UVA | SCHOOL of DATA SCIENCE

“How much influence does the opening move have on the game of chess?”

Final Project

DS 5110 - Fall 2021

Big Data Systems

Dr. Adam Tashman

Group 10

- Antoine Edelman <ae4k@virginia.edu>
- Xin Huang <xh2jg@virginia.edu>
- Robert Knuuti <uqq5zz@virginia.edu>

Executive Summary

- ELO is the strongest influencer of a player winning or losing a match
- When removing ELO score that there were no strongly correlated parameters, showing that all other parameters are significant, but only minimally influence the prediction accuracy.
- No one opening appears to be stronger than another, although the King's pawn game appears to be fairly popular among the chess players.
- Overall, this shows that from our collected predictors that one cannot reliably predict the outcome of a chess match when training a model based on moves, skill level, and match types.

Code Structure and Methods

- Followed Microsoft's Team Data Science Process for collaboration
  - Provided structure for files to be located.
  - Helped in thinking about caching models or datasets in a multi data scientist way.
    - Start with the dataprep folder and run notebooks in numerical order
    - Then, pick an experiment and run the notebooks in numerical order
- Each experiment follows a similar pattern
  - Create various transforms to support modeling for features and label columns (StringIndexer, OneHotEncoder and CountVectorizer)
  - Group all transformations into a pipeline
  - Build a ParamGridBuilder for setting hyperparameter tuning
  - Build a CrossValidation model and Select the best CrossValidation model using training then test data.
  - Evaluate the performance of the model using the test data

Data Science Lifecycle

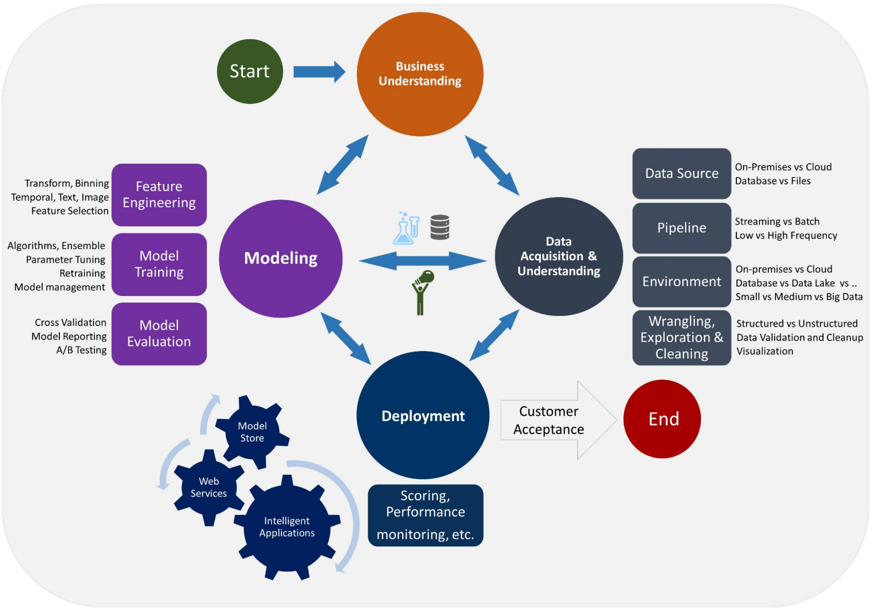
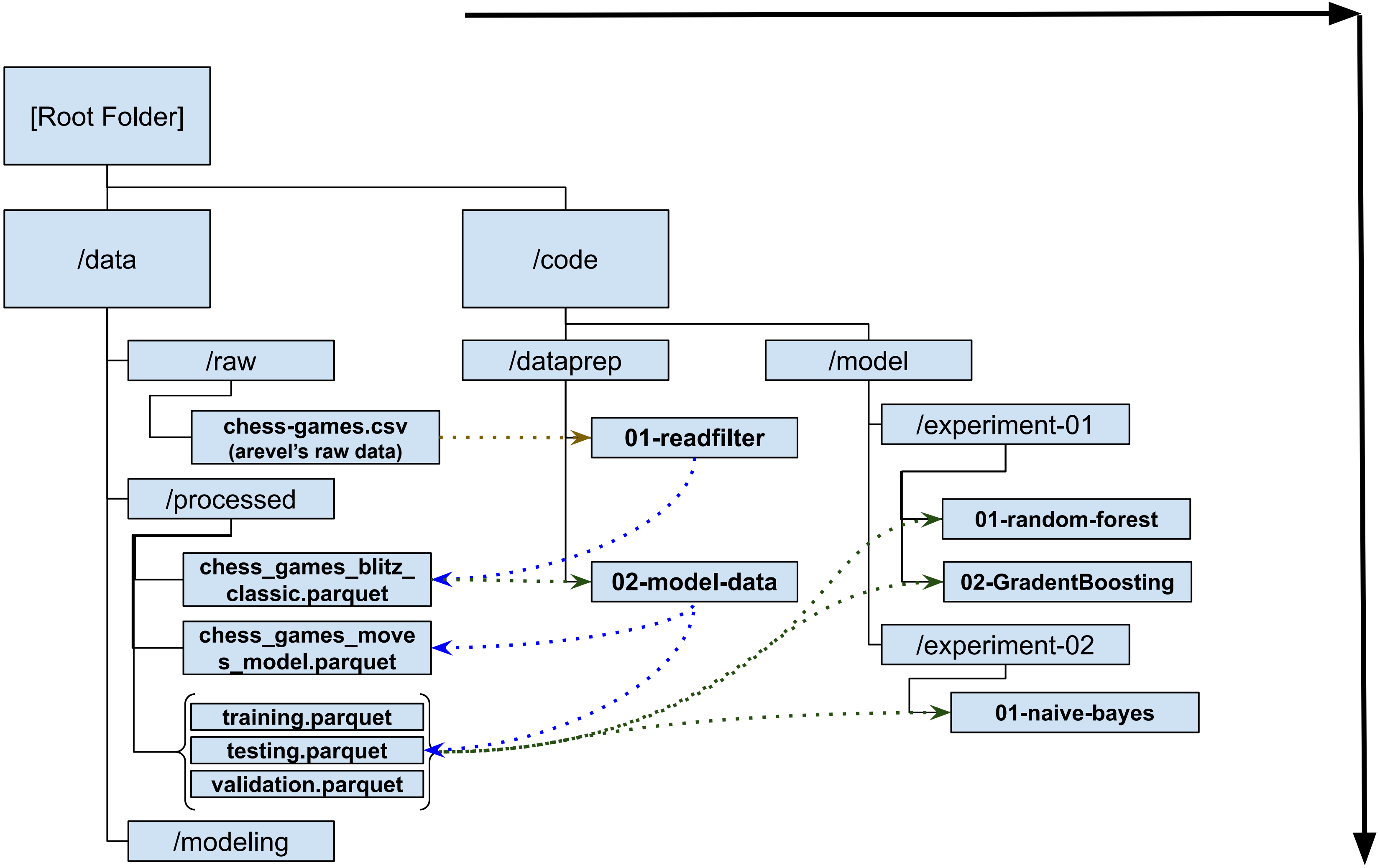


Image Credit: Microsoft (2021).


[Root Folder]
/data
/code
/raw
/dataprep
/model
chess-games.csv
(arevel's raw data)
01-readfilter
/experiment-01
/processed
01-random-forest
chess_games_blitz_
classic.parquet
02-model-data
02-GradentBoosting
chess_games_move
s_model.parquet
/experiment-02
training.parquet
01-naive-bayes
testing.parquet
validation.parquet
/modeling

Data Summary

event	white	black	result	UTCDate	UTCTime	WhiteElo	BlackElo	WhiteRatingDiff	BlackRatingDiff	ECO	Opening	TimeControl	Termination	AN
Blitz	Nippis	Misha_44	1-0	2016-01-26	18:03:38	2068	1846	11.0	-5.0	A34	English Opening: ...	300+0	Normal	1. c4 c5 2. Nc3 N...
Blitz	abracadaver	andremoniy	1-0	2016-01-26	18:03:39	1708	1399	3.0	-3.0	A40	English Defense #2	180+0	Normal	1. d4 b6 2. c4 Bb...
Classical	tewarisachin	mohamad9003	0-1	2016-01-26	18:03:39	1542	1790	-6.0	5.0	B00	Nimzowitsch Defen...	600+0	Time forfeit	1. e4 Nc6 2. d4 d...
Blitz	shamshi	kbsanswer	1-0	2016-01-26	18:03:42	1467	1679	18.0	-17.0	C21	Danish Gambit	180+1	Normal	1. e4 e5 2. d4 ex...
Blitz	yourkingismine	BurneyXM	0-1	2016-01-26	18:03:40	1249	1174	-15.0	14.0	C22	Center Game: Paul...	300+0	Normal	1. e4 e5 2. d4 ex...

Kaggle Dataset collected from LiChess
Record of all Matches in one month

Total Observations: 6,256,184 (4.38GB)
Columns: 15

https://www.kaggle.com/arevel/chess-games

Transformations and Preprocessing

Column	Datatype
Event	Categorial
White, Black	Categorial
Result	Categorial
UTCDate, UTCTime	Datetime
WhiteElo, BlackElo	Integer
WhiteRatingDiff, BlackRatingDiff	Integer
ECO	Categorical
Opening	Categorial
TimeControl	Complex (string)
Termination	Complex (string)
AN	Complex (string)

Feature Engineering

- Limit events to Classic and Blitz
- Drop abandoned games
- Expanded AN to array of moves
- Calculated difference between WhiteElo and BlackElo
- Converted number of terms to Bins to represent Complexity.
- Created indicator where White wins (1-0)

Feature
Event
ECO
Opening
moves
result_moves
complexity
EloDiff
white_result (label)
Class_Type

Final Count: 3,850,385 (1.3GB in parquet)

Resulting Dataframe

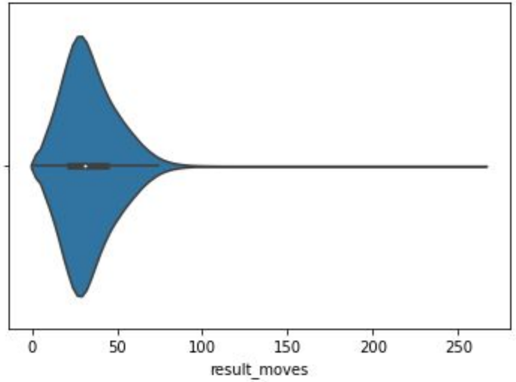
```
def movetype(x):
    import re
    moves = re.split('\d+\. ', x)[1:]
    return [x.strip() for x in moves]


udf_movetype = F.udf(lambda x: movetype(x), T.ArrayType(T.StringType()))
df_filtered = df_filtered.withColumn('moves', udf_movetype(F.col('AN')))

# Convert result column into seperate white/black win columns
white_win_udf = F.udf(lambda result: float(frac(result.split('-')[0])), T.DoubleType())
df_filtered = df_filtered.withColumn("white_games_won", white_win_udf(F.col("result")))
black_win_udf = F.udf(lambda result: float(frac(result.split('-')[1])), T.DoubleType())
df_filtered = df_filtered.withColumn("black_games_won", black_win_udf(F.col("result")))
df_filtered = df_filtered.withColumn("tie", F.col("white_games_won") == F.col("black_games_won"))

# Identify the total number of moves in a game
df_filtered = df_filtered.withColumn("result_moves", F.size(F.col("moves")))
# Categorize games based upon total move size.
df_filtered = df_filtered.withColumn("game_complexity",
                                     F.when(F.col("result_moves") ==  1, 1)\
                                      .when(F.col("result_moves") <= 10, 2)\
                                      .when(F.col("result_moves") <= 20, 3)\
                                      .when(F.col("result_moves") <= 30, 4)\
                                      .when(F.col("result_moves") <= 40, 5)\
                                      .when(F.col("result_moves") <= 50, 6)\
                                      .otherwise(7))

df_filtered = df_filtered.withColumn("EloDiff", F.col("WhiteElo") - F.col("BlackElo"))


# Collect only the first subset of moves in a game
df_filtered = df_filtered.withColumn("first_ten", F.slice(F.col("moves"), 1, 10))
df_filtered = df_filtered.withColumn("first_two", F.slice(F.col("moves"), 1, 2))

# We establish a formal win column that we'll be using as a logistical response
# We will not consider a tie as a win for white.
df_filtered = df_filtered.withColumn("white_result",
                                     F.when(F.col("white_games_won") > 0.5, "win")\
```

event	white_result	first_two	ECO	EloDiff	Opening	game_complexity
Blitz	win	[c4 c5, Nc3 Nf6]	A34	222	English Opening: ...	6
Blitz	win	[d4 b6, c4 Bb7]	A40	309	English Defense #2	5
Classical	loss	[e4 Nc6, d4 d5]	B00	-248	Nimzowitsch Defen...	7
Blitz	win	[e4 e5, d4 exd4]	C21	-212	Danish Gambit	4
Blitz	loss	[e4 e5, d4 exd4]	C22	75	Center Game: Paul...	4

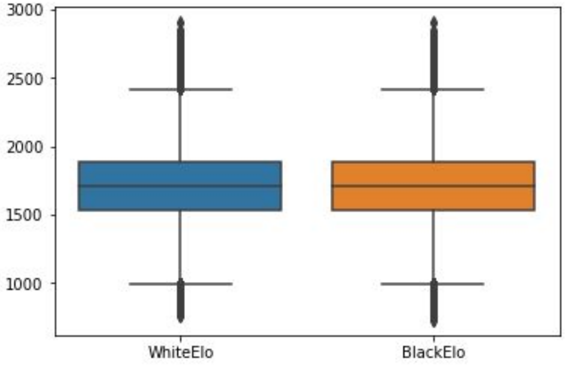
Label

Visualizing Data - EDA

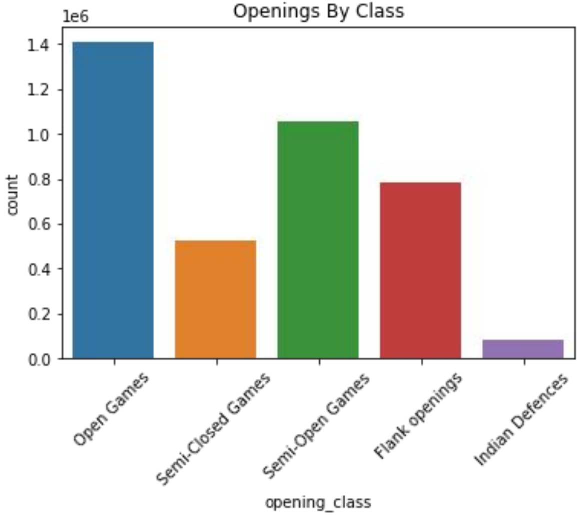
Moves per Game

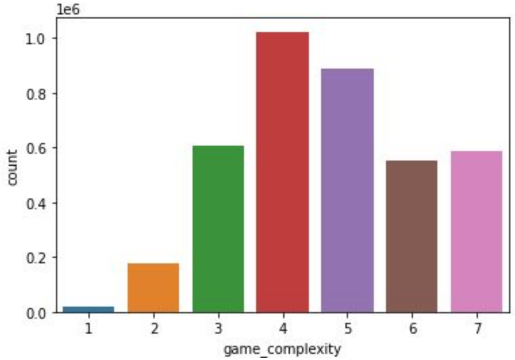


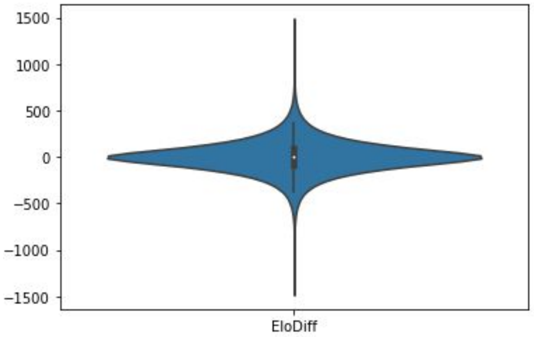
ELO Ratings



ELO Frequency by Class







Game Complexity

ELO Difference per Game

UVA SCHOOL of DATA SCIENCE

Models

Experiment -

- Naive Bayes

Experiment -

- Random Forest

Experiment -

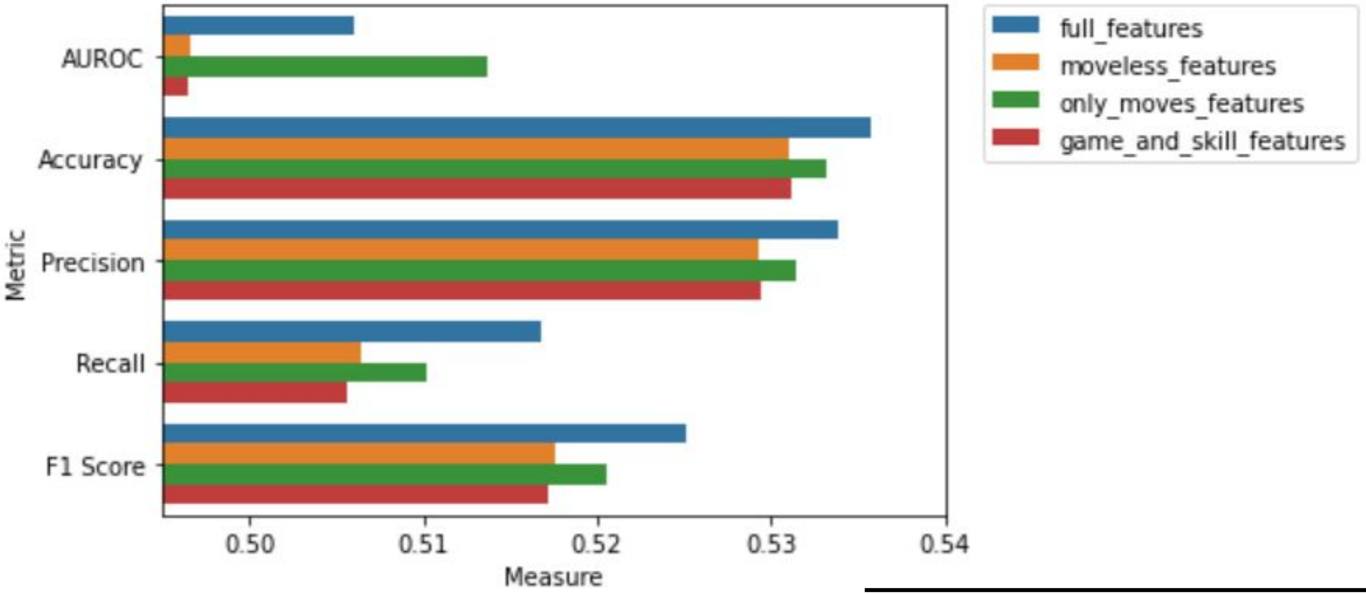
- Gradient Boosted Trees

Experiment - Naive Bayes Model

- Built four models using different predictor sets:
  - full_features ➔ ECO, EloDiff, Event, and First two actions
  - moveless_features ➔ EloDiff and Event
  - only_moves_features ➔ First two actions
  - game_and_skill_features ➔ ECO and EloDiff
- Utilized CrossValidation and BinaryClassificationEvaluator to find the best models using the AUROC as its metric.
- Only one tuning parameter: Smoothing (corrects “wiggly” lines).
  - Created a ParamGrid using a range from 0 to 1 in 0.2 steps.
  - Interestingly, the model performed best when smoothing was set to 0

UVA SCHOOL of DATA SCIENCE

Best Model Performance



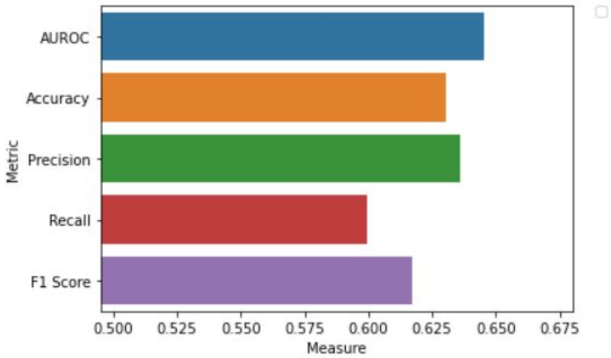
Measures	full	moveless	only_moves	game_and _skill
AUROC	0.506093	0.496643	**0.51368**	0.49655
Accuracy	**0.535799**	0.531085	0.533177	0.531128
Precision	**0.533898**	0.529331	0.531427	0.529431
Recall	**0.516721**	0.506458	0.510157	0.505604
F1	**0.525169**	0.517642	0.520575	0.517244

Experiment - Conclusion

Conclusion: Naive Bayes *does not provide statistical evidence of significant predictors* to determining a match win (only 0.01-0.015% performance difference). Additionally, the model performs slightly better than random guessing, with the best AUROC being 0.5137. *This model is ill suited for this dataset*.

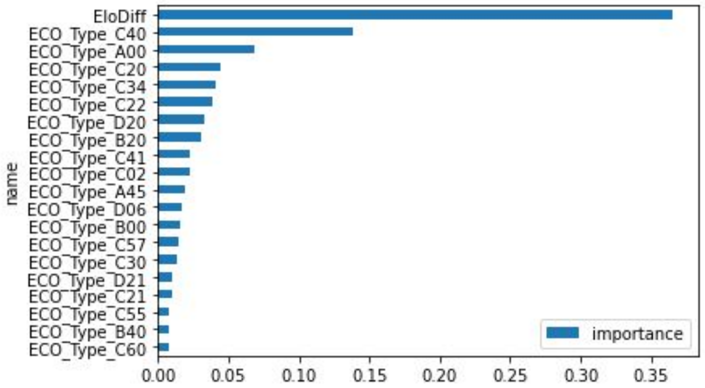
Experiment - Random Forest

The best combination of hyperparameters is numTrees = 30; maxDepth = 8; impurity = "entropy"; maxBins = 28; subsamplingRate = 1, which generates an AUC of 0.645.



Based on the optimal hyperparameter, we obtain the top 3 influential variables.

EloDiff is the most influential variable.



Experiment - Gradient Boosted Tree

- Utilized CrossValidation and BinaryClassificationEvaluator to find the best models using the AUROC as its metric. (Same as before)
- Tuning parameters
    - stepSize (1) - learning rate from (0, 1]
    - minInstancesPerNode (25) - if a split causes the number of nodes to be less then this value, it is dropped
    - subsamplingRate (1) - Fraction of the training data used for learning each decision tree, in range (0, 1].
    - maxDepth (8) - depth of the tree

Feature predictivity

GBT Predictive Ability by Feature

Variable Type	AUC	AUC filtered
EloDiff	68.77%	57.06%
ECO_Type	53.46%	51.62%
Class_Type	51.47%	50.85%
event_vector	50.14%	50.10%

GBT Predictive Ability by Features

Variable Type	AUC	AUC filtered
EloDiff, ECO_Type, & Class_Type	68.83%	57.19%
EloDiff & ECO_Type	68.82%	57.16%
EloDiff & Class_Type	68.79%	57.09%
Class_Type & ECO_Type	53.74%	51.81%

The ELO difference seems to be the driving force behind the models.

Gradient Boosted Tree

Results

GBT Final Model Metrics

Metric	Measure
AUC	68.79%
Accuracy	63.09%
Precision	63.16%
Recall	61.66%
F1 Score	62.24%

Conclusion

While Gradient Boosted Trees perform the best, they are mostly driven by the ELO difference. However, other features have a small impact on the model and further investigation into those variables is recommended.

Conclusions and Next Steps

- Our models have shown that the strongest indicator of a win is a player's rated skill.
  - This defends the ELO scoring system
- Overall our models do not perform well enough to state we have "solved chess", but we did find a model that is ~19% better than random guessing.
- Model selection is significant in the analysis of chess, as demonstrated by measured gains between each models.
- An interesting *next step* would be to map each ECO to a particular sentiment.
  - That is, are aggressive players more likely to win or lose?
  - This would need a lexicon of chess openings to support.

GitHub Repo

https://github.com/uva-ds5110-fa21-g10/semester-project

Code is located on Github for all analysis.

Data used in analysis can be obtained from kaggle at https://www.kaggle.com/arevel/chess-games