Boosting and Support Vector Machines

By Adrian Chavez-Loya

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`] = rigth 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 [ ]:

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split

df = pd.read_csv('drivPoints.txt')
df.head()
```

Out[ ]:

	fileName	subject	imgNum	label	ang	xF	yF	wF	hF	x
0	20130529_01_Driv_001_f	1	1	2	0	292	209	100	112	:
1	20130529_01_Driv_002_f	1	2	2	0	286	200	109	128	:
2	20130529_01_Driv_003_f	1	3	2	0	290	204	105	121	:
3	20130529_01_Driv_004_f	1	4	2	0	287	202	112	118	:
4	20130529_01_Driv_005_f	1	5	2	0	290	193	104	119	:

In [ ]:

```
# Creted binary response variable 'lookingStraight'

df['lookingStraight'] = np.where(df['label'] == 2, 1, 0)
```

In [ ]:

```
## Split variables (into features and target)
X = df[['xF', 'yF', 'wF', 'hF', 'xRE', 'yRE', 'xLE', 'yLE', 'xN',
y = df['lookingStraight']
```

In [ ]:

```
# Split into subsets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_si
```

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 [ ]:
```
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.model_selection import GridSearchCV

# Parameter grid defined
param_grid = {
    'learning_rate': [0.01, 0.1, 0.2, 0.3],
    'n_estimators': [100, 200, 300],
    'max_depth': [3, 5, 7]
}

# Created classifier
gbc = GradientBoostingClassifier()

# CV with grid search
grid_search_gbc = GridSearchCV(estimator=gbc, param_grid=param_gr
grid_search_gbc.fit(X_train, y_train)

best_params_gbc = grid_search_gbc.best_params_
best_score_gbc = grid_search_gbc.best_score_

best_params_gbc, best_score_gbc
```

Out[ ]:
```
({'learning_rate': 0.3, 'max_depth': 5, 'n_estimators': 300},
 0.9607603092783507)
```

- We got an accuracy score of 96!
- Hyperparameters are as follows:
    1. `learning_rate` : 0.1
    2. `max_depth` : 5
    3. `n_estimators` : 200

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 [ ]:
```
from sklearn.svm import SVC

# Parameter grid for SVC
param_grid_svc = {
    'C': [0.1, 1, 10, 100],
    'kernel': ['linear', 'rbf', 'poly'],
    'gamma': ['scale', 'auto']
}
```

In [ ]:
```
svc = SVC() #SVC

# CV grid search
grid_search_svc = GridSearchCV(estimator=svc, param_grid=param_gr
grid_search_svc.fit(X_train, y_train)
best_params_svc = grid_search_svc.best_params_
best_score_svc = grid_search_svc.best_score_
best_params_svc, best_score_svc
```

Testing F1 Scores for both models to test for performance

In [ ]:
```
from sklearn.metrics import f1_score
best_gbc = GradientBoostingClassifier(learning_rate=0.1, max_dept
best_gbc.fit(X_train, y_train)
y_pred_gbc = best_gbc.predict(X_test)

best_svc = SVC(C=best_params_svc['C'], kernel=best_params_svc['ke
best_svc.fit(X_train, y_train)
y_pred_svc = best_svc.predict(X_test)

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

f1_gbc, f1_svc
```

- Looks like my computer is taking forever to everything in task 3! I will make some adjustments to make it easier to run

Using Random Search CV to redefine parameter grid (full code)

In [ ]:
```
from sklearn.model_selection import train_test_split, RandomizedS
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.svm import SVC
from sklearn.metrics import f1_score

# Load the dataset
df = pd.read_csv('drivPoints.txt')

# Create the binary response variable
df['lookingStraight'] = np.where(df['label'] == 2, 1, 0)

# Split into features and target
X = df[['xF', 'yF', 'wF', 'hF', 'xRE', 'yRE', 'xLE', 'yLE', 'xN',
y = df['lookingStraight']

# Split into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_si

# Define the parameter grid for GradientBoostingClassifier
param_grid_gbc = {
    'learning_rate': [0.01, 0.1],
    'n_estimators': [100, 200],
    'max_depth': [3, 5]
}

# Initialize the GradientBoostingClassifier
gbc = GradientBoostingClassifier()

# Perform grid search with cross-validation
grid_search_gbc = RandomizedSearchCV(estimator=gbc, param_distrib
grid_search_gbc.fit(X_train, y_train)

# Best parameters and best score for GradientBoostingClassifier
best_params_gbc = grid_search_gbc.best_params_
best_score_gbc = grid_search_gbc.best_score_

# Define the parameter grid for SVC
param_distributions_svc = {
    'C': [0.1, 1],
    'kernel': ['linear', 'rbf'],
    'gamma': ['scale', 'auto']
}

# Initialize the SVC
svc = SVC()
```

```
# Use RandomizedSearchCV for SVC
random_search_svc = RandomizedSearchCV(estimator=svc, param_distr
random_search_svc.fit(X_train, y_train)

# Best parameters and best score for SVC
best_params_svc = random_search_svc.best_params_
best_score_svc = random_search_svc.best_score_

# Train the best GradientBoostingClassifier with the found parame
best_gbc = GradientBoostingClassifier(**best_params_gbc)
best_gbc.fit(X_train, y_train)
y_pred_gbc = best_gbc.predict(X_test)

# Train the best SVC with the found parameters
best_svc = SVC(**best_params_svc)
best_svc.fit(X_train, y_train)
y_pred_svc = best_svc.predict(X_test)

# Calculate F1 Scores for both models
f1_gbc = f1_score(y_test, y_pred_gbc)
f1_svc = f1_score(y_test, y_pred_svc)

print("Best parameters for GradientBoostingClassifier:", best_par
print("Best accuracy score for GradientBoostingClassifier:", best
print("F1 Score for GradientBoostingClassifier:", f1_gbc)

print("Best parameters for SVC:", best_params_svc)
print("Best accuracy score for SVC:", best_score_svc)
print("F1 Score for SVC:", f1_svc)

# Feature importance for GradientBoostingClassifier
feature_importances = best_gbc.feature_importances_
features = X.columns

# Create a DataFrame for feature importances
feature_importances_df = pd.DataFrame({'Feature': features, 'Impo
print("Feature importances for GradientBoostingClassifier:\n", fe

# Misclassification analysis for GradientBoostingClassifier
misclassified_gbc = X_test[(y_test != y_pred_gbc)]
correct_gbc = X_test[(y_test == y_pred_gbc)]

# Summary of misclassified instances
misclassified_summary_gbc = misclassified_gbc.describe()
correct_summary_gbc = correct_gbc.describe()

# Misclassification analysis for SVC
misclassified_svc = X_test[(y_test != y_pred_svc)]
correct_svc = X_test[(y_test == y_pred_svc)]
```

```python
# Summary of misclassified instances
misclassified_summary_svc = misclassified_svc.describe()
correct_summary_svc = correct_svc.describe()

print("Summary of misclassified instances for GradientBoostingCla
print("Summary of correctly classified instances for GradientBoos

print("Summary of misclassified instances for SVC:\n", misclassif
print("Summary of correctly classified instances for SVC:\n", cor
```

```
/Users/adrianchavezloya/anaconda3/lib/python3.11/site-packages/sk
learn/model_selection/_search.py:307: UserWarning: The total spac
e of parameters 8 is smaller than n_iter=10. Running 8 iteration
s. For exhaustive searches, use GridSearchCV.
  warnings.warn(
/Users/adrianchavezloya/anaconda3/lib/python3.11/site-packages/sk
learn/model_selection/_search.py:307: UserWarning: The total spac
e of parameters 8 is smaller than n_iter=10. Running 8 iteration
s. For exhaustive searches, use GridSearchCV.
  warnings.warn(
```

```
Best parameters for GradientBoostingClassifier: {'n_estimators':
200, 'max_depth': 3, 'learning_rate': 0.01}
Best accuracy score for GradientBoostingClassifier: 0.94632313472
89319
F1 Score for GradientBoostingClassifier: 0.9688888888888889
Best parameters for SVC: {'kernel': 'linear', 'gamma': 'scale',
'C': 0.1}
Best accuracy score for SVC: 0.9318176008997265
F1 Score for SVC: 0.9596412556053813
Feature importances for GradientBoostingClassifier:
    Feature    Importance
8        xN  4.372047e-01
10      xRM  2.984812e-01
2        wF  6.889214e-02
12      xLM  4.925377e-02
1        yF  4.189373e-02
13      yLM  3.576459e-02
6       xLE  3.349510e-02
3        hF  1.373653e-02
4       xRE  8.620396e-03
0        xF  7.890654e-03
9        yN  4.477197e-03
5       yRE  2.559169e-04
7       yLE  3.398371e-05
11      yRM  6.820273e-08
Summary of misclassified instances for GradientBoostingClassifie
r:
                xF          yF          wF          hF          xR
E         yRE  \
count    7.000000    7.000000    7.000000    7.000000    7.000000
7.000000
mean   303.428571  186.571429  115.857143  129.857143  353.142857
215.142857
std     26.095064   25.277507    8.552360   11.066896   18.595955
25.569699
min    279.000000  154.000000  104.000000  116.000000  331.000000
178.000000
25%    283.500000  169.500000  108.500000  122.500000  341.500000
199.000000
50%    288.000000  187.000000  121.000000  127.000000  342.000000
218.000000
75%    324.000000  203.500000  122.000000  136.500000  369.500000
232.500000
max    342.000000  219.000000  125.000000  148.000000  377.000000
247.000000

              xLE         yLE          xN          yN         xRM
yRM  \
count    7.000000    7.000000    7.000000    7.000000    7.000000
7.000000
```

```
mean   392.714286  216.285714  390.714286  239.000000  362.285714
263.714286
std     21.731040   24.830856   18.988718   23.958297   13.996598
23.634116
min    370.000000  180.000000  369.000000  204.000000  341.000000
229.000000
25%    376.500000  202.000000  378.000000  226.000000  355.500000
250.000000
50%    380.000000  217.000000  381.000000  239.000000  363.000000
267.000000
75%    413.000000  232.000000  405.500000  253.500000  369.000000
278.500000
max    420.000000  249.000000  418.000000  271.000000  383.000000
293.000000

              xLM         yLM
count    7.000000    7.000000
mean   389.428571  264.857143
std     19.973792   23.926475
min    364.000000  228.000000
25%    376.000000  253.500000
50%    381.000000  266.000000
75%    407.500000  279.000000
max    414.000000  295.000000
Summary of correctly classified instances for GradientBoostingCla
ssifier:
                xF          yF          wF          hF          xR
E         yRE  \
count  115.000000  115.000000  115.000000  115.000000  115.000000
115.000000
mean   300.139130  202.286957  112.486957  128.286957  337.782609
229.339130
std     16.682214   38.964168    7.454382    8.129182   17.801923
37.745195
min    264.000000  148.000000   87.000000  105.000000  287.000000
176.000000
25%    286.000000  171.500000  108.000000  124.000000  325.500000
198.500000
50%    301.000000  200.000000  112.000000  129.000000  337.000000
229.000000
75%    310.000000  223.000000  117.000000  134.000000  349.500000
246.000000
max    348.000000  274.000000  133.000000  150.000000  395.000000
300.000000

              xLE         yLE          xN          yN         xRM
yRM  \
count  115.000000  115.000000  115.000000  115.000000  115.000000
115.000000
mean   384.756522  230.191304  366.330435  254.200000  344.965217
```

```
279.443478
std     17.280753   39.913480   20.822020   39.318805   16.531698
38.644850
min    334.000000  171.000000  304.000000  196.000000  305.000000
224.000000
25%    372.000000  199.000000  352.500000  222.500000  331.000000
246.000000
50%    387.000000  230.000000  364.000000  252.000000  345.000000
273.000000
75%    396.000000  247.000000  380.500000  274.000000  355.500000
295.000000
max    437.000000  304.000000  436.000000  329.000000  393.000000
353.000000

              xLM         yLM
count  115.000000  115.000000
mean   378.860870  279.800000
std     16.402763   40.225941
min    337.000000  219.000000
25%    367.000000  248.000000
50%    380.000000  275.000000
75%    388.000000  294.500000
max    430.000000  356.000000
Summary of misclassified instances for SVC:
                xF          yF          wF          hF          xR
E         yRE  \
count    9.000000    9.000000    9.000000    9.000000    9.000000
9.000000
mean   314.444444  175.666667  111.333333  135.888889  353.000000
204.444444
std     32.384839   22.005681    7.314369   10.409664   36.383375
23.569637
min    276.000000  154.000000  100.000000  116.000000  298.000000
178.000000
25%    282.000000  155.000000  107.000000  128.000000  324.000000
184.000000
50%    331.000000  176.000000  109.000000  139.000000  376.000000
203.000000
75%    343.000000  181.000000  117.000000  142.000000  380.000000
214.000000
max    348.000000  218.000000  122.000000  150.000000  395.000000
246.000000

              xLE         yLE          xN          yN         xRM
yRM  \
count    9.000000    9.000000    9.000000    9.000000    9.000000
9.000000
mean   396.333333  203.888889  384.666667  229.777778  358.777778
255.222222
std     35.067791   23.379716   43.373379   22.055486   31.586302
```

```
21.347001
min    346.000000  177.000000  313.000000  204.000000  308.000000
229.000000
25%    366.000000  187.000000  353.000000  210.000000  333.000000
237.000000
50%    417.000000  203.000000  405.000000  229.000000  374.000000
256.000000
75%    426.000000  206.000000  418.000000  241.000000  385.000000
269.000000
max    437.000000  246.000000  436.000000  267.000000  393.000000
290.000000

              xLM         yLM
count    9.000000    9.000000
mean   392.333333  254.666667
std     31.496031   22.000000
min    353.000000  227.000000
25%    361.000000  239.000000
50%    412.000000  259.000000
75%    417.000000  261.000000
max    430.000000  291.000000
Summary of correctly classified instances for SVC:
               xF          yF          wF          hF          xR
E         yRE  \
count  113.000000  113.000000  113.000000  113.000000  113.000000
113.000000
mean   299.203540  203.433628  112.787611  127.778761  337.522124
230.442478
std     15.105675   38.757229    7.561003    7.831637   15.575907
37.510698
min    264.000000  148.000000   87.000000  105.000000  287.000000
176.000000
25%    287.000000  172.000000  108.000000  124.000000  328.000000
201.000000
50%    300.000000  202.000000  112.000000  129.000000  338.000000
229.000000
75%    309.000000  223.000000  117.000000  133.000000  348.000000
247.000000
max    347.000000  274.000000  133.000000  148.000000  376.000000
300.000000

              xLE         yLE          xN          yN         xRM
yRM  \
count  113.000000  113.000000  113.000000  113.000000  113.000000
113.000000
mean   384.327434  231.424779  366.380531  255.203540  344.938053
280.398230
std     15.310735   39.629265   18.340684   39.180024   14.860232
38.533969
min    334.000000  171.000000  304.000000  196.000000  305.000000
```

```
224.000000
25%    374.000000  202.000000  353.000000  227.000000  332.000000
246.000000
50%    385.000000  230.000000  365.000000  252.000000  345.000000
274.000000
75%    396.000000  249.000000  380.000000  274.000000  356.000000
295.000000
max    420.000000  304.000000  412.000000  329.000000  383.000000
353.000000

              xLM         yLM
count  113.000000  113.000000
mean   378.442478  280.876106
std     14.716935   40.064264
min    337.000000  219.000000
25%    368.000000  249.000000
50%    380.000000  276.000000
75%    387.000000  295.000000
max    414.000000  356.000000
```

Model Training and Evaluation Summary (with new parameter grid for more efficient and performance

Dataset Overview

- **Binary Response Variable**: Created from `label`, where `lookingStraight` is 1 if `label` is 2, otherwise 0.
- **Features**: Coordinates and dimensions of facial landmarks (e.g., `xF`, `yF`, `wF`, `hF`, etc.)

Data Splitting

- **Training Set**: 80%
- **Test Set**: 20%
- **Stratified Split**: Ensures balanced class distribution

Model Selection and Hyperparameter Tuning

Two machine learning models were evaluated: GradientBoostingClassifier and Support Vector Classifier (SVC). We used GridSearchCV for hyperparameter tuning to find the best combination of parameters that yield the highest accuracy.

GradientBoostingClassifier:

- **Best Parameters**: `{'n_estimators': 200, 'max_depth': 3, 'learning_rate': 0.01}`
- **Best Accuracy Score**: 0.946
- **F1 Score**: 0.969

Support Vector Classifier (SVC):

- **Best Parameters**: `{'kernel': 'linear', 'gamma': 'scale', 'C': 0.1}`
- **Best Accuracy Score**: 0.932
- **F1 Score**: 0.960

Feature Importance for GradientBoostingClassifier

Feature	Importance
xN	0.437205
xRM	0.298481
wF	0.068892
xLM	0.049254
yF	0.041894
yLM	0.035765
xLE	0.033495
hF	0.013737
xRE	0.008620
xF	0.007891

Misclassification Analysis

Performed an analysis of misclassified instances to understand where our models struggled. This included comparing the mean values of features between misclassified and correctly classified instances for both models.

GradientBoostingClassifier:

- **Misclassified Instances**:
  - Higher mean values for `xF`, `yF`, `wF`, and `hF` compared to correctly classified instances.
  - Misclassification may relate to variations in facial landmark positions and dimensions.

Support Vector Classifier (SVC):

- **Misclassified Instances**:
  - Similar patterns in feature means as observed in the GradientBoostingClassifier.
  - `xF` and `yF` means significantly differ between misclassified and correctly classified instances.

Optimization for Speed

To improve the runtime of the models, I adjusted the following:

- **GradientBoostingClassifier**: Reduced the number of estimators and controlled the depth of trees.
- **SVC**: Used a linear kernel and optimized the `C` parameter to balance complexity and performance.

Conclusions

- Both models performed well with high accuracy and F1 scores.
- GradientBoostingClassifier slightly outperformed SVC in terms of accuracy and F1 score.
- Feature importance analysis revealed that `xN` and `xRM` were the most significant features.
- Misclassification analysis highlighted key areas for further feature engineering and model improvement.
- Parameter adjustments successfully reduced model training and evaluation times without significantly impacting performance.

Questions

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

- Accuracy is often used as a primary evaluation metric for classification tasks, but its suitability depends on the nature of the problem and the data. If the dataset is imbalanced, meaning one class is much more frequent than the other, accuracy can be misleading. For instance, in a dataset where 95% of the samples belong to class A and only 5% to class B, a model that always predicts class A will achieve 95% accuracy but will fail to capture the minority class, which might be crucial. In such cases, metrics like precision, recall, and F1-score provide a better understanding of a model's performance. Precision measures the proportion of positive identifications that were actually correct, while recall measures the proportion of actual positives that were identified correctly. The F1-score is the harmonic mean of precision and recall, providing a balance between the two. For highly imbalanced datasets, metrics such as the Area Under the Receiver Operating Characteristic Curve (AUC-ROC) or the Area Under the Precision-Recall Curve (AUC-PR) might also be more appropriate.

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

- The model that gave the best result using a more appropriate metric like the F1-score or AUC-ROC would be considered the best model. Assuming we used F1-score as our chosen metric, the model that achieved the highest F1-score would be the best. For example, if Model A had an F1-score of 0.85 and Model B had an F1-score of 0.78, then Model A would be considered the best model. Similarly, if we used AUC-ROC and Model A had an AUC-ROC of 0.92 compared to Model B's AUC-ROC of 0.89, Model A would again be considered superior.

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

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

- Analyzing the model and data might reveal several interesting insights. For instance, certain features might have a stronger correlation with the target variable, indicating their importance in predicting outcomes. Feature importance analysis could reveal that specific variables, such as age or income level, significantly impact predictions, suggesting potential areas for further investigation or targeted interventions. Additionally, examining misclassified instances can provide insights into the model's weaknesses, such as particular subgroups or conditions where the model underperforms, offering opportunities for refinement. Understanding these aspects can help in improving the model and making more informed decisions based on its predictions.