Homework #7: Boosting and Support Vector Machines

You're working for a car manufacturer that is looking to implement driver assistance features such as automated steering and adaptive cruise control. While technologically advanced, these systems still require driver attention. Some manufacturers simply require keeping your hands on the wheel but your company would also like to ensure the driver's focus remains on the road. To accomplish this, they'd like you to construct a model that can use the position of facial features to determine whether the driver is looking straight or not.

A separate system has been used to extract the eye, mouth, and nose positions from images taken of the driver, your goal is to use these features to predict the direction of the driver's gaze. The dataset listed below has been provided for these tasks.

Relevant Dataset

`drivPoints.txt`

- Response Variable: `label`. Note: this includes looking left, right, and straight. We will convert this to a binary response.
- Predictor Variables:
  - [`xF` `yF` `wF` `hF`] = face position
  - [`xRE` `yRE`] = rigth eye position
  - [`xLE` `yL`] = left eye position
  - [`xN` `yN`] = Nose position
  - [`xRM` `yRM`] = right corner of mouth
  - [`xLM` `yLM`] = left corner of mouth

Source

https://archive.ics.uci.edu/ml/datasets/DrivFace

Task 1: Import the dataset and create a binary variable of `lookingStraight`. Split into train/test set.

This variable should take the value of `1` when `label=2` and `0` everywhere else. There should be a large class imbalance between looking straight or not (which you would expect given the people are driving).

In [4]:
```
# Import libraries
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.svm import SVC
from sklearn.metrics import f1_score
import matplotlib.pyplot as plt
import numpy as np
```

In [5]:
```
# Read in the dataset
df = pd.read_csv('drivPoints.txt')

# Create variable for 'lookingStraight'
df['lookingStraight'] = (df['label'] == 2).astype(int)

# Split into train and test sets
X = df[['xF', 'yF', 'wF', 'hF', 'xRE', 'yRE', 'xLE', 'yLE', 'xN', 'yN', 'xRM',
y = df['lookingStraight']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, rando

# Display class distribution
print(y.value_counts())
```

```
lookingStraight
1    546
0     60
Name: count, dtype: int64
```

Task 2: Perform a cross-validated (or use a single validation set) grid search of the hyperparameters for the `GradientBoostingClassifier` to find the best model.

You should at least tune the learning rate and number of trees in the model but feel free to go as deep as you'd like on this analysis).

In [7]:
```
# Define model
gbc = GradientBoostingClassifier()

# Parameter grid
param_grid = {
    'learning_rate': [0.01, 0.1, 0.2],
    'n_estimators': [100, 200, 300]
}

# Perform grid search with cross-validation
grid_search_gbc = GridSearchCV(estimator=gbc, param_grid=param_grid, cv=5, sco
grid_search_gbc.fit(X_train, y_train)

# Print the best parameters and its score
print(f"Best parameters for GradientBoostingClassifier: {grid_search_gbc.best_
print(f"Best cross-validation score: {grid_search_gbc.best_score_}")
```

```
Best parameters for GradientBoostingClassifier: {'learning_rate': 0.1, 'n_esti
mators': 200}
Best cross-validation score: 0.9772981099656357
```

Task 3: Perform a cross-validated (or use a single validation set) grid search of the hyperparameters for the `SVC` (Support Vector Classifier) to find the best model.

You should at least tune `C` and the `kernel` but feel free to go as deep as you'd like on this analysis).

In [9]:
```python
# Define model
svc = SVC()

# Parameter grid
param_grid = {
    'C': [0.1, 1, 10],
    'kernel': ['linear', 'rbf', 'poly']
}

# Perform grid search with cross-validation
grid_search_svc = GridSearchCV(estimator=svc, param_grid=param_grid, cv=5, sco
grid_search_svc.fit(X_train, y_train)

# Print the best parameters its score
print(f"Best parameters for SVC: {grid_search_svc.best_params_}")
print(f"Best cross-validation score: {grid_search_svc.best_score_}")
```

```
Best parameters for SVC: {'C': 10, 'kernel': 'poly'}
Best cross-validation score: 0.9277061855670101
```

Questions

1. Is accuracy the best metric to use in these tasks or would there have been a better one? Explain.

- Because the class shows more people are looking straight than not, accuracy may not be the best metric. F1 score, precision, or recall might prove to be better to take into account that class imbalance. They would consider false positives and negatives. F1 score would give a good balance between precision and recall.

1. Which model gave the "best" result using the metric you chose above?

- GradientBoostingClassifier based on the F1 scores and CV scores

1. (Bonus) Any other interesting insights from this model or data?

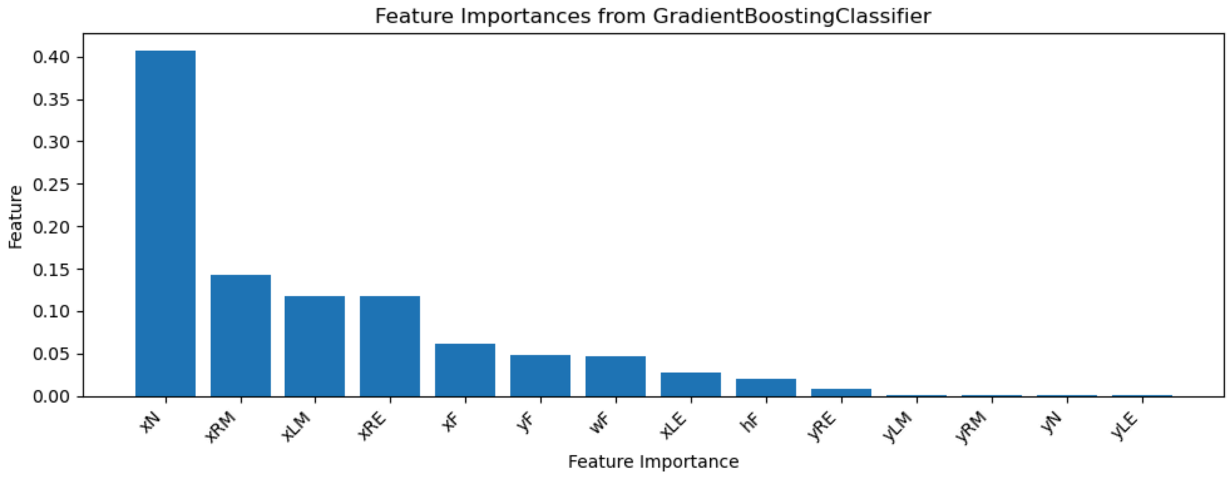
- The most important features from the model are xN, xRM, xLM, xRE, and xF, which suggests the horizontal position of all the facial features is very important in determining when the gaze is straight. This makes sense that they would be the most crucial features because as a person looks left to right, the horizontal position of these features changes drastically. By focusing on these features, systems could be created that help the drivers be alert and focused, notifying them when their features are out of position, suggesting they are inactively driving.

Work for 2 and 3 are found below

In [11]:
```
## Question 2

# Predict on test set
y_pred_gbc = grid_search_gbc.predict(X_test)
y_pred_svc = grid_search_svc.predict(X_test)

# F1-score for both models
f1_gbc = f1_score(y_test, y_pred_gbc)
f1_svc = f1_score(y_test, y_pred_svc)

print(f"F1-score for GradientBoostingClassifier: {f1_gbc}")
print(f"F1-score for SVC: {f1_svc}")

best_model = "GradientBoostingClassifier" if f1_gbc > f1_svc else "SVC"
print(f"The best model is: {best_model}")
```

```
F1-score for GradientBoostingClassifier: 0.981651376146789
F1-score for SVC: 0.9727272727272727
The best model is: GradientBoostingClassifier
```

In [12]:
```
## Question 3

# Feature importances from GradientBoostingClassifier
feature_importances = grid_search_gbc.best_estimator_.feature_importances_
features = X.columns
sorted_indices = np.argsort(feature_importances)[::-1]
sorted_features = features[sorted_indices]
sorted_importances = feature_importances[sorted_indices]

# Bar plot
plt.figure(figsize=(10, 4))
plt.bar(sorted_features, sorted_importances)
plt.xlabel("Feature Importance")
plt.ylabel("Feature")
plt.title("Feature Importances from GradientBoostingClassifier")
plt.xticks(rotation=45, ha='right')
plt.tight_layout()
plt.show()
```


Feature Importances from GradientBoostingClassifier
Feature
Feature Importance
0.00
0.05
0.10
0.15
0.20
0.25
0.30
0.35
0.40
xN
xRM
xLM
xRE
xF
yF
wF
xLE
hF
yRE
yLM
yRM
yN
yLE