Report 2: modeling report

Classification of the problem

- **What kind of machine learning problem is your project like? (classification, regression, clustering, etc)**

I chose to go with the classification machine learning problem for the project. I realized that the problems can be viewed as both regression (if we want to predict the probability number), and classification, and I chose the classification. I realized that this might be simpler, and that the problem can be viewed through the the main objective, that is: to predict the outcome of the tennis matches, whether a player will win or lose. This would mean that our problem is a binary classification task.

- **What task does your project relate to? (fraud detection, facial recognition, sentiment analysis, etc)?**

I think none of the mentioned. Our project is focused on sports betting prediction, predicting the outcome of matches. We want to use machine learning to predict match outcomes for sports betting, and that seems different than tasks like fraud detection, facial recognition, or sentiment analysis.

- **What is the main performance metric used to compare your models? Why this one?**

I would consider F1 Score (F1 Score provides a balance between precision and recall, making it suitable when there is an uneven class distribution or when both false positives and false negatives are important).

- **Did you use other qualitative or quantitative performance metrics? If yes, detail it.**

Yes, I will look at the following (some or all of them):

- ***Accuracy:*** *Overall correctness of the model.*
- ***Precision:*** *Proportion of true positive predictions among all positive predictions.*
- ***Recall (Sensitivity):*** *Proportion of true positive predictions among all actual positive instances.*
- ***AUC-ROC:*** *Area under the Receiver Operating Characteristic curve, which measures the classifier's ability to distinguish between classes.*
- ***Confusion Matrix*** *- to understand where our models make mistakes*

Model choice and optimization

First of all I had to complete my preprocessing for the modeling part. Here it is:

```
1. # PREPROCESSING FOR MODELING
2.
3. from sklearn.model_selection import train_test_split
4. from sklearn.preprocessing import LabelEncoder, StandardScaler
5.
6. # separate the dataset into features (explanatory variables) and target variable
```

```
feats = df.drop(['Date', 'Winner', 'Loser'], axis=1)  # exclude non-numeric and target columns
target = df['Winner']

# One-hot encode categorical variables
categorical_cols = ['Surface', 'Round']
X_train_cat_encoded = pd.get_dummies(feats[categorical_cols])

# encode target variable 'Winner'
from sklearn.preprocessing import LabelEncoder
label_encoder = LabelEncoder()
target_encoded = label_encoder.fit_transform(target)

# combine the encoded categorical features with the imputed numerical features
feats_combined = pd.concat([feats.drop(categorical_cols, axis=1), X_train_cat_encoded], axis=1)

# split the dataset into training, validation, and test sets
X_train, X_temp, y_train, y_temp = train_test_split(feats_combined, target_encoded, test_size=0.3, random_state=42)
X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp, test_size=0.5, random_state=42)

# standardize combined features
scaler = StandardScaler()
X_train_final = scaler.fit_transform(X_train)
X_val_final = scaler.transform(X_val)
X_test_final = scaler.transform(X_test)
```

- **What algorithms have you tried?**

Logistic Regression, DecisionTreeClassifier, RandomForest, Support Vector Machine (SVM) algorithm.

1. Logistic Regression

```
# train Logistic Regression model
from sklearn.linear_model import LogisticRegression
logreg_model = LogisticRegression(random_state=42)
logreg_model.fit(X_train_final, y_train)

# evaluate the model on training and test data
train_score = logreg_model.score(X_train_final, y_train)
test_score = logreg_model.score(X_test_final, y_test)

print(f"Training Score: {train_score:.4f}")
print(f"Test Score: {test_score:.4f}")

# evaluate the model
from sklearn.metrics import classification_report, confusion_matrix
y_pred = logreg_model.predict(X_test_final)

# display classification report
print("\nClassification Report:")
print(classification_report(y_test, y_pred))

# F1 score
overall_f1 = f1_score(y_test, y_pred, average='weighted')
print(f"\nOverall F1 Score: {overall_f1:.4f}")

# confusion matrix
```

```
conf_matrix = confusion_matrix(y_test, y_pred)
print("\nConfusion Matrix:")
print(conf_matrix)

```

Results:

```
Training Score: 0.1368
Test Score: 0.1083

Classification Report:
precision recall f1-score support
2 0.14 0.05 0.08
39 3 0.27 0.20
0.23 60 4 0.00
0.00 0.00 1 5
0.00 0.00 0.00 0.8
0.00 0.00 0.00 8
9 0.00 0.00 0.00
.
.
.
accuracy 0.11 11177
macro avg 0.03 0.03 0.03 11177
weighted avg 0.09 0.11 0.09 11177

Overall F1 Score: 0.0921

Confusion Matrix:
[[ 2 0 0 ... 0 0 0]
[ 0 12 0 ... 0 0 0]
[ 0 0 0 ... 0 0 0]
...
[ 0 0 0 ... 0 0 0]
[ 0 0 0 ... 0 0 0]
[ 0 0 0 ... 0 0 0]]
```

Training Score: 0.1368, Test Score: 0.1083:

The training and test scores show the accuracy of the model. The values are relatively low, so the model's performance is not strong.

Precision, Recall, and F1-Score per Class:

All are generally low for most classes.

Accuracy, Macro Avg, Weighted Avg:

The overall accuracy is 0.11, indicating that the model correctly predicts the class for approximately 11% of instances.

Macro Avg is the average performance across all classes. The values are low (0.03), meaning a poor performance.

Weighted Avg considers the imbalance in class sizes. The weighted average values are also low (0.09).

Overall F1 Score: 0.0921:

The overall F1 score combines precision and recall, and its low value (0.0921) indicates a lack of balance between these two metrics.

Conclusion:

The logistic regression model seems to have poor performance based on the metrics. Let's consider other models first.

2. Decision Tree model

```
# build a decision tree model and train it
from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.metrics import precision_recall_fscore_support, accuracy_score

# selecting some features
selected_features = ['proba_elo', 'Winner_Win_Percentage', 'Winner_set_percentage']

X_train_top_n = X_train[selected_features]
X_test_top_n = X_test[selected_features]

# train Decision Tree model
dt_model_top_n = DecisionTreeClassifier(random_state=42, max_depth=3)
dt_model_top_n.fit(X_train_top_n, y_train)

# evaluate the model on training and test data
train_score_top_n = dt_model_top_n.score(X_train_top_n, y_train)
test_score_top_n = dt_model_top_n.score(X_test_top_n, y_test)

print(f"Training Score: {train_score_top_n:.4f}")
print(f"Test Score: {test_score_top_n:.4f}")

# (b) evaluate the model
y_pred_top_n = dt_model_top_n.predict(X_test_top_n)

# Calculate precision, recall, f1-score, and support
precision, recall, f1, support = precision_recall_fscore_support(y_test, y_pred_top_n,
average='weighted')

# accuracy
accuracy = accuracy_score(y_test, y_pred_top_n)

# summary
print(f"\nOverall Accuracy: {accuracy:.4f}")
print(f"Weighted Precision: {precision:.4f}")
print(f"Weighted Recall: {recall:.4f}")
print(f"Weighted F1 Score: {f1:.4f}")

# confusion matrix
conf_matrix_top_n = confusion_matrix(y_test, y_pred_top_n)
print("\nConfusion Matrix:")
print(conf_matrix_top_n)

# display the importance of all variables
feature_importance_top_n = dt_model_top_n.feature_importances_
print("\nFeature Importances:")
for feature, importance in zip(selected_features, feature_importance_top_n):
    print(f"{feature}: {importance:.4f}")

# convert integer class labels to strings
class_names = list(map(str, dt_model_top_n.classes_))

# display the tree with normalized data
plt.figure(figsize=(20, 20))
plot_tree(dt_model_top_n, feature_names=selected_features, class_names=class_names,
filled=True, rounded=True)
```

```
plt.show()
# (g) display the tree with unnormalized data
X_train_top_n_unscaled = pd.DataFrame(
    scaler.inverse_transform(X_train_top_n),
    columns=X_train_top_n.columns,
    index=X_train_top_n.index
)
plt.figure(figsize=(20, 20))
plot_tree(dt_model_top_n, feature_names=selected_features, class_names=class_names,
filled=True, rounded=True)
plt.show()
```

I kept getting an error and couldn't draw the decision tree. Here are the rest of the results:

Results:

Training Score: 0.0665, Test Score: 0.0631:

Both are again low, meaning that the decision tree model does not perform well on the dataset.

Overall Accuracy, Weighted Precision, Recall, and F1 Score:

Overall accuracy is 0.0631, meaning that the model correctly predicts the class for approximately 6.31% of instances.

Weighted precision, recall, and F1 score are extremely low, suggesting poor performance overall.

Confusion Matrix:

There are lots of zeros in the confusion matrix, which might mean that the model is consistently predicting a particular class and failing to capture the complexity of the dataset.

Feature Importances:

The decision tree has identified "Winner_Win_Percentage" as the most important feature (importance score: 1.0000), indicating that this feature alone is sufficient for making predictions in the tree. The other features, including "proba_elo" and "Winner_set_percentage," have small importance in this model.

3. Random Forest

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score,
precision_score, recall_score, f1_score

# build a Random Forest model and train
rf_model = RandomForestClassifier(random_state=42, n_estimators=100)
rf_model.fit(X_train, y_train)

# evaluate the model on training and test data
train_score_rf = rf_model.score(X_train, y_train)
test_score_rf = rf_model.score(X_test, y_test)

```

```
print(f"Training Score: {train_score_rf:.4f}")
print(f"Test Score: {test_score_rf:.4f}")

# evaluate the model
y_pred_rf = rf_model.predict(X_test)

# classification report
print("\nClassification Report:")
print(classification_report(y_test, y_pred_rf))

# confusion matrix
conf_matrix_rf = confusion_matrix(y_test, y_pred_rf)
print("\nConfusion Matrix:")
print(conf_matrix_rf)

# importance of all variables
feature_importance_rf = rf_model.feature_importances_
print("\nFeature Importances:")
for feature, importance in zip(X_train.columns, feature_importance_rf):
    print(f"{feature}: {importance:.4f}")

# additional metrics
accuracy_rf = accuracy_score(y_test, y_pred_rf)
precision_macro_rf = precision_score(y_test, y_pred_rf, average='macro')
recall_macro_rf = recall_score(y_test, y_pred_rf, average='macro')
f1_macro_rf = f1_score(y_test, y_pred_rf, average='macro')

print("\nAdditional Metrics:")
print(f"Overall Accuracy: {accuracy_rf:.4f}")
print(f"Macro Precision: {precision_macro_rf:.4f}")
print(f"Macro Recall: {recall_macro_rf:.4f}")
print(f"Macro F1 Score: {f1_macro_rf:.4f}")
```

Results:

```
Training Score: 1.0000
Test Score: 0.3074

Classification Report:
              precision    recall  f1-score   support

           2       0.12      0.18      0.14        39
           3       0.84      0.70      0.76        60
           4       0.00      0.00      0.00         1
           6       0.00      0.00      0.00         0
           8       0.00      0.00      0.00         8
           9       0.00      0.00      0.00         3
          10       0.00      0.00      0.00         0
          12       0.21      0.30      0.25        88
          13       0.50      0.50      0.50         2
          15       0.00      0.00      0.00         0
          16       0.00      0.00      0.00         1
          17       0.00      0.00      0.00         3
          18       0.10      0.25      0.14         4
          19       0.00      0.00      0.00         0
…
         892       0.00      0.00      0.00         4
         893       0.00      0.00      0.00         2
         894       0.00      0.00      0.00         2
         895       0.00      0.00      0.00         8
         896       0.00      0.00      0.00         2
         897       0.00      0.00      0.00         3
         898       0.00      0.00      0.00         2

    accuracy                           0.31     11177
   macro avg       0.10      0.10      0.09     11177
weighted avg       0.31      0.31      0.30     11177

```

```
34.
35. Confusion Matrix:
36. [[ 7  0  0 ...  0  0  0]
37.  [ 0 42  0 ...  0  0  0]
38.  [ 0  0  0 ...  0  0  0]
39.  ...
40.  [ 0  0  0 ...  0  0  0]
41.  [ 0  0  0 ...  0  0  0]
42.  [ 0  0  0 ...  0  0  0]]
43.
44. Feature Importances:
45. Best of: 0.0109
46. WRank: 0.1078
47. LRank: 0.0741
48. Wsets: 0.0149
49. Lsets: 0.0230
50. PSW: 0.0532
51. PSL: 0.0548
52. B365W: 0.0502
53. B365L: 0.0494
54. elo_winner: 0.1110
55. elo_loser: 0.0768
56. proba_elo: 0.0780
57. Winner_Win_Percentage: 0.1445
58. Loser_Win_Percentage: 0.0735
59. Surface_Carpet: 0.0038
60. Surface_Clay: 0.0081
61. Surface_Grass: 0.0066
62. Surface_Hard: 0.0093
63. Round_1st Round: 0.0141
64. Round_2nd Round: 0.0136
65. Round_3rd Round: 0.0057
66. Round_4th Round: 0.0014
67. Round_Quarterfinals: 0.0075
68. Round_Round Robin: 0.0010
69. Round_Semifinals: 0.0045
70. Round_The Final: 0.0024
71.
72. Additional Metrics:
73. Overall Accuracy: 0.3074
74. Macro Precision: 0.0985
75. Macro Recall: 0.0959
76. Macro F1 Score: 0.0927
```

Results:

Accuracy:

Training Score: 100% – The model fits the training data perfectly.

Test Score (Accuracy): 30.74% – The model performs poorly on unseen data.

Precision, Recall, F1-Score:

Varies across classes. Some have decent scores, but many are low.

Confusion Matrix:

Shows correct/incorrect predictions for each class.

Feature Importances:

Important features include "Winner_Win_Percentage," "elo_winner," and "proba_elo."

Additional Metrics:

Macro Precision, Recall, F1 Score: Low show overall poor performance across classes.

4. SVM

```
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
from sklearn.metrics import precision_score, recall_score, f1_score

# create an SVM classifier
svm_classifier = SVC(kernel='linear', C=1, random_state=42)

# train SVM model
svm_classifier.fit(X_train_final, y_train)

# predictions on the test set
y_pred_svm = svm_classifier.predict(X_test_final)

# evaluate the SVM model
accuracy_svm = accuracy_score(y_test, y_pred_svm)
precision_svm = precision_score(y_test, y_pred_svm, average='weighted')
recall_svm = recall_score(y_test, y_pred_svm, average='weighted')
f1_svm = f1_score(y_test, y_pred_svm, average='weighted')

# metrics
print("SVM Accuracy:", accuracy_svm)
print("SVM Precision:", precision_svm)
print("SVM Recall:", recall_svm)
print("SVM F1 Score:", f1_svm)

# classification Report
print("\nClassification Report:\n", classification_report(y_test, y_pred_svm))

# confusion Matrix
print("\nConfusion Matrix:\n", confusion_matrix(y_test, y_pred_svm))

```

```
SVM Accuracy: 0.13849870269303033
SVM Precision: 0.1361703705401239
SVM Recall: 0.13849870269303033
SVM F1 Score: 0.12384781726352675

Classification Report:
               precision    recall  f1-score   support

           2       0.11      0.21      0.14        39
           3       0.35      0.22      0.27        60
           4       0.00      0.00      0.00         1
           6       0.00      0.00      0.00         0
           8       0.00      0.00      0.00         8
           9       0.00      0.00      0.00         3
          10       0.00      0.00      0.00         0
          11       0.00      0.00      0.00         0
          12       0.05      0.17      0.08        88
          13       0.00      0.00      0.00         2
          15       0.00      0.00      0.00         0
          16       0.00      0.00      0.00         1
          17       0.00      0.00      0.00         3
          18       0.00      0.00      0.00         4
          20       1.00      0.07      0.12        45
```

```
24.           22       0.00      0.00      0.00        53
25.           23       0.31      0.07      0.12        56
26.           25       0.02      0.05      0.03        22
27.           26       0.10      0.12      0.11        24
28.           27       0.00      0.00      0.00         1
29.           28       0.00      0.00      0.00         1
30.           29       0.00      0.00      0.00         0
31.          …
32.
33.          878       0.00      0.00      0.00        15
34.          879       0.00      0.00      0.00         0
35.          880       0.00      0.00      0.00         5
36.          881       0.00      0.00      0.00         1
37.          882       0.00      0.00      0.00         2
38.          884       0.00      0.00      0.00         1
39.          885       0.00      0.00      0.00         0
40.          886       0.17      0.14      0.15         7
41.          887       0.00      0.00      0.00         1
42.          888       0.00      0.00      0.00         2
43.          889       0.00      0.00      0.00         0
44.          890       0.00      0.00      0.00        27
45.          891       0.00      0.00      0.00        21
46.          892       0.00      0.00      0.00         4
47.          893       0.00      0.00      0.00         2
48.          894       0.00      0.00      0.00         2
49.          895       1.00      0.25      0.40         8
50.          896       0.00      0.00      0.00         2
51.          897       0.00      0.00      0.00         3
52.          898       0.00      0.00      0.00         2
53.
54.     accuracy                           0.14     11177
55.    macro avg       0.04      0.03      0.03     11177
56. weighted avg       0.14      0.14      0.12     11177
57.
58.
59. Confusion Matrix:
60.  [[ 8  0  0 ...  0  0  0]
61.  [ 0 13  0 ...  0  0  0]
62.  [ 0  0  0 ...  0  0  0]
63.  ...
64.  [ 0  0  0 ...  0  0  0]
65.  [ 0  0  0 ...  0  0  0]
66.  [ 0  0  0 ...  0  0  0]]
67.
```

Results:

Accuracy: The model is correctly predicting the target variable for 13.85% of the instances.

Precision: About 13.62% of the instances predicted as positive by the model are actually positive.

Recall: The model is able to capture approximately 13.85% of the actual positive instances.

F1 Score: 0.1238 - model's overall performance is relatively low.

- **Describe which one(s) you selected and why?**

Considering the results of the models I've tried, it seems that the Random Forest model has performed relatively well compared to Logistic Regression, Decision Tree, and SVM. It had a higher accuracy, precision, recall, and F1-score.

Because of this, I will focus on further improving and optimizing the Random Forest model.

- **Did you use parameter optimization techniques such as Grid Search and Cross Validation?**

Yes, I tried.

```
from sklearn.model_selection import RandomizedSearchCV
from sklearn.ensemble import RandomForestClassifier

# define the parameter grid to search
param_dist = {
    'n_estimators': [50, 100, 150],
    'max_depth': [None, 10, 20],
    'min_samples_split': [2, 5],
    'min_samples_leaf': [1, 2]
}

# Random Forest classifier
rf_classifier = RandomForestClassifier(random_state=42)

# instantiate the RandomizedSearchCV object
random_search = RandomizedSearchCV(
    rf_classifier, param_distributions=param_dist,
    n_iter=10, cv=5, scoring='accuracy', n_jobs=-1, random_state=42
)

# fit the random search to the data
random_search.fit(X_train_final, y_train)

# best parameters found
print("Best Parameters:", random_search.best_params_)

# best model
best_rf_model = random_search.best_estimator_

# evaluate the best model on the test set
y_pred = best_rf_model.predict(X_test_final)

```

```
from sklearn.metrics import accuracy_score

# evaluate on Validation Set
y_val_pred = best_rf_model.predict(X_val_final)
val_accuracy = accuracy_score(y_val, y_val_pred)
print("Validation Accuracy:", val_accuracy)

# evaluate on Test Set
y_test_pred = best_rf_model.predict(X_test_final)
test_accuracy = accuracy_score(y_test, y_test_pred)
print("Test Accuracy:", test_accuracy)
```

```
12.
13. # feature importances
14. feature_importances = best_rf_model.feature_importances_
15.
16. # data frame to display feature importances
17. feature_importance_df = pd.DataFrame({'Feature': feats_combined.columns, 'Importance':
feature_importances})
18. feature_importance_df = feature_importance_df.sort_values(by='Importance', ascending=False)
19.
20. # display feature importances
21. print("\nFeature Importances:")
22. print(feature_importance_df)
23.
24. from sklearn.metrics import precision_score, recall_score, f1_score, confusion_matrix
25.
26. precision = precision_score(y_test, y_test_pred, average='weighted')  # Use 'micro',
'macro', or 'weighted'
27. recall = recall_score(y_test, y_test_pred, average='weighted')  # Use 'micro', 'macro', or
'weighted'
28. f1 = f1_score(y_test, y_test_pred, average='weighted')  # Use 'micro', 'macro', or
'weighted'
29.
30. print("Precision:", precision)
31. print("Recall:", recall)
32. print("F1-Score:", f1)
33.
34. conf_matrix = confusion_matrix(y_test, y_test_pred)
35. print("Confusion Matrix:")
36. print(conf_matrix)
37.
```

RESULTS:

```
1. Validation Accuracy: 0.3748881598568446
2. Test Accuracy: 0.3810375670840787
3.
4. Feature Importances:
5.                    Feature  Importance
6. 12  Winner_Win_Percentage    0.209635
7. 9               elo_winner    0.119007
8. 1                    WRank    0.116500
9. 11               proba_elo    0.070273
10. 10              elo_loser    0.066390
11. 13   Loser_Win_Percentage    0.063321
12. 2                   LRank    0.062512
13. 6                     PSL    0.058093
14. 5                     PSW    0.054869
15. 8                   B365L    0.044269
16. 7                   B365W    0.044068
17. 4                   Lsets    0.016683
18. 15           Surface_Clay    0.010282
19.         Round_1st Round    0.010259
20. 17           Surface_Hard    0.009828
21. 19         Round_2nd Round    0.009606
22. 3                   Wsets    0.008932
23. 0                 Best of    0.007474
24. 16          Surface_Grass    0.005767
25. 22    Round_Quarterfinals    0.004405
26. 20         Round_3rd Round    0.002849
27. 24        Round_Semifinals    0.002020
28. 14          Surface_Carpet    0.001102
29. 25         Round_The Final    0.001087
30. 21         Round_4th Round    0.000529
31. 23       Round_Round Robin    0.000241
32.
33. Precision: 0.37815545235955145
34. Recall: 0.3810375670840787
35. F1-Score: 0.36410233441443
```

```
Confusion Matrix:
[[1 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 ...
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 1 0]
 [0 0 0 ... 0 0 2]]
```

Tuned Random Forest Model:

- *Validation Accuracy: 0.3749*
- *Test Accuracy: 0.3810*
- *Precision: 0.3782*
- *Recall: 0.3810*
- *F1-Score: 0.3641*

Feature Importances (Top 5 Features):

1. *Winner_Win_Percentage*
2. *elo_winner*
3. *WRank*
4. *proba_elo*
5. *elo_loser*

Based on this comparison, it appears that the tuned model has an improvement in accuracy, precision, and recall compared to the original model. The feature importances also show a rearrangement, with Winner_Win_Percentage being the most important feature in the tuned model. This is the model with the best results so far.

- **Have you tested advanced models? Bagging, Boosting, Deep Learning...**

Why?

Yes, some.

I tried AdaBoost (Adaptive Boosting), which is a boosting algorithm. It combines multiple weak learners (typically decision trees) to create a strong learner. Each weak learner is trained sequentially, and the algorithm gives more weight to the instances that were misclassified by the previous learners.

```python
from sklearn.ensemble import AdaBoostClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score

# base Decision Tree classifier (weak learner)
base_classifier = DecisionTreeClassifier(random_state=42)

# define the parameter grid for AdaBoost
param_grid_adaboost = {
    'n_estimators': [50, 100, 200],
    'learning_rate': [0.01, 0.1, 0.5, 1]
}
```

```
14.
15. # AdaBoost classifier
16. adaboost_classifier = AdaBoostClassifier(base_classifier, random_state=42)
17.
18. # instantiate the GridSearchCV object for AdaBoost
19. grid_search_adaboost = GridSearchCV(adaboost_classifier, param_grid_adaboost, cv=5,
scoring='accuracy', n_jobs=-1)
20.
21. # fit the grid search to the data
22. grid_search_adaboost.fit(X_train_final, y_train)
23.
24. # best parameters found
25. print("Best Parameters for AdaBoost:", grid_search_adaboost.best_params_)
26.
27. # best AdaBoost model
28. best_adaboost_model = grid_search_adaboost.best_estimator_
29.
30. # evaluate on the test set
31. y_test_pred_adaboost = best_adaboost_model.predict(X_test_final)
32.
33. # calculate relevant metrics
34. accuracy_adaboost = accuracy_score(y_test, y_test_pred_adaboost)
35. precision_adaboost = precision_score(y_test, y_test_pred_adaboost, average='weighted')
36. recall_adaboost = recall_score(y_test, y_test_pred_adaboost, average='weighted')
37. f1_adaboost = f1_score(y_test, y_test_pred_adaboost, average='weighted')
38.
39. # results
40. print("\nAdaBoost Metrics:")
41. print("Test Accuracy:", accuracy_adaboost)
42. print("Precision:", precision_adaboost)
43. print("Recall:", recall_adaboost)
44. print("F1-Score:", f1_adaboost)
45.
```

AdaBoost Results:

- *Test Accuracy: 0.419200954084675*
- *Precision: 0.4280393161228633*
- *Recall: 0.419200954084675*
- *F1-Score: 0.4173056320120354*

Compared to the previous Random Forest model, this is an improvement and this is the best model in our modeling so far.

Interpretation of results

- **Have you analyzed the errors in your model?**

Yes.

```
 1. # ERROR ANALYSIS
 2. from sklearn.metrics import confusion_matrix, classification_report
 3.
 4. # confusion matrix
 5. conf_matrix_adaboost = confusion_matrix(y_test, y_test_pred_adaboost)
 6.
 7. # Display the confusion matrix
 8. print("Confusion Matrix:")
 9. print(conf_matrix_adaboost)
10.
```

```
# Generate and display the classification report
class_report_adaboost = classification_report(y_test, y_test_pred_adaboost)
print("\nClassification Report:")
print(class_report_adaboost)

Confusion Matrix:
[[ 1  0  0 ...  0  0  0]
 [ 0  0  0 ...  0  0  0]
 [ 0  0 23 ...  0  0  0]
 ...
 [ 0  0  0 ...  0  0  0]
 [ 0  0  1 ...  0  4  0]
 [ 0  0  0 ...  0  0  5]]

Classification Report:
              precision    recall  f1-score   support

           0       0.33      0.11      0.17         9
           2       0.00      0.00      0.00         7
           3       0.64      0.59      0.61        39
           5       0.00      0.00      0.00         1
           6       0.00      0.00      0.00         0
           7       0.00      0.00      0.00         1
           8       0.50      0.33      0.40         3
           9       0.19      0.24      0.21        25
          10       0.12      0.33      0.18         3
…
         521       0.00      0.00      0.00         0
         522       0.00      0.00      0.00         1
         523       0.00      0.00      0.00         0
         524       0.00      0.00      0.00         1
         525       0.27      0.25      0.26        16
         526       0.62      0.65      0.63        37
         528       0.00      0.00      0.00         0
         529       0.00      0.00      0.00         8
         531       0.00      0.00      0.00         2
         532       0.00      0.00      0.00         0
         534       0.00      0.00      0.00         1
         535       0.00      0.00      0.00         0
         536       0.00      0.00      0.00         0
         537       0.40      0.22      0.29        18
         538       0.31      0.38      0.34        13

    accuracy                           0.42      3354
   macro avg       0.19      0.19      0.18      3354
weighted avg       0.43      0.42      0.42      3354\

```

- **Did this contribute to his improvement? If yes, describe.**

I couldn't figure out a way to use the error analysis to improve the model.

- **Have you used interpretability techniques such as SHAP, LIME, Skater...**

(Grad-CAM for Deep Learning...)

No.