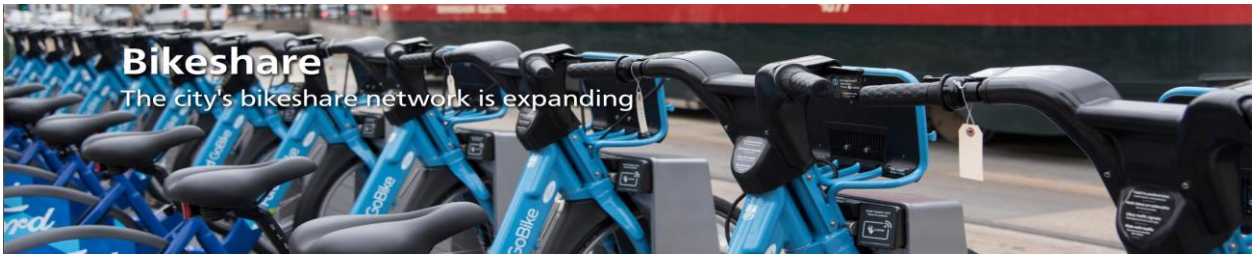
# HW05 – Classification for Bike Sharing




**FILE**

bikeday.csv - For this assignment, you will be using different classification techniques with the San Francisco bike sharing dataset.

**Format of this Homework**

For full points on this assignment, it is important that your Jupyter Notebook is formatted correctly with markdown, comments, and code that works.

**You are to do the following for each section:**

- Include a title as markdown Heading 2, for example: "**1.4 Decision Tree Classifier**"
- Include a description of the section detailing its purpose (*markdown*)
- Include your code and make sure it is executable and correct. (*code*)
- At the end of the section, include a summary of the results in markdown **based on the "Business and Technology Objectives".**

**How to turn it in:**

- Your Jupyter notebook file must be named HW05_LastnameFirstInitial.ipynb. For example, HW05_ApigianC.ipynb.
- You are to turn in your Jupyter notebook file only. No data files and no folders.
- It is assumed that you created your Jupyter notebook in a folder named HW05_student and in that folder is a data folder. It is expected the path for importing data is in "data" folder, for example '**data/bikeday.csv'**.

# Can we predict running out of bikes?

**Business and Technology Objectives**

Look at bike rental data for numerical predictions and relationships

- **Business Objective:** Identify when a station will be out of bikes
- **Technology Objective:** Logistic regression – based on probabilities (1.2)
- **Technology Objective:** Decision Tree Classifier – identifies specific features (1.3)
- **Technology Objective:** Random Forest (1.4)

**Main Deliverable:** Create one DataFrame that includes all X_test, y_test data and all predictions and probabilities for logistic regression, decision tree, and random forest. Name the DataFrame df_bike_results.

**Final DataFrame: df_bike_results**

| | trip_time_sum | trip_count | subscribe_percent | dock_count | mean_temperature_f | bikes_out | Predict_Log | Prob_NoAffair | Prob_YesAffair | Predict_Tree | Predict_RF |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 96 | 9 | 0.555556 | 19 | 57 | 1 | 0 | 0.933964 | 0.066036 | 0 | 0 |
| 1 | 3054 | 10 | 0.500000 | 15 | 60 | 0 | 0 | 0.838738 | 0.161262 | 0 | 0 |
| 2 | 11 | 1 | 1.000000 | 19 | 61 | 0 | 0 | 0.995547 | 0.004453 | 0 | 0 |
| 3 | 291 | 28 | 1.000000 | 15 | 58 | 1 | 1 | 0.307917 | 0.692083 | 1 | 1 |
| 4 | 221 | 19 | 0.789474 | 15 | 60 | 0 | 0 | 0.722010 | 0.277990 | 0 | 1 |

*Note: does not include Pred_Enhance*

It is hard to get a bike if none are available. Therefore, we would like to predict if the bike station will run out of bikes. In the Data Preparation part of HW04, you changed bikes_avail_min to a 0 or 1***. "1" being they did run out of bikes and "0" that they never ran out of bikes.*** Looking at bikes_avail_min as the target variable (y) we want to try and predict if the station will run out of bikes on any given day. We will do this with Logistic Regression, Decision Trees, and Random Forest. **For Part 2**, make sure it is obvious in your notebook where this part of the assignments starts. For example:

| ID | date | start_station_id | trip_time_sum | trip_count | subscribe_percent | station_id | bikes_avail_median | docks_avail_median | bikes_avail_min | dock_count | doc |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 8/29/13 | 2 | 73 | 5 | 0.600000 | 2 | 2.0 | 25.0 | 1 | 27 | |
| 1 | 8/29/13 | 3 | 94 | 9 | 0.555556 | 3 | 9.0 | 6.0 | 3 | 15 | |
| 2 | 8/29/13 | 4 | 18 | 3 | 1.000000 | 4 | 5.0 | 6.0 | 0 | 11 | |
| 3 | 8/29/13 | 5 | 20 | 3 | 1.000000 | 5 | 9.0 | 10.0 | 8 | 19 | |
| 4 | 8/29/13 | 6 | 33 | 4 | 1.000000 | 6 | 4.0 | 11.0 | 1 | 15 | |

**XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX ¶**

**XXXXXXXXXXXXXXXXXXXX Classification XXXXXXXXXXXXXXXXXXX**

**XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX**

```
y = df_bike_class['bikes_out']
```

```
X = df_bike_class
```

### 1.1 Data Preparation

- Make sure to setup the folder structure correctly and create a new ipynb file named HW05_LastNameFirstInitial, a data folder with data.
- To see the decision tree, you will need to use the terminal in Anaconda to install the following: (*You need to install graphviz*)
    - pip install pydotplus
    - pip install pydot
    - conda install -c anaconda graphviz
- Import bikeday.csv, header and index column = 0. Name the dataframe **df_bike_class**.
- Drop the following variables from **df_bike_class**. (This includes all variables for X and y.)
    - date
    - station_id
    - start_station_id
    - bikes_avail_median
    - docks_avail_median
    - bikes_avail_min
- Create a dataframe (**X**) from **df_bike_class**
    - Should include all variables except **bikes_out**
- Create a dataframe (**y**)
    - Should include the **bikes_out** variable only.
    - **DO NOT** Reshape the **y** variable. **USE:**
        - y = df_bike_enhanced['bikes_out'] **OR**
        - y = df_bike_enhanced[ ['bikes_out'] ]
- Split the data for test/train.
    - The train and test X datasets should be based on X. (X_train and X_test)
    - The train and test y datasets should be based on y. (y_train and y_test)
    - Set the test size to 0.3
    - Set the random state to 42
- Create scaled sets of the X dataframe.
    - X_scaled = scaler.fit_transform(X)
- Split the data for test/train for the scaled dataframe.
    - The train and test X datasets should be based on X_scaled. (X_trainSC and X_testSC)
    - The train and test y datasets should be based on y. (y_trainSC and y_testSC)
    - Set the test size to 0.3
    - Set the random state to 42.

Note: This gives you two datasets to use to test. One with raw data and one with a scaled X dataset.

**1.2 Logistic Regression**

- Import:
  - from sklearn.linear_model import LogisticRegression
  - from sklearn.metrics import confusion_matrix, classification_report
- Create a Logistic Regression classifier using:
  - logr = LogisticRegression()
  - logr.fit(X_train, y_train)
  - score = logr.score(X_test, y_test)
  - print(score)
- Create a predicted variable from X_test
  - log_pred = logr.predict(X_test)
  - set log_predict as a dataframe
  - rename the column ***Predict_Log***
- Create predicted probabilities from X_test
  - log_prob = logr. predict_proba(X_test)
  - set log_prob as a dataframe
  - rename the columns ***Prob_Avail*** and ***Prob_Out***
- Print confusion matrix and classification report
  - print(confusion_matrix(y_test, log_pred))
  - print("")
  - print(classification_report(y_test, log_pred))

**Create df_bike_results**

- Create a new dataframe named df_bike_results by:
  - Concat X_test and y_test
  - Keep only the following columns:
    - trip_time_sum, trip_count, subscribe_percent, dock_count, mean_temperature_f, bikes_out
  - Reset the index
  - Drop ID -> *df_bike_results.drop('ID', axis = 1)*
  - Concat df_bike_results with log_predict and log_prob.

**1.3 Logistic Regression with Standard Scaler**

- Rerun a logistic regression using the standard scaler datasets
  - X_trainSC
  - X_testSC
  - y_trainSC
  - y_testSC
- Make sure to print the score, confusion matrix and classification report
- You DO NOT need to add this to df_bike_results
- ***Summarize the Logistic Regression results in your own words***

### 1.4 Decision Tree Classifier

- Import:
  - from sklearn.feature_extraction.text import CountVectorizer
  - from sklearn.tree import DecisionTreeClassifier
- Create a Decision Tree Classifier using:
  - classifier=DecisionTreeClassifier()
  - classifier=classifier.fit(X_train,y_train)
- Print the confusion matrix and the classification report
  - dt_preds = classifier.predict(X_test)
  - set dt_preds as a dataframe
  - rename the column ***Predict_Tree***
- Print confusion matrix and classification report
  - print(confusion_matrix(y_test, dt_preds))
  - print("")
  - print(classification_report(y_test, dt_preds))
- Run a function to identify the best number of depths for your classifier

```
depth = range(1,12)
scores = []

for d in depth:
    classifier=DecisionTreeClassifier(max_depth = d)
    classifier=classifier.fit(X_train,y_train)
    scores.append(classifier.score(X_test, y_test))
    print("iteration {} done".format(d))


plt.plot(depth, scores, '-o')
plt.xlabel('depth, d')
plt.ylabel('scores')
plt.xticks(depth)
plt.show()
```

- Create a **Second** Decision Tree Classifier using:
  - classifier2=DecisionTreeClassifier(max_depth = ___)
    - Select the depth based on the above function.
  - classifier2=classifier.fit(X_train,y_train)
  - classifier2.score(X_test, y_test)
- Print the confusion matrix and the classification report
  - dt_preds2 = classifier2.predict(X_test)
  - print(confusion_matrix(y_test, dt_preds2))
  - print(classification_report(y_test, dt_preds2))
- **Feature Importance**
  - Find out which features are most important by using the following code:
    - dt_fi = pd.DataFrame(classifier.feature_importances_)
    - names = pd.DataFrame(list(X.columns))
    - df_feat_imp = pd.concat([dt_fi, names], axis = 1)
    - df_feat_imp.columns = ['Importance', 'Features']
    - df_feat_imp.sort_values('Importance', ascending = False)
  - Review the importance features and summarize the results with markdown.
- **Summarize the Decision Tree results in your own words**

### 1.5 Random Forest

- Import:
  - from sklearn.ensemble import RandomForestClassifier
- Run a random forest suing the following code:
  - rf = RandomForestClassifier(n_estimators = 90, max_depth = 10)
  - rf = rf.fit(X_train, y_train)
- Print Scores
  - score = rf.score(X_test, y_test)
  - print(score)
  - rf_pred = rf.predict(X_test)
  - set rf_pred as a dataframe
  - rename the column ***Predict_RF***
- Print confusion matrix and classification report
  - print(classification_report(y_test, rf_pred))
  - print("")
  - print(confusion_matrix(y_test, rf_pred))

**Final DataFrame: df_bike_results**

| | trip_time_sum | trip_count | subscribe_percent | dock_count | mean_temperature_f | bikes_out | Predict_Log | Prob_Avail | Prob_Out | Predict_Tree | Predict_RF |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 96 | 9 | 0.555556 | 19 | 57 | 1 | 0 | 0.933964 | 0.066036 | 0 | 0 |
| 1 | 3054 | 10 | 0.500000 | 15 | 60 | 0 | 0 | 0.838738 | 0.161262 | 0 | 0 |
| 2 | 11 | 1 | 1.000000 | 19 | 61 | 0 | 0 | 0.995547 | 0.004453 | 0 | 0 |
| 3 | 291 | 28 | 1.000000 | 15 | 58 | 1 | 1 | 0.307917 | 0.692083 | 1 | 1 |
| 4 | 221 | 19 | 0.789474 | 15 | 60 | 0 | 0 | 0.722010 | 0.277990 | 0 | 1 |

- **Feature Importance**
  - Find out which features are most important by using the following code:
    - fi = pd.DataFrame(rf.feature_importances_)
    - columns = pd.DataFrame(list(X.columns))
    - features = pd.concat([columns, fi], axis = 1)
    - features.columns = ['Feature', 'Importance']
    - features.sort_values("Importance", ascending = False)
- Review the importance features and summarize the results with markdown.
- ***Summarize the Random Forest results in your own words***

### 1.6 Select a test and try to enhance the results

- Select one of the three classification tests and change the parameters and see if it gives you a better or worse result.
- Run the test with your parameters
- Print Scores

  - score = _____.score(X_test, y_test)
  - print(score)
  - enhance_pred = ____.predict(X_test)
  - set enhance_pred as a dataframe
  - rename the column ***Predict_Enhance***
- Print confusion matrix and classification report
  - print(classification_report(y_test, enhance_pred))
  - print("")
  - print(confusion_matrix(y_test, enhance_pred))
- ***Summarize the results in your own words, was it better or worse, what were your changes?***

**1.7 Create a complete dataset with all predictions**

- Create a DataFrame named ***df_bike_results*** that includes the actual data and the predictions, which should include:
  - All features from X and include y
  - Predict_Log, Prob_Avail, and Prob_Out
  - Predict_Tree
  - Predict_RF
  - Predict_Enhance

| | trip_time_sum | trip_count | subscribe_percent | dock_count | mean_temperature_f | bikes_out | Predict_Log | Prob_Avail | Prob_Out | Predict_Tree | Predict_RF |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 96 | 9 | 0.555556 | 19 | 57 | 1 | 0 | 0.933964 | 0.066036 | 0 | 0 |
| 1 | 3054 | 10 | 0.500000 | 15 | 60 | 0 | 0 | 0.838738 | 0.161262 | 0 | 0 |
| 2 | 11 | 1 | 1.000000 | 19 | 61 | 0 | 0 | 0.995547 | 0.004453 | 0 | 0 |
| 3 | 291 | 28 | 1.000000 | 15 | 58 | 1 | 1 | 0.307917 | 0.692083 | 1 | 1 |
| 4 | 221 | 19 | 0.789474 | 15 | 60 | 0 | 0 | 0.722010 | 0.277990 | 0 | 1 |

*Figure: Snapshot of df_bike_results (does not include Predict_Enhance)*

**1.8 Final Analysis -** Please give a summary of all classification models and give some insight into the results. How did it do, which test is best? What features are the most significant?

**Submission**

Save your file as HW05_LastNameFirstInitial.ipynb. Make sure each section has a noticeable header with numbering in markdown. Make sure each section has a summary of the findings in markdown that explains the results based on the business objective. Turn it in to D2L per the Dropbox instructions. (file only, no data)