Statistical Machine Learning Project

Vencel David Koczka, Aiman Baig, Giorgos Tsouderos, Tomas Grahn

March 16, 2024

Abstract

The goal of this project was to find a model to predict increased bike demand. Feature engineering was used to highlight features that showed high proportions of increased demand and remove features which showed low information about high demand times. From these features four models were trained and optimised with respect to their F-Beta score (β=1.5). These models included logistic regression, quadratic discriminant analysis, K-nearest neighbors and random forest. Of these models quadratic discriminant analysis proved the best with a F-Beta score (β=1.5) of 0.5767 on a hold-out data set.

1 Introduction

Capital Bikeshare is a 24-hour public bicycle-sharing. The problem arises is that there are certain occasions when, due to various circumstances, there are not as many bikes available as there are demands. The District Department of Transportation in the city wants to know if at certain hours an increase in the number of bikes available will be necessary.

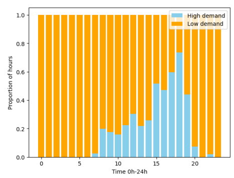
The goal of the project is to predict whether an increase in the number of bikes is necessary or not based on various time and weather data. To accomplish this, first the data was explored. Then feature engineering applied to best expose times of increased demand. Then four models were explored to determine which is best able to predict an increase in demand given a set test of data. Finally the trained models were tested on a set of held out test data to determine which model is best suited for the problem.

2 Data Analysis

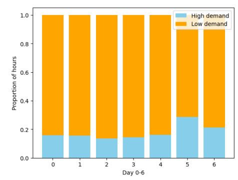
2.1 Numerical and categorical

Reviewing the data in each column revealed that the following categories were categorical: Hour of day, Day of week, Month, Holiday, Weekday, Summertime, Snow, Increase stock. These were numerical: Temperature, Dew, Humidity, Precipitation, Snow depth, Wind speed, Cloud cover, Visibility.

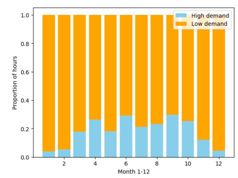
2.2 Trends in time



(a) Proportion of demand at different times of the day



(b) Proportion of demand during the week

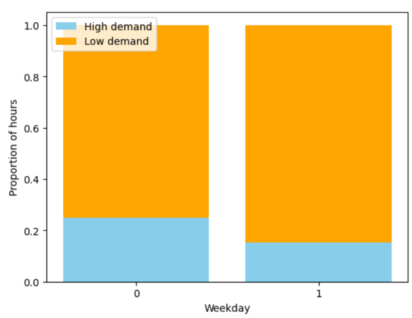


(c) Proportion of demand during different months

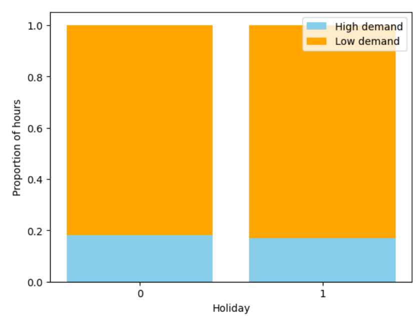
Figure 1: Trends in time,days of the week and months

Figure 1a shows 2 separate areas of high demand. A lower one from 8am to 2pm and a larger one from 3pm to 7pm. Figure 1b shows a slight peak in high demand on Saturday and Sunday, with all other days being fairly equal. Figure 1c see that April, June, September have the highest demand and January, February and December have the least.

2.2.1 Weekdays & Holidays



(a) Proportion of demand on weekdays



(b) Proportion of demand on holidays

Figure 2: Trends in weekdays and holidays

Figure 2a shows that there is slightly more demand on weekends. Figure 2b shows that there is no significant difference between holiday and non-holiday on demand.

2.3 Trends for weather

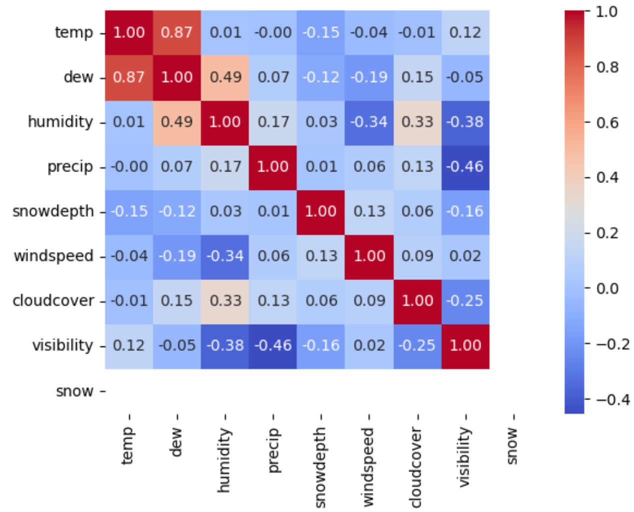
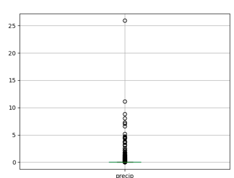
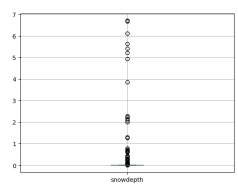


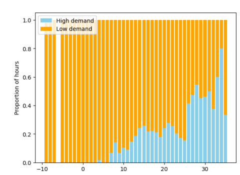
Figure 3: Correlation between features



(a) Box plot of precipitation



(b) Box plot of snow depth



(c) Proportion of demand at different temperatures

Figure 4: Trends in precipitation, snow depth and temperature

Figure 3 shows us that Snow has no correlation to anything, This is because it is only 0 or NaN in the dataset. It also shows us that dew is highly correlated to temperature. Figure 4a shows that a majority of the values are 0. Furthermore looking at the values that are not 0 shows they are all low demand hours. Figure 4b shows that a majority of the values are 0. Furthermore looking at the values that are not 0 shows they are all low demand hours. Figure 4c there is a spike in demand past 25 degrees. There is also increased demand past 12 degrees with central spikes at 16 and 21.

2.4 Basic analysis and naive classifier

By looking at the proportion of high and low demand samples a naive classifier can be constructed. The data is 82% low demand and 18% high demand. Hence a classifier that always classifies as low demand would have an accuracy of 82%. This classifier however has 0 F-Beta score (β=1.5) which is our key metric for finding an appropriate classifier.

2.5 Feature engineering

Feature engineering was used to enhance as much information relating to high demand as possible. Snow was removed as it was only 0 or Nan. Holiday was removed as there was no greater proportion of high demand whether there was a holiday or not. Snow depth and precipitation were removed as they are almost always 0 and when it they are non zero they are low demand. Dew was removed as it was highly correlated with temperature and it so did not add extra information regarding the high demand times. Finally the features were normalised so that k-NN could use.

Due to the high demand between 3pm and 7pm an additional feature was constructed called rush hour to prioritise this information. Finally another feature was added for good temperature to reflect the high demand when the temperature was greater than 25 degrees.

3 Implementation of Methods

Based on the direction given by Capital Bikeshare to look for hours when an increased bike demand would occur, it was decided that the key metric for optimisation of models would be a F-beta score (β=1.5) to give more weight to recall. Recall was seen as more important as it was assumed missing high demand days was the biggest loss for the company. Accuracy, precision and recall are also reported.

3.1 Logistic Regression

Logistic Regression is a binary classification algorithm that models the probability of an instance belonging to a certain class. The Sigmoid function is used in logistic regression. The sigmoid functions is defined as: $\sigma(z) = \frac{1}{1+e^{-z}}$ where σ(z) is the output between 0 and 1, and z is linear combination of the input.

The Logistic Regression model is trained by adjusting the coefficients to minimize the difference between the predicted probabilities and the actual class labels in the training data. The model parameters are adjusted iteratively to minimize the overall loss across all training instances. Grid search with 5 folds was used to search over the possible hyper-parameters. The search iterated over different parameters for regularization amount, regularization method and the maximum iterations. The search was optimised for F-Beta (β=1.5) and the following score was achieved.

Table 1: Logistic Regression Performance

Accuracy	Precision	Recall	F_β
0.8469	0.5128	0.4000	0.4290

The method performed quite well with respect to accuracy but was lacking in the ability to separate the positive samples. This was likely due to the model's linear nature which limits the complexity of the boundaries it can find.

3.2 Discriminant Analysis: QDA

Discriminant analysis is a method of classification in which density functions are constructed based on the training data and are used to classify new points. The density functions are assumed to be normally distributed. The maximum likelihood distribution is found during training by optimising the following

$$f(x;\theta) = \arg\max_{\theta} \sum_{i=1}^{n} \ln(\mathbb{P}(x_i|y_i,\theta)) + \ln(\mathbb{P}(y_i|\theta))$$

If you assume that the functions will share the same covariance matrix then you will be performing linear discriminant analysis. If you allow them to have differing covariance matrices then you arrive at quadratic discriminant analysis. If your data is not linearly separable it is favorable to use quadratic discriminant analysis as it can create non linear boundaries. In the case of our problem quadratic discriminate analysis was seen as favorable as there are some features such as temperature that could yield additional information if a more complex boundary is used.

Sklearn offers the ability to regularise a QDA model with shrinkage. This can assist with data that has a high number of features compared to the amount of data as it helps to compensate for the fact the empirical covariance matrix is a poor estimate for the true covariance matrix. Table 2 shows that shrinkage only made the model perform worse with respect to F Beta (β=1.5) so for the final QDA no shrinkage was used.

Table 2: QDA Performance

Version	Accuracy	Precision	Recall	F_β
QDA	0.8203	0.5443	0.8113	0.7049
QDA with shrinkage	0.8320	0.7083	0.3208	0.3857

The QDA method performed well particularly with respect to recall which also gave it a high F-beta score. QDA seems to perform well due to it's ability to fit non-linear boundaries between the data. This may be due to how the model can better reflect the relationship between high demand and features such as temperature and hour of the day. These features don't appear linear in when there is or isn't demand and the QDA can likely model the relationship better.

3.3 K-nearest neighbor

The k-nearest neighbors algorithm is a supervised machine learning algorithm used for classification and regression problems. A data point is a vector in a multidimensional feature space. When classifying a new data point, its assigned a class label is based on the classes of its k closest neighbors. There are several ways to calculate the distance between each data point (Euclidean, Manhattan, Hamming) but in this case the Minkowski distance was used.

$$MinkowskiDistance = \sum_{i=1}^{n} (|x_i - y_i|)^{\frac{1}{p}}$$

This distance metric is a generalised form of the Euclidean and Manhattan distances, where the change of the p parameter enables the creation of other distance metrics. The k parameter allows the selection of the amount of neighbors checked when deciding to assign a label to a new data point. To find the optimal k parameter, the elbow method was used where the k-NN algorithm is executed for a range of k values and the k that minimizes the loss of the optimizing metric we chose, is the that is selected. In our case, to find the optimal number of neighbors we averaged the F-Beta error rate over 5 random splits which resulted in k=2. When a small value of k is used it usually indicates overfitting, however since KNN is a non parametric algorithm, it can be difficult to avoid these cases. The results after running the KNN algorithm for k=2 are highlighted in the table below.

Table 3: KNN Performance

Accuracy	Precision	Recall	F_β
0.7406	0.3225	0.6000	0.4745

3.4 Tree-based methods

We experimented with three different kind of Tree-based methods, namely Decision Trees, Bagging, and Random Forest. Our best results were achieved with Random Forest, out of these methods. Random forest trains multiple decision trees and averages the results which leads to better performance.

The training process for decision trees splits the feature space into regions. Step by step we always choose the best available feature and value to split on, and this way the tree becomes deeper and deeper. By setting parameters for the trees we can restrict how deep a tree can get, and also manipulate the splitting logic itself, by for example setting the minimum samples a leaf should contain. When predicting new datapoints we use $\hat{y}_l = MajorityVote\{y_i : \mathbf{x}_i \in R_l\}$, Which means that each final region (leaf) has a class determined by the dominant data points contained in that region. The new point will be classified based on that.

For finding the best hyperparameters we used RandomGridSearch method provided by sklearn, as it is faster than simple GridSearch. This method tries out different combination of hyperparameters, and compares the results with cross-validation. For random forest classifiers we found that we should set the number of estimators to 334, the maximum depth of the trees to 50, and minimum samples in each leaf to 1, minimum samples in each split to 5, with Bootstrap set to False, and max_features parameter set to 'sqrt'.

When using decision trees, we can easily overfit on our data if the maximum tree depth is too large. In the case of random forests this is not a bad behaviour, because we average the results out on numerous trees, consequently regularizing the model. For this method our feature engineering does not affect the final performance. This is because the splitting happens on the same features. By training a simple decision tree, we can see that these parameters are hour_of_day, and temp.

Table 4: Random Forest Performance

Accuracy	Precision	Recall	F_β
0.8531	0.5294	0.5400	0.5366

4 Conclusion

Table 5: Results of evaluation

Model	Accuracy	Precision	Recall	F_β
Logistic Regression	0.8469	0.5128	0.4000	0.4290
QDA	0.7750	0.3854	0.7400	0.5767
K-nn	0.7406	0.3225	0.6000	0.4745
Random Forest	0.8531	0.5294	0.5400	0.5366

Figure 5 shows performance of the tuned methods with respect to the held out test set. In terms of our key metric F-beta (β=1.5) the QDA method performed the best and will be chosen as the model for production. Random forest had a better performance in accuracy and a strong performance in precision. Should the users require a model with higher precision and accuracy it is recommended to switch to the random forest model. The F-beta score could be further improved by exploring the use of neural networks, more data exploration and tailored feature engineering for the methods.

A Appendix

```
# -*- coding: utf-8 -*-
"""SML_Project_Final_corrected.ipynb

Automatically generated by Colaboratory.

Original file is located at
    https://colab.research.google.com/drive/1QTEIux5jWMkKk9PVBq7J5F8Bs9N638if

# Data Analysis
i) Which are the numerical and which are the categorical?

ii) Is there a greater trend to need an increase in the availability of bicycles? Study this question
from various perspectives:

    Can any trend be seen comparing different hours, weeks, and months?

    Is there any difference between weekdays and holidays?

    Is there any trend depending on the weather? Rainy days, snowy days, etc.

Write concise answers to each question and support your findings with evidence (statistics, plots,
etc.). Discuss the results. Additionally, you can explore the correlation of features, outliers, range of
values, and many more aspects.
"""

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
data = pd.read_csv('data/training_data.csv')
data

data.info()

categorical = data[['hour_of_day','day_of_week','month','holiday','weekday','summertime','snow','increase_stock']]
# Snow is a very bad variable it's only either 0 or nan
numerical = data[['temp','dew','humidity','precip','snowdepth','windspeed','cloudcover','visibility']]

# Naive classifier and proportion of data
total = len(data)
increase = len(data[data["increase_stock"]=='high_bike_demand'])
decrease = len(data[data["increase_stock"]=='low_bike_demand'])
proportion_increase = increase/total #18%
proportion_decrease = decrease/total #82%
# Hence a naive classifier that always predicts decrease achieve 82% missclassification

# Different hours
```

```python
hours = sorted(data['hour_of_day'].unique())
hours_high_demand = []
hours_low_demand = []
for hour in hours:
  hours_high_demand.append(len(data.query("hour_of_day == @hour and increase_stock == 'high_bike_demand'")))
  hours_low_demand.append(len(data.query("hour_of_day == @hour and increase_stock == 'low_bike_demand'")))

plt.figure()
plt.bar(hours,hours_high_demand)
plt.xlabel('Time 0h-24h')
plt.ylabel('Number of hours with high demand')
plt.figure()
plt.bar(hours,hours_low_demand)
plt.xlabel('Time 0h-24h')
plt.ylabel('Number of hours with low demand')
plt.figure()
plt.bar(hours,hours_high_demand,color='skyblue',label='High demand')
plt.bar(hours,hours_low_demand,bottom=hours_high_demand,color='orange',label='Low demand')
plt.xlabel('Time 0h-24h')
plt.ylabel('Hours')
plt.legend()
plt.figure()
total_hours = np.add(hours_high_demand,hours_low_demand)
plt.bar(hours,hours_high_demand/total_hours,color='skyblue',label='High demand')
plt.bar(hours,hours_low_demand/total_hours,bottom=hours_high_demand/total_hours,color='orange',label='Low demand')
plt.xlabel('Time 0h-24h')
plt.ylabel('Proportion of hours')
plt.legend()

# Different weeks
dws = sorted(data['day_of_week'].unique())
dw_high_demand = []
dw_low_demand = []
for dw in dws:
  dw_high_demand.append(len(data.query("day_of_week == @dw and increase_stock == 'high_bike_demand'")))
  dw_low_demand.append(len(data.query("day_of_week == @dw and increase_stock == 'low_bike_demand'")))

plt.figure()
plt.bar(dws,dw_high_demand)
plt.xlabel('Day 0-6')
plt.ylabel('Hours in high demand')
plt.bar(dws,dw_low_demand)
plt.figure()
plt.bar(dws,dw_high_demand)
plt.xlabel('Day 0-6')
plt.ylabel('Hours in low demand')
plt.bar(dws,dw_low_demand)
plt.figure()
plt.bar(dws,dw_high_demand,color='skyblue',label='High demand')
```

```python
plt.bar(dws,dw_low_demand,bottom=dw_high_demand,color='orange',label='Low demand')
plt.xlabel('Day 0-6')
plt.ylabel('Hours')
plt.legend()
plt.figure()
total_hours = np.add(dw_high_demand,dw_low_demand)
plt.bar(dws,dw_high_demand/total_hours,color='skyblue',label='High demand')
plt.bar(dws,dw_low_demand/total_hours,bottom=dw_high_demand/total_hours,color='orange',label='Low demand')
plt.xlabel('Day 0-6')
plt.ylabel('Proportion of hours')
plt.legend()

# Different months
months = sorted(data['month'].unique())
month_high_demand = []
month_low_demand = []
for month in months:
  month_high_demand.append(len(data.query("month == @month and increase_stock == 'high_bike_demand'")))
  month_low_demand.append(len(data.query("month == @month and increase_stock == 'low_bike_demand'")))

plt.figure()
plt.bar(months,month_high_demand)
plt.xlabel('Month 1-12')
plt.ylabel('Hours with high demand')
plt.figure()
plt.bar(months,month_low_demand)
plt.xlabel('Month 1-12')
plt.ylabel('Hours with low demand')
plt.figure()
plt.bar(months,month_high_demand,color='skyblue',label='High demand')
plt.bar(months,month_low_demand,bottom=month_high_demand,color='orange',label='Low demand')
plt.xlabel('Month 1-12')
plt.ylabel('Hours')
plt.legend()
plt.figure()
total_hours = np.add(month_high_demand,month_low_demand)
plt.bar(months,month_high_demand/total_hours,color='skyblue',label='High demand')
plt.bar(months,month_low_demand/total_hours,bottom=month_high_demand/total_hours,color='orange',label='Low demand')
plt.xlabel('Month 1-12')
plt.ylabel('Proportion of hours')
plt.legend()

"""In the next cell we test for the diferent correlations.

We remove three columns [dew, snow, weekday] because they are highly correlated with other columns.

```

```
We keep all the others for now, we may remove ones later.
"""

import seaborn as sns
# Weekdays and holidays
# Weekdays correlated to days as weekdays are just 0-5, weekends are 0-2
correlation_matrix = data.corr()
# Plotting the heatmap
sns.heatmap(correlation_matrix, annot=True, cmap="coolwarm", fmt=".2f")
plt.show()

# Do some bar plots of numerical values
numerical.boxplot()

# Temprature
data['itemp'] = data['temp'].astype(int)
temps = data['itemp']
temps_high_demand = []
temps_low_demand = []
for temp in temps:
  temps_high_demand.append(len(data.query("itemp == @temp and increase_stock == 'high_bike_demand'")))
  temps_low_demand.append(len(data.query("itemp == @temp and increase_stock == 'low_bike_demand'")))
plt.figure()
total_hours = np.add(temps_high_demand,temps_low_demand)
plt.bar(temps,temps_high_demand/total_hours,color='skyblue',label='High demand')
plt.bar(temps,temps_low_demand/total_hours,bottom=temps_high_demand/total_hours,color='orange',label='Low demand')
plt.xlabel('Temprature')
plt.ylabel('Proportion of hours')
plt.legend()

#Weekdays
weekdays = data['weekday']
weekday_high_demand = []
weekday_low_demand = []
for weekday in weekdays:
  weekday_high_demand.append(len(data.query("weekday == @weekday and increase_stock == 'high_bike_demand'")))
  weekday_low_demand.append(len(data.query("weekday == @weekday and increase_stock == 'low_bike_demand'")))
plt.figure()
total_hours = np.add(weekday_high_demand,weekday_low_demand)
plt.bar(weekdays,weekday_high_demand/total_hours,color='skyblue',label='High demand')
plt.bar(weekdays,weekday_low_demand/total_hours,bottom=weekday_high_demand/total_hours,color='orange',label='Low demand')
plt.xlabel('Weekday')
plt.ylabel('Proportion of hours')
plt.xticks([0,1])
plt.legend()

```

```
#Holiday
holidays = data['holiday']
holiday_high_demand = []
holiday_low_demand = []
for holiday in holidays:
  holiday_high_demand.append(len(data.query("holiday == @holiday
    and increase_stock == 'high_bike_demand'")))
  holiday_low_demand.append(len(data.query("holiday == @holiday and
     increase_stock == 'low_bike_demand'")))
plt.figure()
total_hours = np.add(holiday_high_demand,holiday_low_demand)
plt.bar(holidays,holiday_high_demand/total_hours,color='skyblue',
    label='High demand')
plt.bar(holidays,holiday_low_demand/total_hours,bottom=
    holiday_high_demand/total_hours,color='orange',label='Low
    demand')
plt.xlabel('Holiday')
plt.ylabel('Proportion of hours')
plt.xticks([0,1])
plt.legend()

"""# Importing packages"""

import numpy as np
import pandas as pd

import sklearn.preprocessing as skl_pre
import sklearn.linear_model as skl_lm
import sklearn.discriminant_analysis as skl_da
import sklearn.neighbors as skl_nb

from sklearn import tree
from sklearn.ensemble import BaggingClassifier,
    RandomForestClassifier


import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split

from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import GridSearchCV
from sklearn.datasets import make_classification
from sklearn.metrics import fbeta_score, make_scorer, recall_score,
     precision_score
from sklearn.preprocessing import normalize

"""# Train - valid - test split"""

#just importing the feature manipulation here
data = pd.read_csv('data/training_data.csv')
filtered = data.drop(['precip', 'holiday', 'snowdepth', 'snow', '
    dew'], axis=1)
data_set = filtered
data_set['rushhour'] = filtered['hour_of_day'].apply(lambda x: 1 if
     x > 2 and x < 8 else 0)
```

```
data_set['goodtemp'] = filtered['temp'].apply(lambda x: 1 if x>25 else 0)

X = normalize(data_set.drop('increase_stock',axis=1))
y = data_set['increase_stock']

from sklearn.model_selection import train_test_split
X_in, X_test, y_in, y_test = train_test_split(X, y, test_size=0.20, random_state=42)

def print_metrics(model,X_test,y_test):
  y_preds = model.predict(X_test)
  y_test = np.array(y_test).reshape(-1)
  #Confusion Matrix
  cross_vals=pd.crosstab(y_preds, y_test)
  print(cross_vals)
  missclassification_rate = np.mean([y_preds != y_test])
  print(f"Missclassification {missclassification_rate}")
  accuracy = 1-missclassification_rate
  print(f"Accuracy {accuracy}")
  recall = recall_score(y_test, y_preds,pos_label='high_bike_demand')
  print(f"Recall {recall}")
  precision = precision_score(y_test, y_preds,pos_label='high_bike_demand')
  print(f"Precision {precision}")
  beta = 1.5
  F_beta = ((1+beta**2) *precision * recall) / (beta**2 * precision + recall)
  print(f"F_beta {F_beta}")

"""# Logistic Regression"""

model = LogisticRegression(solver='liblinear')
param_grid = {
    'C': [0.001, 0.01, 0.1, 1, 10, 100, 1000],
    'penalty': [None,'l1','l2'],
    'max_iter': [10, 50 ,100, 500, 1000,5000],
    'solver': ['liblinear']
}
fb_score = make_scorer(fbeta_score, beta=1.5, pos_label='high_bike_demand')
grid_search = GridSearchCV(model, param_grid, cv=5, scoring=fb_score)
grid_search.fit(X_in, y_in)
best_model = grid_search.best_estimator_
#print_metrics(best_model, X_test, y_test)

grid_search.best_params_

model = LogisticRegression(solver='liblinear',C=1000,max_iter=50, penalty='l1')
model.fit(X_in, y_in)
print_metrics(model, X_test, y_test)

"""# Discriminate Analysis"""

```

```
X_train, X_valid, y_train, y_valid = train_test_split(X_in, y_in, test_size=0.20, random_state=1337)

model1 = skl_da.QuadraticDiscriminantAnalysis(reg_param=0.005)
model2 = skl_da.QuadraticDiscriminantAnalysis(reg_param=0)
model1.fit(X_train, y_train)
model2.fit(X_train, y_train)
print_metrics(model1, X_valid, y_valid)
print_metrics(model2, X_valid, y_valid)

model = skl_da.QuadraticDiscriminantAnalysis(reg_param=0)
model.fit(X_in, y_in)
print_metrics(model, X_test, y_test)

"""# K nearest neighbours"""

#elbow method for figuring out optimal number of k neighbours
from sklearn.metrics import precision_score
from sklearn.model_selection import KFold

def metric_f_beta(y_test,y_preds):
  y_test = np.array(y_test).reshape(-1)
  recall = recall_score(y_test, y_preds,pos_label='high_bike_demand')
  precision = precision_score(y_test, y_preds,pos_label='high_bike_demand')
  beta = 1.5
  F_beta = ((1+beta*2) *precision * recall) / (beta*2 * precision + recall)
  return F_beta
n = [x for x in range(1,50)]
error_rates = []

kf = KFold(n_splits=5, random_state=42, shuffle=True)
y_in = np.array(y_in)
for j, (train_index, test_index) in enumerate(kf.split(X_in)):
  error_rate_i = []
  for i in n :
    model = skl_nb.KNeighborsClassifier(n_neighbors=i)
    model.fit(X_in[train_index], y_in[train_index])
    y_pred = model.predict(X_in[test_index])
    error_rate = 1-metric_f_beta(y_in[test_index], y_pred)
    error_rate_i.append(error_rate)
  error_rates.append(error_rate_i)

avg_errors = np.mean(np.array(error_rates), axis=0)

plt.plot(n, avg_errors)
plt.xlabel('Number of neighbors (k)')
plt.ylabel('Error f_beta rate')
plt.show()
print(f'minimum is: {np.min(avg_errors)} at k={np.argmin(avg_errors)+1}')

n = 2
model = skl_nb.KNeighborsClassifier(n_neighbors=n)
model.fit(X_in, y_in)
```

```
print_metrics(model, X_test, y_test)

"""# Tree based methods"""

from sklearn.model_selection import RandomizedSearchCV
from sklearn.ensemble import RandomForestClassifier
model = RandomForestClassifier()
# Number of trees in random forest
n_estimators = [int(x) for x in np.linspace(start = 1, stop = 1000, num = 10)]
# Number of features to consider at every split
max_features = ['auto', 'sqrt']
# Maximum number of levels in tree
max_depth = [int(x) for x in np.linspace(1, 100, num = 5)]
max_depth.append(None)
# Minimum number of samples required to split a node
min_samples_split = [2, 5, 8 ,10]
# Minimum number of samples required at each leaf node
min_samples_leaf = [1, 2, 4, 8, 15]
# Method of selecting samples for training each tree
bootstrap = [True, False]
# Create the random grid
random_grid = {'n_estimators': n_estimators,
               'max_features': max_features,
               'max_depth': max_depth,
               'min_samples_split': min_samples_split,
               'min_samples_leaf': min_samples_leaf,
               'bootstrap': bootstrap}
fb_score = make_scorer(fbeta_score, beta=1.5,pos_label='high_bike_demand')
rf_random = RandomizedSearchCV(estimator=model, param_distributions=random_grid, n_iter=100, cv=3, verbose=2, random_state=42, n_jobs=-1, scoring=fb_score)
rf_random.fit(X_in, y_in)

rf_random.best_params_

model = RandomForestClassifier(n_estimators=334, max_depth=50, min_samples_leaf=1, min_samples_split=5, bootstrap=False, max_features='sqrt')
model.fit(X_in, y_in)
print_metrics(model, X_test, y_test)

"""# Final test"""

final_test = pd.read_csv('test_data.csv')
final_test = final_test.drop(['precip', 'holiday', 'snowdepth', 'snow', 'dew'], axis=1)
final_test['rushhour'] = final_test['hour_of_day'].apply(lambda x: 1 if x > 2 and x < 8 else 0)
final_test['goodtemp'] = final_test['temp'].apply(lambda x: 1 if x >25 else 0)
final_test = normalize(final_test)

# Run final test using random forest as it had highest accuracy
# and we assume this will be the metric used to measure performance
```

```python
model = RandomForestClassifier(n_estimators=334, max_depth=50,
    min_samples_leaf=1, min_samples_split=5, bootstrap=False,
    max_features='sqrt')
model.fit(X, y)

preds = model.predict(final_test)
np.unique(preds, return_counts=True)

import csv
out_preds = [0 if pred == 'low_bike_demand' else 1 for pred in
    preds]
with open(r'out.csv', 'w') as fp:
    for pred in out_preds:
        # write each item on a new line
        fp.write("%s," % pred)
```