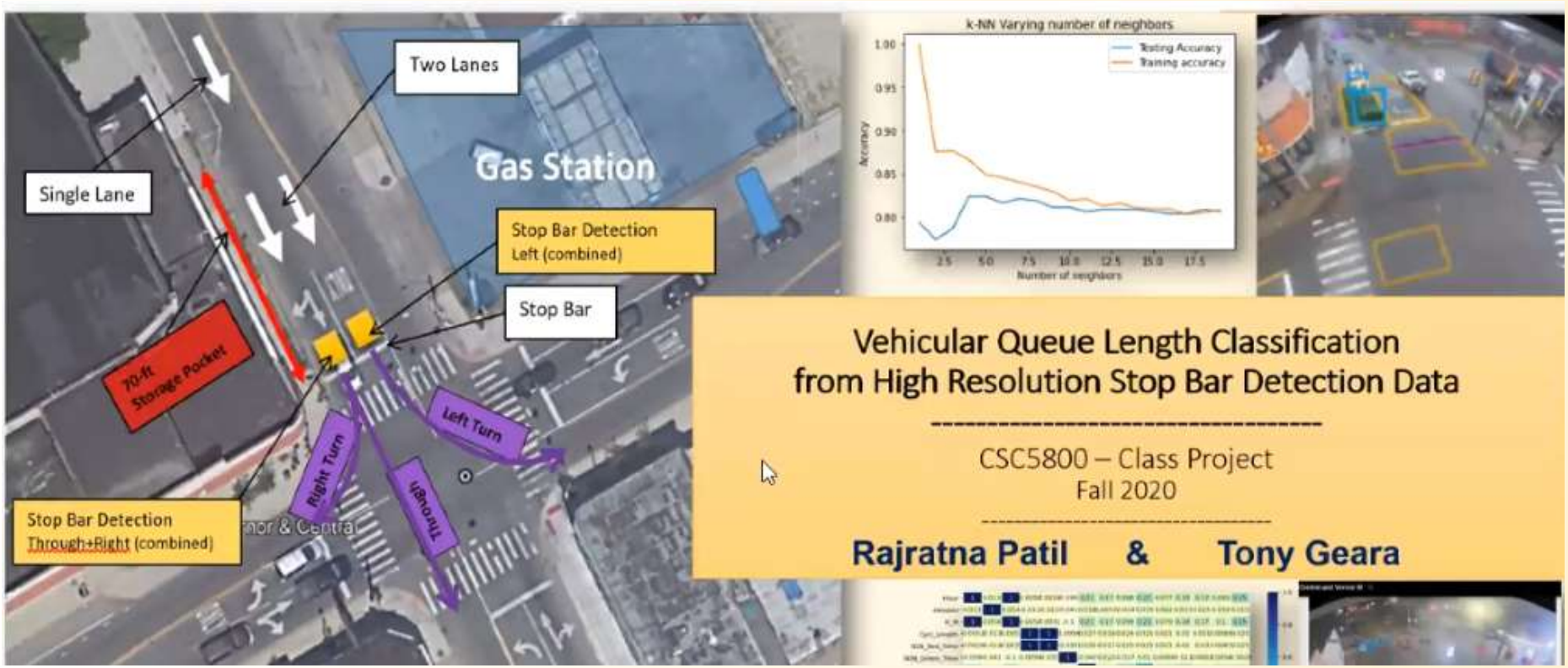
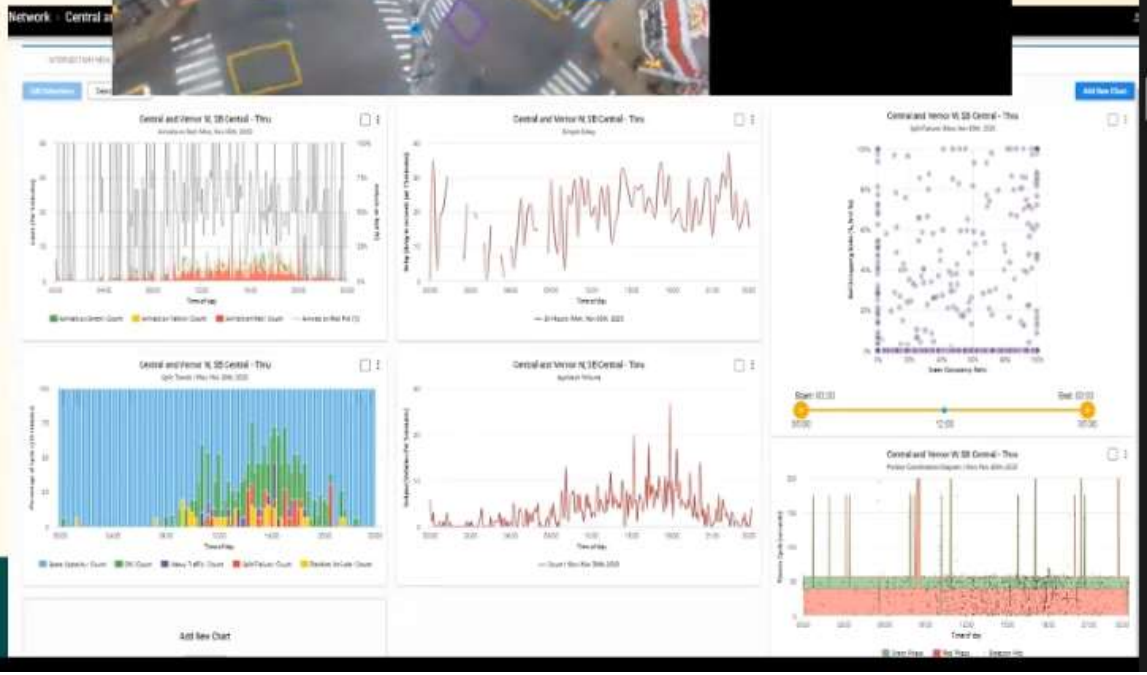

Two Lanes
Gas Station
Single Lane
Stop Bar Detection
Left (combined)
Stop Bar
70-ft
Storage Pocket
Left Turn
Right Turn
Through
Stop Bar Detection
Through+Right (combined)
k-NN Varying number of neighbors
Testing Accuracy
Training accuracy
Accuracy
Number of neighbors
Vehicular Queue Length Classification
from High Resolution Stop Bar Detection Data
CSC5800 – Class Project
Fall 2020
Rajratna Patil & Tony Geara

Introduction

- Traffic Signal Performance (TSPM)

- Automated Traffic Signal performance Measures (ATSPM)



Basic ATSPMs: Approach Volume and turning movement counts,

Advanced ATSPMs: Purdue Phase Determination, Split Monitoring, Pedestrian Delay, Preemption, Split Failures, Approach Delay, Queue Detection, and Arrivals on Red.

Less common ATSPMs: Red Light Running (RLR), Yellow-and-Red actuation, Headway, and Speed.

ATSPM Infrastructure

Traditional required installations

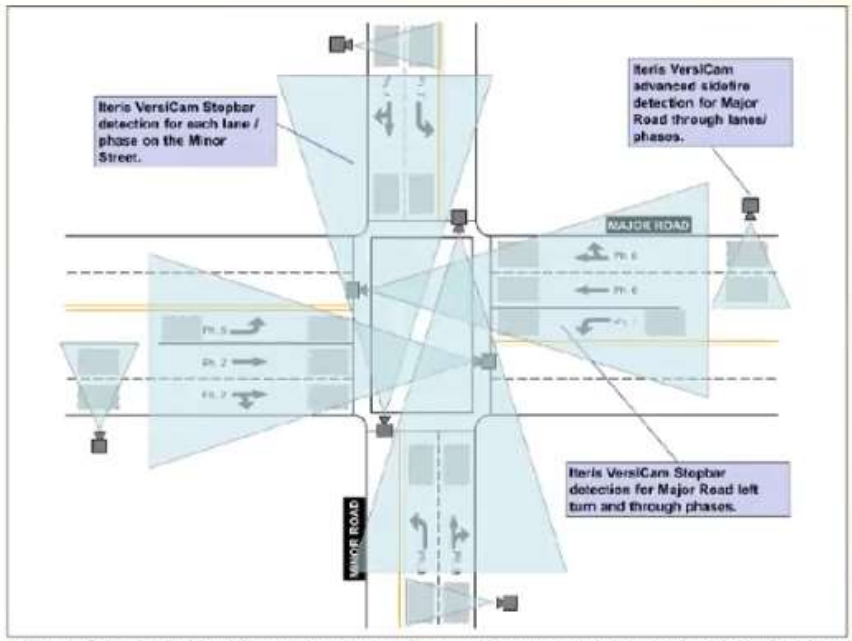
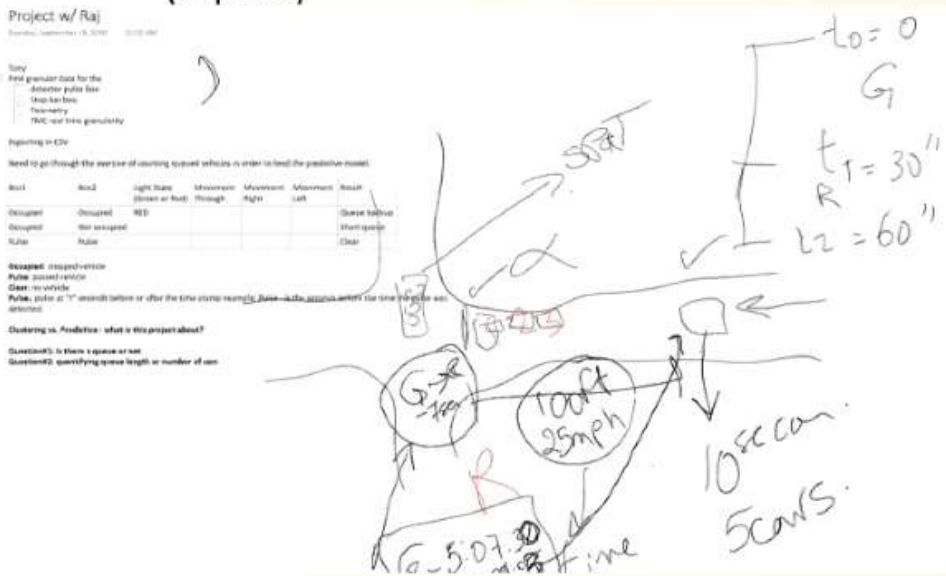


Figure 8.11: Iteris VersiCam video detector layouts for stop bar detection for Major Road and

Proposed Single-Source Installations

Idea Development

Initial Idea and Hand Sketch (Sept 15)



Proposed Approach (Dec)

Problem and Objective

Project Statement/Goals

i. How can we leverage existing HiRes stop bar video detection in the City of Detroit to predict performance (queuing) at Urban intersections?

ii. If so, how accurately can that be done relative to existing state of the practice metrics?

iii. Are there other existing data that can be leveraged to improve the accuracy of the performance metrics?

Objective Statement

i. Collect and Visualize existing High Resolution Signal Controller data at an intersections in the city of Detroit.

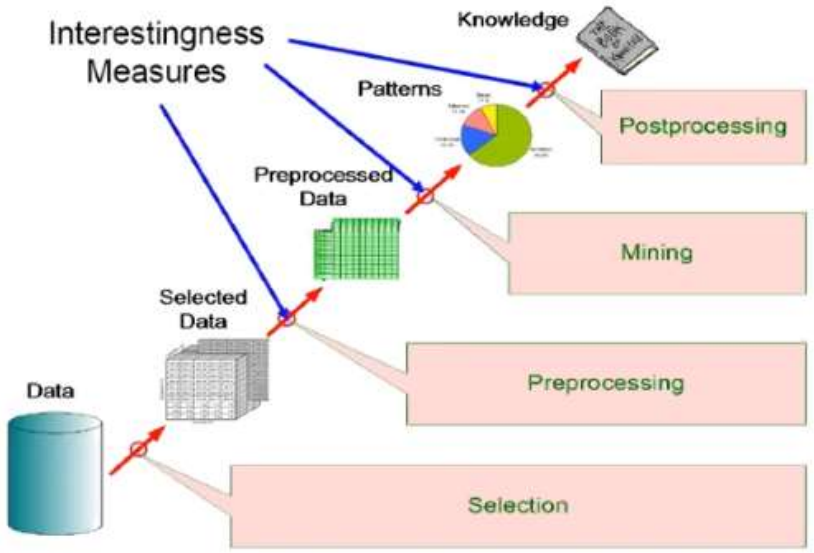
ii. Attempt to create a Methodology to define the performance measure logic using stop bar detection

iii. Compare/classify queue performance measure using advanced detection (real world) with performance measures using stop bar detection

iv. Allow method to be able to Scale and measure the performance difference between advanced and stop bar logic.

Methodology

Application of Interestingness Measure



- Data Extracted from Miovision API –
  - November 30th was chosen
  - Day within the last month with highest southbound movement and overall signal volume for the day and per hour to account for worst case scenario.
  - High Resolution (HiRes) Data Extracted:
    - Stop-Bar Detection data
    - Advance Pulse-Box Detection data
    - Turning Movement Counts data (TMC)
  - Aggregated 1-minute Data Extracted"
    - Turning Movement Counts
- Account for Timestamp time in extraction (based on UTC time for the HiRes data and local time for the TMC) – HiRes had to be converted to local time.
- Extracted HiRes files were a series of 1-hr groups in JSON format that needed conversion and joining
- JSON converted to .txt files then worked with as .csv
- Timestamp converted to regular time with millisecond precision (functional time accuracy is to 100 millisecond [i.e.: 0.1 seconds])
- Data was checked for errors and gaps (missing data)
- A queue estimation model was built from advanced and stop-bar detection boxes (paired with the traffic signal status [red or green]). However, error was high.
  - Alternatively, a queue estimation model was used from the HiRes TMC data.
- Visualization:
  - HiRes Stop Bar Detection data was graphed for visual inspection of the arrivals on Red and Green.
- Recreated Stop-Bar Detection, signal status (and time left on red or green), headway (closeness of vehicles), speed, and historical TMC (some of those items were not accurate enough and were not used in the model).
- Model – Queue prediction (classification) discussed next section

Methodology and Schema

DATA DIMENSION: 1343 x 19 (final set)

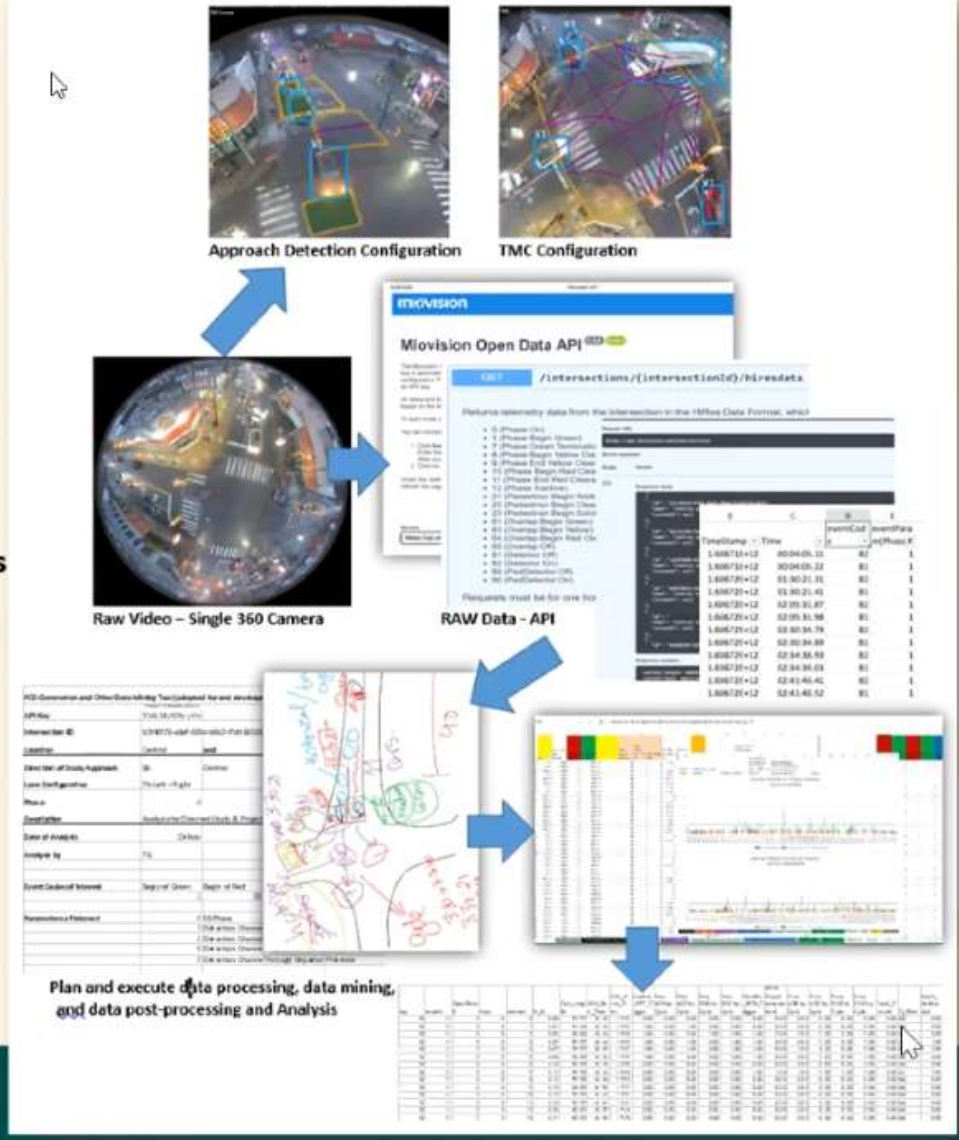
#	Attributes	Description	Example
1	Hour	Hour during which the data point occurred	10
2	minutes	Minute during which the data point occurred	20
3	H_M	Hour+Minute (fraction) during which the data point occurred	10.33
4	Cycl_Length	Cycle length in Seconds (to the 1/10th of a second)	59.97
5	SGN_Red_Time	Red Time in Seconds (to the 1/10th of a second)	40.55
6	SGN_Green_Time	Green Time in Seconds (to the 1/10th of a second)	19.42
7	StopBar_LEFT_Trigger	Trigger of detector box at stop bar during the cycle for the Left Turn lane	1
8	Freq. AOR by Cycle	Number of Vehicles Arriving on Red during Cycle (Left Turn)	1
9	Freq. AOG by Cycle	Number of Vehicles Arriving on Green during Cycle (Left Turn)	0
10	Freq. DOR by Cycle	Number of Vehicles Departing on Red during Cycle (Left Turn)	0
11	Freq. DOG by Cycle	Number of Vehicles Departing on Green during Cycle (Left Turn)	1
12	StopBar_RtTh_Trigger	Trigger of detector box at stop bar during the cycle for the Through+Right Turn lane	1
13	RTOR Depart (assumption)	Number of Vehicles Turning Right on Red (RTOR) during Cycle (Th+Rt lane)	0
14	Freq. AOR by Cycle	Number of Vehicles Arriving on Red during Cycle (Th+Rt lane)	1
15	Freq. AOG by Cycle	Number of Vehicles Arriving on Green during Cycle (Th+Rt lane)	0
16	Freq. DOR by Cycle	Number of Vehicles Departing on Red during Cycle (Th+Rt lane)	0
17	Freq. DOG by Cycle	Number of Vehicles Departing on Green during Cycle (Th+Rt lane)	1
18	Truck_Present	Presence of trucks during cycle	0
19	Q_filled	**Classification**: NQ, LQ, FQ (see provided table)	LQ

Distribution of Variables: NQ=1 , LQ=2, FQ= 3

Queue Classification Category	NQ	LQ	FQ
Category Numerical Code	1	2	3
Category Values	0	1-5	6 and above
Description	No cars during cycle	1 to 5 vehicles queued during cycle	6 or more vehicles queued during cycle
frequency	570	677	111

Basic Class Descriptive Statistics

MaxQ_NotNeeded	
Mean	1.865243
Standard Error	0.058582
Median	1
Mode	0
Standard Deviation	2.158826
Sample Variance	4.66053
Kurtosis	0.039275
Skewness	1.010565
Range	8
Minimum	0
Maximum	8
Sum	2533
Count	1358
Confidence Level(95.0%)	0.114922



MODEL BUILDING:

Data- Non-Linear
Model –Non-Linear

Models Chosen-

- **1. Logistic Regression**
- **2. Decision Tree**
- **3. Linear Discriminant Analysis**
- **4. K nearest Neighbors**
- **5. Gaussian Naïve Bayes**
- **6. SVM**
- **7. NN -Multilayer Perceptron**

Model Comparison:

MODEL	TRAINING ACCURACY	TESTING ACCURACY
1. Logistic Regression	0.82	0.83
2. Decision Tree	1	0.75
3. Linear Discriminant Analysis	0.81	0.82
4. K Nearest Neighbors	0.85	0.82
5. Gaussian Naïve Bayes	0.52	0.52
6. SVM	0.85	0.85
7. NN -Multilayer Perceptron	0.85	0.85

MODEL BUILDING:

Top-performing Models discovered

- **KNN-K-Nearest Neighbors**
- **SVM**
- **NN-Perceptron**

MODEL	TRAINING ACCURACY	TESTING ACCURACY
K Nearest Neighbors	0.85	0.82
SVM	0.85	0.85
NN -Multilayer Perceptron	0.85	0.85

```
#KNN
print('Confusion Matrix and Precision-Recall for KNN')
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
pred2 = knn.predict(X_test)
print(confusion_matrix(y_test, pred2))
print(classification_report(y_test, pred2))
```

SVM

```
Confusion Matrix and Precision-Recall for SVM
[[160   8   0]
 [ 22 180   1]
 [  2  27   3]]
              precision    recall  f1-score   support

           1       0.87      0.95      0.91       168
           2       0.84      0.89      0.86       203
           3       0.75      0.09      0.17        32

    accuracy                           0.85       403
   macro avg       0.82      0.64      0.65       403
weighted avg       0.84      0.85      0.83       403
```

MLP

```
Confusion Matrix and Precision-Recall for MLP
[[160   8   0]
 [ 22 179   2]
 [  0  28   4]]
              precision    recall  f1-score   support

           1       0.88      0.95      0.91       168
           2       0.83      0.88      0.86       203
           3       0.67      0.12      0.21        32

    accuracy                           0.85       403
   macro avg       0.79      0.65      0.66       403
weighted avg       0.84      0.85      0.83       403
```

KNN

```
Confusion Matrix and Precision-Recall for KNN
[[150  18   0]
 [ 21 174   8]
 [  0  24   8]]
              precision    recall  f1-score   support

           1       0.88      0.89      0.88       168
           2       0.81      0.86      0.83       203
           3       0.50      0.25      0.33        32

    accuracy                           0.82       403
   macro avg       0.73      0.67      0.68       403
weighted avg       0.81      0.82      0.81       403
```

KNN gives us a balanced classification accuracy, for instance it predicts class FQ=3 more accurately than other two classifiers.

MODEL BUILDING: **WINNER MODEL**

Confusion Matrix and Precision-Recall for SVM

```
[[160   8   0]
 [ 22 180   1]
 [  2  27   3]]
              precision    recall  f1-score   support

           1       0.87      0.95      0.91       168
           2       0.84      0.89      0.86       203
           3       0.75      0.09      0.17        32

    accuracy                           0.85       403
   macro avg       0.82      0.64      0.65       403
weighted avg       0.84      0.85      0.83       403
```

Confusion Matrix and Precision-Recall for KNN

```
[[150  18   0]
 [ 21 174   8]
 [  0  24   8]]
              precision    recall  f1-score   support

           1       0.88      0.89      0.88       168
           2       0.81      0.86      0.83       203
           3       0.50      0.25      0.33        32

    accuracy                           0.82       403
   macro avg       0.73      0.67      0.68       403
weighted avg       0.81      0.82      0.81       403
```

Q_filled	Correctly classified SVM	Correctly classified MLP	Correctly classified KNN
NQ=1	160	160	150
LQ=2	180	179	174
FQ=3	3	4	8

Confusion Matrix and Precision-Recall for MLP

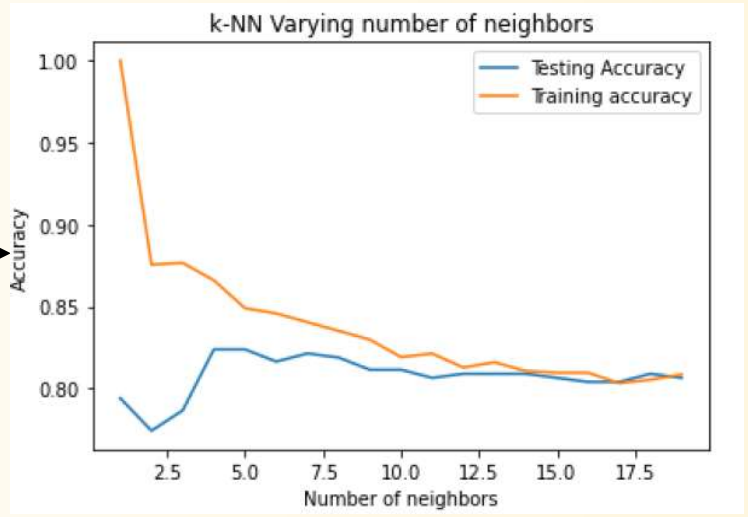
```
[[160   8   0]
 [ 22 179   2]
 [  0  28   4]]
              precision    recall  f1-score   support

           1       0.88      0.95      0.91       168
           2       0.83      0.88      0.86       203
           3       0.67      0.12      0.21        32

    accuracy                           0.85       403
   macro avg       0.79      0.65      0.66       403
weighted avg       0.84      0.85      0.83       403
```

KNN gives us a balanced classification accuracy, for instance it predicts class FQ=3 more accurately than other two classifiers**.**

WINNER MODEL: KNN K-Nearest Neighbors

KNN : Choosing K

```
neighbors = np.arange(1,20)
train_accuracy =np.empty(len(neighbors))
test_accuracy = np.empty(len(neighbors))

for i,k in enumerate(neighbors):
    #Setup a knn classifier with k neighbors
    knn = KNeighborsClassifier(n_neighbors=k)

    #Fit the model
    knn.fit(X_train, y_train)

    #Compute accuracy on the training set
    train_accuracy[i] = knn.score(X_train, y_train)

    #Compute accuracy on the test set
    test_accuracy[i] = knn.score(X_test, y_test)


plt.title('k-NN Varying number of neighbors')
plt.plot(neighbors, test_accuracy, label='Testing Accuracy')
plt.plot(neighbors, train_accuracy, label='Training accuracy')
plt.legend()
plt.xlabel('Number of neighbors')
plt.ylabel('Accuracy')
plt.show()
```



Q_filled	Correctly classified SVM	Correctly classified MLP	Correctly classified KNN
NQ=1	160	160	150
LQ=2	180	179	174
FQ=3	3	4	8

KNN :

At K= 2 to 3 → Training accuracy is high but testing accuracy is low

→ Training error low and testing error high

At K=5 → Testing and training accuracy are both optimal

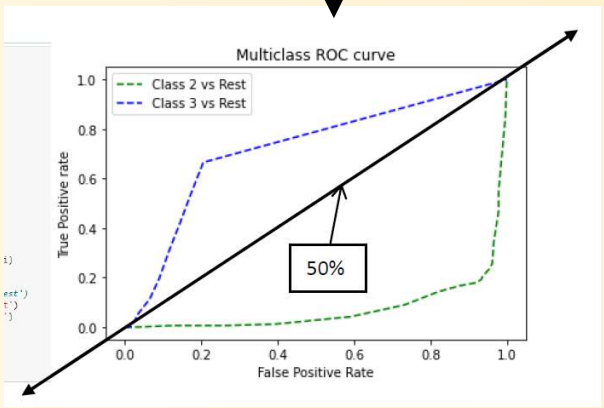
ROC Curve: LQ=2 vs FQ=3

INFERNCE:

- LQ=2 → KNN model is a **very good model (above 50% chance)**
- FQ=3 → KNN model is a **bad model (below 50%).**

- But **KNN** offers much better and balanced than any other model tried so far.
- LQ=2 and FQ=3 are penalizing each other
  Predicting one better than other possible.
- Trade-off between these needs to be based on objective/cost function.
- Practically speaking, we need both to be predicted accurately.

```
# 2-class classification

from sklearn.model_selection import train_test_split
from sklearn.metrics import roc_curve
from sklearn.metrics import roc_auc_score


pred = knn.predict(X_test)
pred_prob = knn.predict_proba(X_test)

# roc curve for classes
fpr = {}
tpr = {}
thresh ={}

n_class = 3

for i in range(1,n_class):
    fpr[i], tpr[i], thresh[i] = roc_curve(y_test, pred_prob[:,i], pos_label=i)

# plotting
#plt.plot(fpr[0], tpr[0], linestyle='--',color='orange', label='Class 1 vs Rest')
plt.plot(fpr[1], tpr[1], linestyle='--',color='green', label='Class 2 vs Rest')
plt.plot(fpr[2], tpr[2], linestyle='--',color='blue', label='Class 3 vs Rest')
plt.title('Multiclass ROC curve')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive rate')
plt.legend(loc='best')
plt.savefig('Multiclass ROC',dpi=300);
```



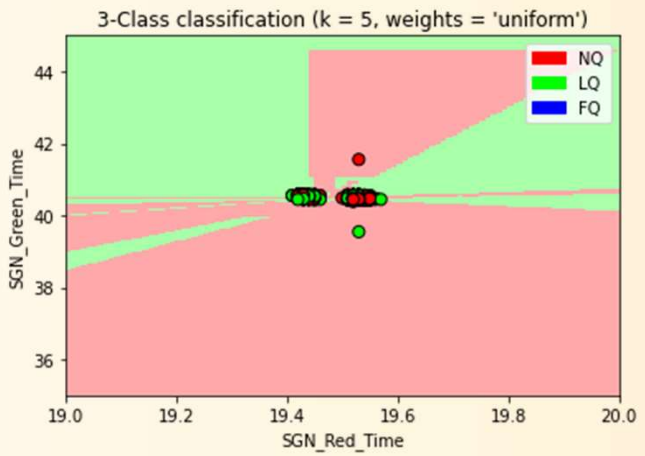
Classification Boundary

```
import matplotlib.cm as cm
from matplotlib.colors import ListedColormap, BoundaryNorm
import matplotlib.patches as mpatches
import matplotlib.patches as mpatches

def plot_Queue_knn(X_train, y_train, n_neighbors, weights):

    XXX= data[['SGN_Green_Time','SGN_Red_Time']]
    yyy= data['Q_filled']

    X_mat = XXX.to_numpy()
    y_mat = yyy.to_numpy()

    # Create color maps
    cmap_light = ListedColormap(['#FFAAAA', '#AAFFAA', '#AAAAFF'])
    cmap_bold  = ListedColormap(['#FF0000', '#00FF00', '#0000FF'])

    knn2 = KNeighborsClassifier(n_neighbors, weights=weights)
    knn2.fit(X_mat, y_mat)

    # Plot the decision boundary by assigning a color in the color map to each mesh point.
    mesh_step_size = .01  # step size in the mesh
    plot_symbol_size = 50

    x_min, x_max = X_mat[:, 0].min() - 1, X_mat[:, 0].max() + 1
    y_min, y_max = X_mat[:, 1].min() - 1, X_mat[:, 1].max() + 1
    xx, yy = np.meshgrid(np.arange(x_min, x_max, mesh_step_size),
                         np.arange(y_min, y_max, mesh_step_size))
    Z = knn2.predict(np.c_[xx.ravel(), yy.ravel()])

    # Put the result into a color plot
    Z = Z.reshape(xx.shape)
    plt.figure()
    plt.pcolormesh(xx, yy, Z, cmap=cmap_light)

    # Plot training points
    plt.scatter(X_mat[:, 0], X_mat[:, 1], s=plot_symbol_size, c=y, cmap=cmap_bold, edgecolor = 'black')
    plt.xlim(19, 20)
    plt.ylim(35, 45)
    patch0 = mpatches.Patch(color='#FF0000', label='NQ')
    patch1 = mpatches.Patch(color='#00FF00', label='LQ')
    patch2 = mpatches.Patch(color='#0000FF', label='FQ')
    plt.legend(handles=[patch0, patch1, patch2])


    plt.xlabel('SGN_Red_Time')
    plt.ylabel('SGN_Green_Time')
    plt.title("3-Class classification (k = %i, weights = '%s')"
              % (n_neighbors, weights))

    plt.show()

plot_Queue_knn(X_train,y_train,  5, 'uniform')
```

INFERNCE:

- For NQ=1 and LQ=2 a clear boundary is visible.
- But for FQ=3 there is no a clear boundary visible

- Reasons:
  1. Unbalanced class distribution (FQ=3 significantly low)
  2. LQ=2 and FQ=3 penalization issue.



CONCLUSION

- Difficult to predict Queue length solely from Stop bar detection –
- This project is a first attempt at doing so with a single low investment single-source detection (potential trade: accuracy for price and effort)
- Various prediction classification models were tested to provide a comparison (Logistic Regression, Decision Tree, Linear Discriminant Analysis, K Nearest Neighbors, Gaussian Naïve Bayes, SVM, and NN -Multilayer Perceptron).
- **Top-performing Models: KNN-K-Nearest Neighbors, SVM, NN-Perceptron**
- **K nearest neighbor (KNN) seems to be outperforming the other methods**
  - Predicting NQ and LQ classes: model was accurate.
  - Predicting FQ class: fell short in all methods and needs further investigation.
- Potential reason why results were not as strongly predicted along the three classes:
  - Uneven class distribution (with much lower instances of FQ compared with the other two classes)
  - potential data accuracy issues
  - Further review and refinement of analysis methods and configurations needed
  - need for a larger data pool