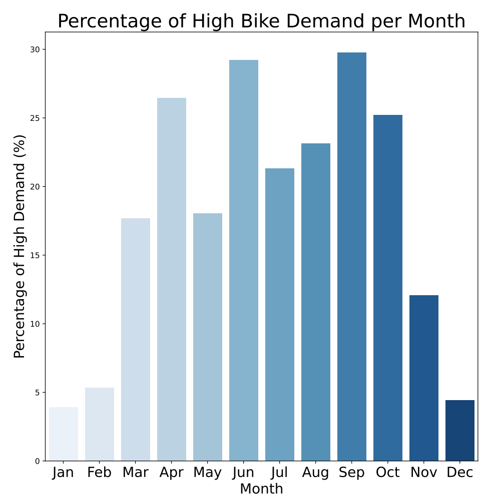
3 Data Analysis

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

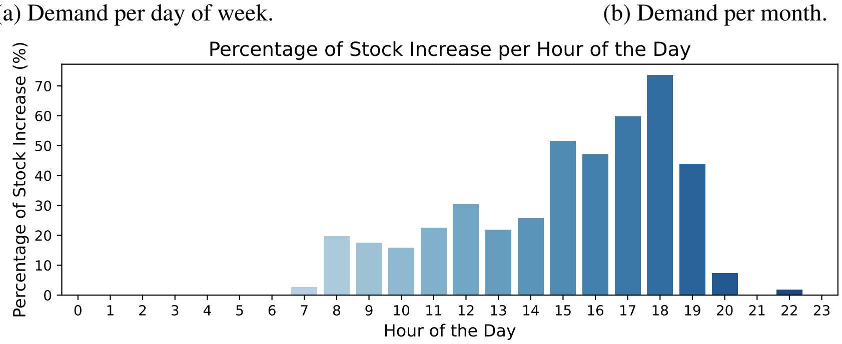
- *Numerical*: `temp`, `dew`, `humidity`, `precip`, `snow`, `snowdepth`, `windspeed`, `cloudcover` and `visibility`.
- *Categorical*: `hour_of_day`, `day_of_week`, `month`, `holiday`, `weekday`, `summertime`, and `increase_stock`



(a) Demand per day of week.



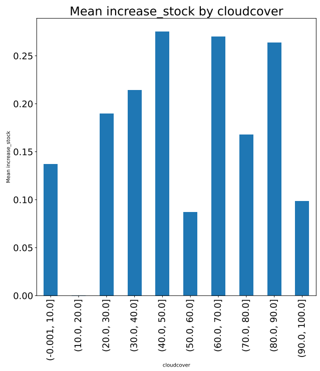
(b) Demand per month.



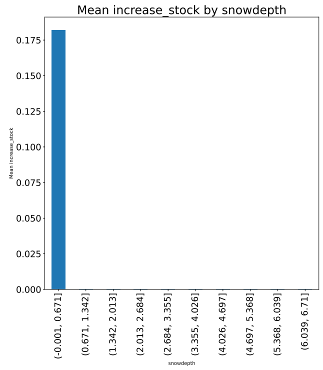
(c) Demand per hour of day.

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

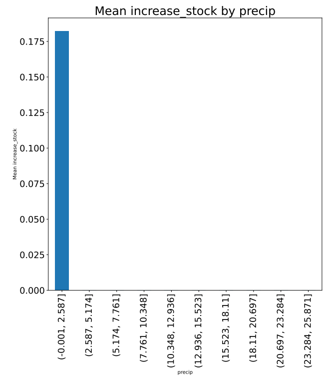
There are some trends seen in the data when it comes to time and weather. From figure 2, one can see a periodic relationship for the months, where there is a higher demand during the warmer months,



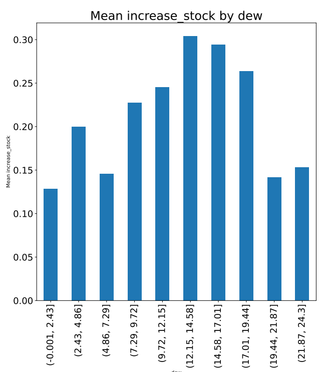
(a) Demand per cloudcover (percentage).



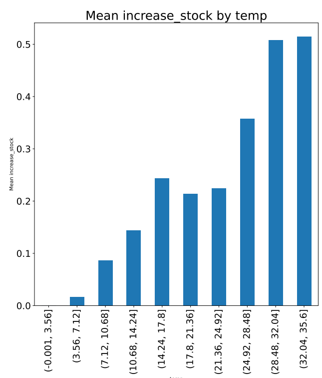
(b) Demand per snowdepth (mm).



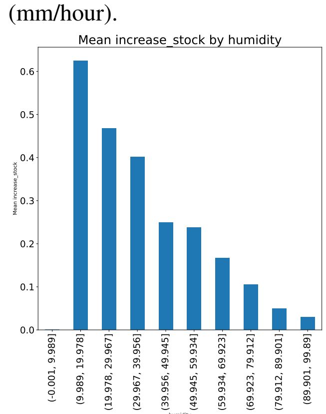
(c) Demand per precipitation (mm/hour).



(d) Demand per dew point (°C).



(e) Demand per temperature (°).



(f) Demand per humidity level (percentage).

Figure 3: Bike demand vs. various weather parameters.

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 eigth 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 CO_2 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:

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

And the precision and recall is defined as:

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

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:

Here you can clearly see random forest and k-nearest neighbour are the best classifiers both

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%

outpreforming 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 out waying 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 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

```python
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report

df = pd.read_csv('training_data_vt2025.csv')

# modify the month to represent the periodicity that is observed in data.
df['month_cos'] = np.cos(df['month']*2*np.pi/12)
df['month_sin'] = np.sin(df['month']*2*np.pi/12)

# time of day, replaced with 3 bool values: is_night, is_day and is_evening,
# adding the new categories back in the end.
def categorize_demand(hour):
    if 20 <= hour or 7 >= hour:
        return 'night'
    elif 8 <= hour <= 14:
        return 'day'
    elif 15 <= hour <= 19:
        return 'evening'

df['time_of_day'] = df['hour_of_day'].apply(categorize_demand)
df_dummies = pd.get_dummies(df['time_of_day'], prefix='is', drop_first=False)
df = pd.concat([df, df_dummies], axis=1)

# Create bool of snowdepth and percipitation
df['snowdepth_bool'] = df['snowdepth'].replace(0, False).astype(bool)
df['precip_bool'] = df['precip'].replace(0, False).astype(bool)

# Seperate training data from target
X=df[[#'holiday',
        'weekday',
        #'summertime',
        'temp',
        #'dew',
        #'humidity',
        #'visibility',
        #'windspeed',
        #'month',
        'month_cos',
        'month_sin',
        #'hour_of_day',
        'is_day',
        'is_evening',
        'is_night',
        #'hour_cos',
        #'hour_sin',
        'snowdepth_bool',
        'precip_bool'
        ]]

y=df['increase_stock']

# Split dataset into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

```

```
# Apply Linear Discriminant Analysis (LDA)
lda = LinearDiscriminantAnalysis(n_components=1)
X_train_lda = lda.fit_transform(X_train, y_train)
X_test_lda = lda.transform(X_test)

# Train a classifier (Logistic Regression)
clf = LogisticRegression()
clf.fit(X_train_lda, y_train)

# Make predictions
y_pred = clf.predict(X_test_lda)

# Evaluate accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f"Model Accuracy: {accuracy:.2f}")

print(classification_report(y_test, y_pred))
```

Listing 1: Code for LDA

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.discriminant_analysis import
    QuadraticDiscriminantAnalysis
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report

df = pd.read_csv('training_data_vt2025.csv')

# modify the month to represent the periodicity that is observed in
    data.
df['month_cos'] = np.cos(df['month']*2*np.pi/12)
df['month_sin'] = np.sin(df['month']*2*np.pi/12)

# time of day, replaced with 3 bool values: is_night, is_day and
    is_evening,
# adding the new categories back in the end.
def categorize_demand(hour):
    if 20 <= hour or 7 >= hour:
        return 'night'
    elif 8 <= hour <= 14:
        return 'day'
    elif 15 <= hour <= 19:
        return 'evening'

df['time_of_day'] = df['hour_of_day'].apply(categorize_demand)
df_dummies = pd.get_dummies(df['time_of_day'], prefix='is', drop_first
    =False)
df = pd.concat([df, df_dummies], axis=1)

# Create bool of snowdepth and percipitation
df['snowdepth_bool'] = df['snowdepth'].where(df['snowdepth'] == 0, 1)
df['precip_bool'] = df['precip'].where(df['precip'] == 0, 1)

# Seperate training data from target
X=df[[#'holiday',
        'weekday',
        #'summertime',
        'temp',
        #'dew',
        #'humidity',
        #'visibility',
        #'windspeed',
        #'month',
```

```
        'month_cos',
        'month_sin',
        #'hour_of_day',
        'is_day',
        'is_evening',
        'is_night',
        #'snowdepth_bool',
        'precip_bool'
        ]]

y=df['increase_stock']

# Split dataset into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size
    =0.2, random_state=42)

# Apply Quadratic Discriminant Analysis (QDA)
qda = QuadraticDiscriminantAnalysis()
X_train_lda = qda.fit(X_train, y_train)

# Make predictions
y_pred = qda.predict(X_test)

# Evaluate accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f"Model Accuracy: {accuracy:.2f}")

print(classification_report(y_test, y_pred))
```

Listing 2: Code for QDA

```
import pandas as pd
import numpy as np
import matplotlib
import matplotlib.pyplot as plt
from sklearn import tree
from sklearn.ensemble import BaggingClassifier, RandomForestClassifier
import graphviz
from sklearn.model_selection import GridSearchCV, RandomizedSearchCV
from sklearn.metrics import classification_report

df = pd.read_csv('training_data_vt2025.csv')
#df.info()

# Modify the dataset, emphasizing different variables
df.iloc[:,12]=df.iloc[:,12]**2
df.iloc[:,13]=np.sqrt(df.iloc[:,13])
df.iloc[:,11] = df.iloc[:,11]**2

df['month_cos'] = np.cos(df.month*np.pi/12)
df['month_sin'] = np.sin(df.month*np.pi/12)

# time of day, replaed with low,medium and high demand,
# adding the new categories back in the end.
def categorize_demand(hour):
    if 20 <= hour or 7 >= hour:
        return 'night'
    elif 8 <= hour <= 14:
        return 'day'
    elif 15 <= hour <= 19:
        return 'evening'

df['demand_category'] = df['hour_of_day'].apply(categorize_demand)
df_dummies = pd.get_dummies(df['demand_category'], prefix='demand',
    drop_first=False)
```

```
df = pd.concat([df, df_dummies], axis=1)

# converting to bools
def if_zero(data):
    if data == 0:
        return True
    else:
        return False

# temperature

df['snowdepth_bool'] = df['snowdepth'].replace(0, False).astype(bool)
df['precip_bool'] = df['precip'].replace(0, False).astype(bool)

# Split into train and test:

#df.iloc[:,15]=df.iloc[:,15].replace('low_bike_demand',False)
#df.iloc[:,15]=df.iloc[:,15].replace('high_bike_demand',True)
np.random.seed(0)

df_modified=df[[#'holiday',
                'weekday',
                #'summertime',
                'temp',
                #'dew',
                #'humidity',
                'visibility',
                'windspeed',
                'month_cos',
                'month_sin',
                'demand_day',
                'demand_evening',
                'demand_night',
                'snowdepth_bool',
                'precip_bool',
                'increase_stock']]

N = df_modified.shape[0]
n = round(0.7*N)
trainI = np.random.choice(N,size=n,replace=False)
trainIndex = df_modified.index.isin(trainI)
train = df_modified.iloc[trainIndex]
test = df_modified.iloc[~trainIndex]

X_train = train.drop(columns=['increase_stock'])
# Need to transform the qualitative variables to dummy variables

y_train = train['increase_stock']

model = RandomForestClassifier(random_state=42)
param_grid = {
    'n_estimators': [100, 200, 300],
    'max_depth': [10, 20, None],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}

# Set up Grid Search
random_search = RandomizedSearchCV(model, param_grid, cv=5, scoring='
    accuracy', n_jobs=-1, verbose=2)

# Fit on training data
random_search.fit(X_train, y_train)

# Get the best hyperparameters
```

```
print("Best Parameters: ", random_search.best_params_)
print("Best Accuracy: %.2f" % random_search.best_score_)

# Update the model with the best parameters
best_model = random_search.best_estimator_

# Fit the best model on the training data
best_model.fit(X_train, y_train)

# Make predictions using the optimized model




###
#dot_data = tree.export_graphviz(model, out_file=None, feature_names =
     X_train.columns,class_names = model.classes_,
#                                    filled=True, rounded=True,
    leaves_parallel=True, proportion=True)
#graph = graphviz.Source(dot_data)
#graph.render("decision_tree", format="pdf")
X_test = test.drop(columns=['increase_stock'])
y_test = test['increase_stock']
y_predict = best_model.predict(X_test)



print(classification_report(y_test, y_predict))
```

Listing 3: Code for Random Forest

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import sklearn.linear_model as skl_lm
import sklearn.preprocessing as pp
import sklearn.metrics as skl_m

import sklearn.neighbors as skl_nb

df = pd.read_csv('training_data_vt2025.csv')
#df.info()

# Modify the dataset, emphasizing different variables
#df.iloc[:,12]=df.iloc[:,12]**2
#df.iloc[:,13]=np.sqrt(df.iloc[:,13])
#df.iloc[:,11] = df.iloc[:,11]**2

df['month_cos'] = np.cos(df.month*np.pi/12)
df['month_sin'] = np.sin(df.month*np.pi/12)

# time of day, replaed with low,medium and high demand,
# adding the new categories back in the end.
def categorize_demand(hour):
    if 20 <= hour or 7 >= hour:
        return 'night'
    elif 8 <= hour <= 14:
        return 'day'
    elif 15 <= hour <= 19:
        return 'evening'

df['demand_category'] = df['hour_of_day'].apply(categorize_demand)
df_dummies = pd.get_dummies(df['demand_category'], prefix='demand',
    drop_first=False)
df = pd.concat([df, df_dummies], axis=1)
```

```
# converting to bools
def if_zero(data):
    if data == 0:
        return True
    else:
        return False

# temperature

df['snowdepth_bool'] = df['snowdepth'].replace(0, False).astype(bool)
df['precip_bool'] = df['precip'].replace(0, False).astype(bool)

# Split into train and test:

#df.iloc[:,15]=df.iloc[:,15].replace('low_bike_demand',False)
#df.iloc[:,15]=df.iloc[:,15].replace('high_bike_demand',True)
np.random.seed(0)

df_modified=df[[#'holiday',
                'weekday',
                #'summertime',
                'temp',
                #'dew',
                'humidity',
                'visibility',
                'windspeed',
                'month_cos',
                'month_sin',
                'demand_day',
                'demand_evening',
                'demand_night',
                'snowdepth_bool',
                'precip_bool',
                'increase_stock']]

N = df_modified.shape[0]
n = round(0.7*N)
trainI = np.random.choice(N,size=n,replace=False)
trainIndex = df_modified.index.isin(trainI)
train = df_modified.iloc[trainIndex]
test = df_modified.iloc[~trainIndex]

# Set up X,Y

# Train data
X = train.iloc[:,0:-2]
Y = train['increase_stock']

# Test data
X_test = test.iloc[:,0:-2]
Y_test = test['increase_stock']


"""
# Tests for k-value
# TEST 1 - uniform distance
missclassification = []
for k in range(500): # Try n_neighbours = 1, 2, ....,

    #kNN method
    scaler = pp.StandardScaler().fit(X)
    model = skl_nb.KNeighborsClassifier(n_neighbors = k+1, weights = 'uniform')
    model.fit(scaler.transform(X),Y)
```

```

    # Prediction
    y_hat = model.predict(scaler.transform(X_test))
    missclassification.append(np.mean(y_hat != Y_test))

K = np.linspace(1, 500, 500)
plt.plot(K, missclassification, '.')
plt.ylabel('Missclassification')
plt.xlabel('Number of neighbours')
plt.show()

#TEST 2
missclassification = []
for k in range(500): # Try n_neighbours = 1, 2, ....,

    #kNN method
    scaler = pp.StandardScaler().fit(X)
    model = skl_nb.KNeighborsClassifier(n_neighbors = k+1, weights = 'distance')
    model.fit(scaler.transform(X),Y)

    # Prediction
    y_hat = model.predict(scaler.transform(X_test))
    missclassification.append(np.mean(y_hat != Y_test))

K = np.linspace(1, 500, 500)
plt.plot(K, missclassification, '.')
plt.ylabel('Missclassification')
plt.xlabel('Number of neighbours')
plt.show()
"""



# creating the model
model = skl_nb.KNeighborsClassifier(n_neighbors = 120, weights = 'distance')


# Scaling the data, otherwise
scaler = pp.StandardScaler().fit(X)
model.fit(scaler.transform(X),Y)
y_hat = model.predict(scaler.transform(X_test))



'''
# oskalad data
model.fit(X,Y)
y_hat = model.predict(X_test)'''

# Get confusion matrix
diff = pd.crosstab(y_hat, Y_test)
print(f'Confusion matrix: \n {diff}')

# No. of TP,TN,FP,FN
'''TP = diff.iloc[0,0]
TN = diff.iloc[1,1]
FP = diff.iloc[1,0]
FN = diff.iloc[0,1]'''

# Get metrics:
print(skl_m.classification_report(Y_test, y_hat))
```

Listing 4: Code for K- nearest neighbours

```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import sklearn.linear_model as skl_lm
import sklearn.preprocessing as pp
import sklearn.metrics as skl_m

df = pd.read_csv('training_data_vt2025.csv')
#df.info()

# Modify the dataset, emphasizing different variables
#df.iloc[:,12]=df.iloc[:,12]**2
#df.iloc[:,13]=np.sqrt(df.iloc[:,13])
#df.iloc[:,11] = df.iloc[:,11]**2

df['month_cos'] = np.cos(df.month*2*np.pi/12) # period of 12 months
df['month_sin'] = np.sin(df.month*2*np.pi/12)

# time of day, replaed with low,medium and high demand,
# adding the new categories back in the end.
def categorize_demand(hour):
    if 20 <= hour or 7 >= hour:
        return 'night'
    elif 8 <= hour <= 14:
        return 'day'
    elif 15 <= hour <= 19:
        return 'evening'

# Adding the categories back, but creating three new categories
# for the different times
df['demand_category'] = df['hour_of_day'].apply(categorize_demand)
df_dummies = pd.get_dummies(df['demand_category'], prefix='demand',
    drop_first=False)
df = pd.concat([df, df_dummies], axis=1)

# converting to bools
df['snowdepth_bool'] = df['snowdepth'].replace(0, False).astype(bool)
df['precip_bool'] = df['precip'].replace(0, False).astype(bool)

# Split into train and test:
np.random.seed(0)

# Can try different combinations, which inputs give worse perf etc
df_modified=df[[#'holiday',
                'weekday',
                #'summertime',
                'temp',
                #'dew',
                'humidity',
                'visibility',
                'windspeed',
                'month_cos',
                'month_sin',
                'demand_day',
                'demand_evening',
                'demand_night',
                'snowdepth_bool',
                'precip_bool',
                'increase_stock']]

N = df_modified.shape[0]
n = round(0.7*N)
trainI = np.random.choice(N,size=n,replace=False)
trainIndex = df_modified.index.isin(trainI)
train = df_modified.iloc[trainIndex]
```

```
test = df_modified.iloc[~trainIndex]

# Set up X,Y

# Train data
X = train.iloc[:,0:-2]
Y = train['increase_stock']

# Test data
X_test = test.iloc[:,0:-2]
Y_test = test['increase_stock']

model = skl_lm.LogisticRegression()

# Scaling the data, otherwise
scaler = pp.StandardScaler().fit(X)
model.fit(scaler.transform(X),Y)
y_hat = model.predict(scaler.transform(X_test))

'''
# oskalad data
model.fit(X,Y)
y_hat = model.predict(X_test)'''

# Get confusion matrix
diff = pd.crosstab(y_hat, Y_test)
print(f'Confusion matrix: \n {diff}')

# No. of TP,TN,FP,FN
'''TP = diff.iloc[0,0]
TN = diff.iloc[1,1]
FP = diff.iloc[1,0]
FN = diff.iloc[0,1]'''

# Get metrics:
print(skl_m.classification_report(Y_test, y_hat))
```
Listing 5: Code for Logistic Regression