LoLWinProbabilityModel

May 11, 2025

[ ]:
```
import pandas as pd
#import lol matches for model
data = pd.read_csv('ranked_14_min_stats_zero_center.csv')
data.head(50)
```

[13]:
```
from sklearn.preprocessing import OneHotEncoder
#onehotencode champion names
encoder = OneHotEncoder(handle_unknown='ignore', sparse_output=False)
champion_onehot = encoder.fit_transform(data[['championName']])
#get champion column out
champion_onehot_columns = encoder.get_feature_names_out(['championName'])

champion_df = pd.DataFrame(champion_onehot, columns = champion_onehot_columns)

copied_final_df_encoded_champions = pd.concat([data.reset_index(drop = True),␣
↪champion_df], axis = 1)
copied_final_df_encoded_champions.drop('championName', axis = 1, inplace = True)
```

[ ]:
```
copied_final_df_encoded_champions.columns
```

[81]:
```
#get rid of unneeded columns
copied_final_df_encoded_champions_dropped_columns =␣
↪copied_final_df_encoded_champions.drop(columns =['goldPerMinute',
                                                                                   ␣
↪                      'minionsKilled',
                                                                                   ␣
↪                      'jungleMinionsKilled',
                                                                                   ␣
↪                      'csPerMinute',
                                                                                   ␣
↪                      'participantId',
                                                                                   ␣
↪                      'teamPosition'],axis = 'columns')
```

[23]:
```
#rename 100/200 to Blue/Red
```

```
copied_final_df_encoded_champions_dropped_columns['teamId'] =␣
↪copied_final_df_encoded_champions_dropped_columns['teamId'].replace({100:
↪'Blue',
                                                                              ␣
↪                                                          200:'Red'})
```

[25]:
```
#rename horde kills to grubs
copied_final_df_encoded_champions_dropped_columns =␣
↪copied_final_df_encoded_champions_dropped_columns.rename(columns =␣
↪{'teamHordeKills_14': 'teamGrubKills'})
```

[27]:
```
#aggregate stats for team based modeling
champion_names = [col for col in␣
↪copied_final_df_encoded_champions_dropped_columns if col.
↪startswith('championName_')]

stat_agg = {'firstBloodKill': 'max',
            'totalGold' : 'sum',
            'xp': 'sum',
            'level': 'mean',
            'platesTaken_14' : 'sum',
            'towersKilled_14' : 'sum',
            'teamGrubKills': 'max',
            'teamDragonKills_14': 'max',
            'wards_placed': 'sum',
            'roleGoldDiff': 'sum',
            'roleXpDiff': 'sum',
            'roleCsDiff' : 'sum',
            'roleKillDiff' : 'sum',
            'roleDeathsDiff' : 'sum',
            'roleAssistsDiff' : 'sum',
            'roleVisionDiff' : 'sum',
            'win': 'max'}

for col in champion_names:
    stat_agg[col] = 'sum'


copied_final_df_encoded_champions_dropped_columns_grouped_team =␣
↪(copied_final_df_encoded_champions_dropped_columns
                                                                           .
↪groupby(['match_id','teamId'])
                                                                           .agg(stat_agg)
                                                                           .
↪reset_index())
```

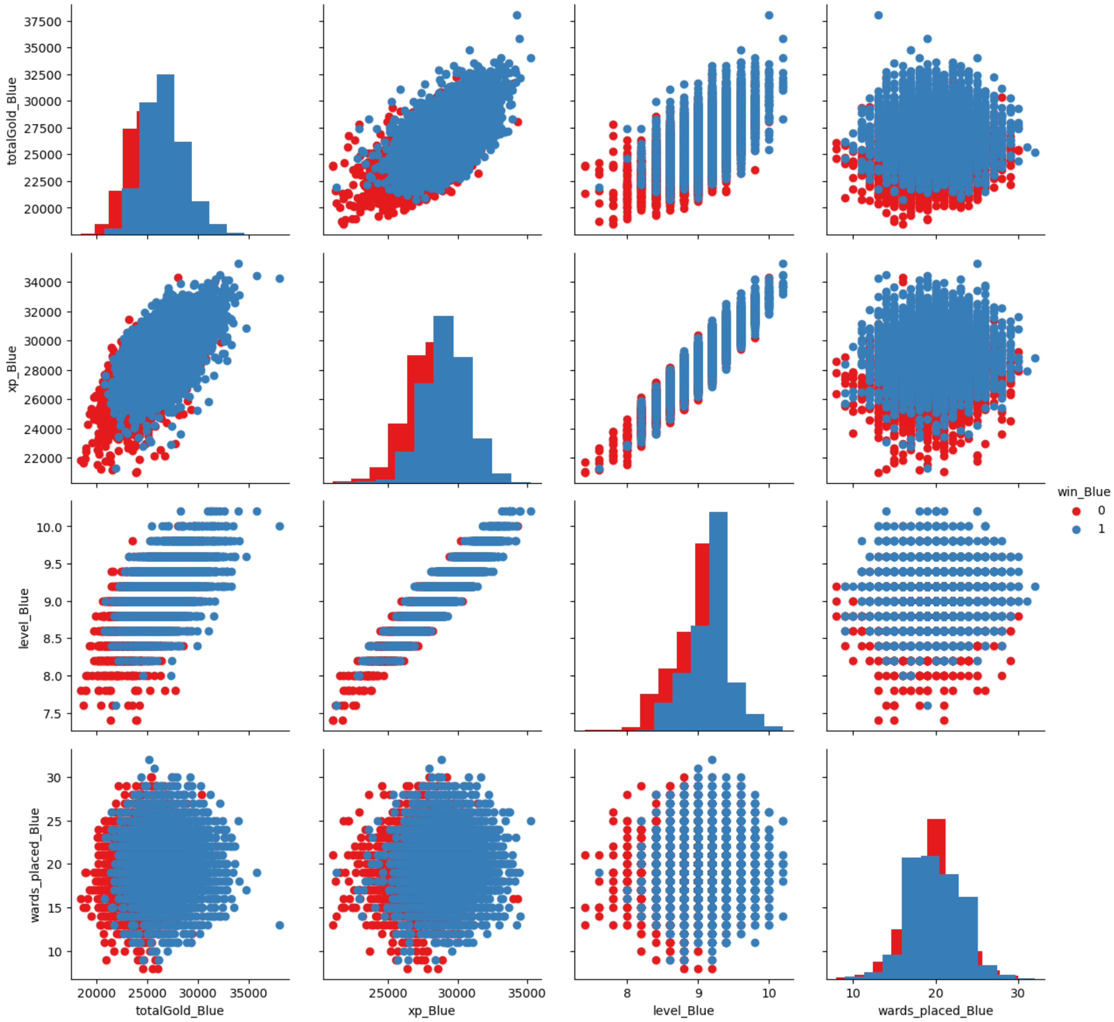
[ ]:
```python
#pivot so we can get one row per match comparing both teams side-by-side as␣
↪blue vs red
pivoted = copied_final_df_encoded_champions_dropped_columns_grouped_team.
↪pivot(index='match_id', columns='teamId')

#flatten column names
pivoted.columns = ['{}_{}'.format(stat, team) for stat, team in pivoted.columns]
pivoted.reset_index(inplace=True)
pivoted
```

[33]:
```python
#rename role based headers since we are now looking at blue vs red team metrics
pivoted.columns = [col.replace("role", "") if col.startswith("role") else col
                   for col in pivoted.columns]
```

[35]:
```python
#remove win_red since we are predicting blue win rate
pivoted = pivoted.drop(columns =['win_Red'],axis = 'columns')
```

[37]:
```python
#take a look at scatterplot matrix to visualize relationships between some of␣
↪the features
import seaborn as sns
import matplotlib.pyplot as plt
pairgrid = sns.PairGrid(pivoted, vars =␣
↪['totalGold_Blue','xp_Blue','level_Blue','wards_placed_Blue'], hue =␣
↪'win_Blue', height = 3, palette = 'Set1')
pairgrid.map_diag(plt.hist)
pairgrid.map_offdiag(plt.scatter)
pairgrid.add_legend()
```

[37]: <seaborn.axisgrid.PairGrid at 0x156db2b10>



[39]:

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import cross_val_score
import numpy as np

#split data into predictors and response
X = pivoted.drop(columns=['match_id', 'win_Blue'])
y = pivoted['win_Blue']

#initialize the model
model = RandomForestClassifier(n_estimators=100, random_state=42)

#5-fold cross-validation with ROC AUC scoring
scores = cross_val_score(model, X, y, cv=5, scoring='roc_auc')

#Output the results
```

```
print("ROC AUC scores from each fold:", scores)
print("Mean ROC AUC:", np.mean(scores).round(4))
```

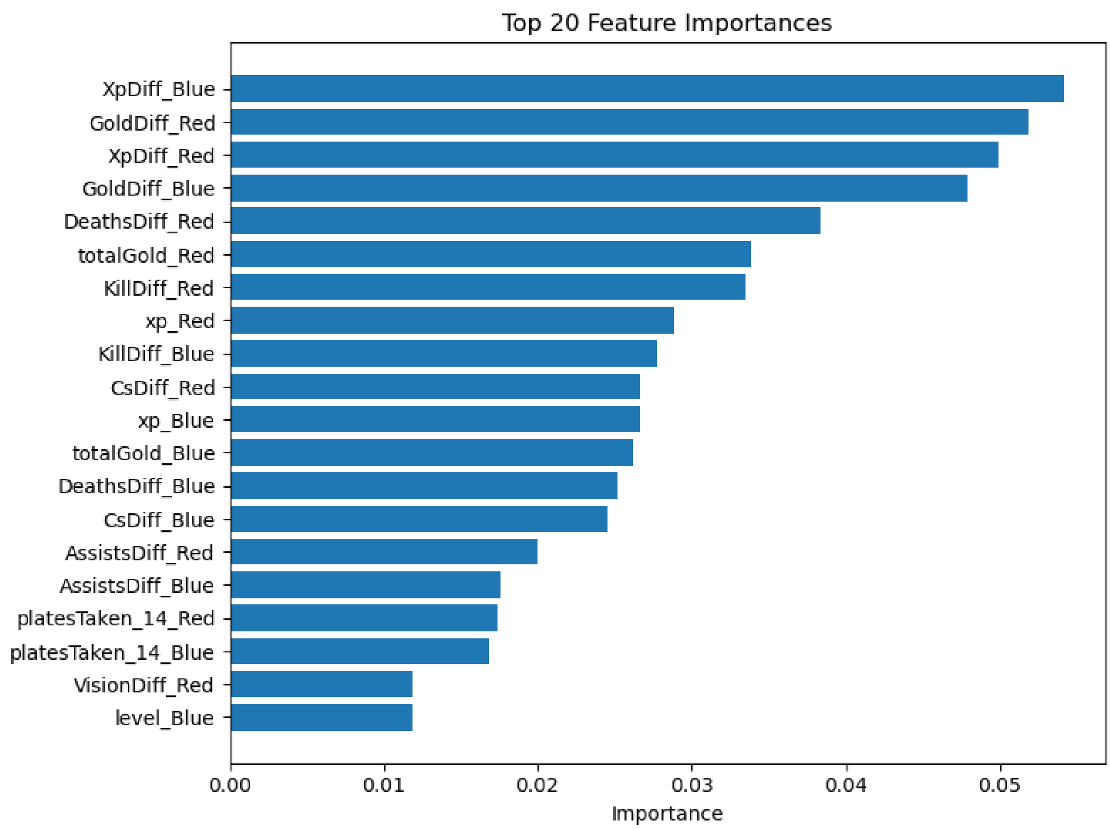
```
ROC AUC scores from each fold: [0.84582887 0.85099096 0.84066587 0.85403146
0.8376285 ]
Mean ROC AUC: 0.8458
```

[41]:
```
#fit the model
model.fit(X, y)

#get feature importances
importances = model.feature_importances_
feature_names = X.columns

#get indices of top 20 features, sorted descending
top_indices = importances.argsort()[-20:][::-1]

#print top 20 features from most to least important
for i in top_indices:
    print(f"{feature_names[i]}: {importances[i]:.4f}")
```

```
XpDiff_Blue: 0.0542
GoldDiff_Red: 0.0518
XpDiff_Red: 0.0500
GoldDiff_Blue: 0.0479
DeathsDiff_Red: 0.0384
totalGold_Red: 0.0339
KillDiff_Red: 0.0335
xp_Red: 0.0289
KillDiff_Blue: 0.0278
CsDiff_Red: 0.0266
xp_Blue: 0.0266
totalGold_Blue: 0.0262
DeathsDiff_Blue: 0.0252
CsDiff_Blue: 0.0245
AssistsDiff_Red: 0.0200
AssistsDiff_Blue: 0.0176
platesTaken_14_Red: 0.0174
platesTaken_14_Blue: 0.0168
VisionDiff_Red: 0.0119
level_Blue: 0.0118
```

[79]:
```
#feature importance
import numpy as np

importances = model.feature_importances_
indices = np.argsort(importances)[-20:]  # top 20
plt.figure(figsize=(8, 6))
```

```
plt.barh(np.array(X.columns)[indices], importances[indices])
plt.title("Top 20 Feature Importances")
plt.xlabel("Importance")
plt.tight_layout()
plt.show()
```



[43]:
```
from sklearn.model_selection import GridSearchCV

#number of trees in forest, depth of each tree, and min samples required to␣
↪split the node
param_grid = {'n_estimators': [100, 200],'max_depth': [None, 10,␣
↪20],'min_samples_split': [2, 5]}

#base model
rf = RandomForestClassifier(random_state=42)

#grid search with 5-fold cross-validation
grid_search = GridSearchCV(
    estimator=rf,
    param_grid=param_grid,
```

```
    cv=5,
    scoring='roc_auc',
    n_jobs=-1,
    verbose=1)

#fit the grid search to the data
grid_search.fit(X, y)
```

```
Fitting 5 folds for each of 12 candidates, totalling 60 fits
```

[43]:
```
GridSearchCV(cv=5, estimator=RandomForestClassifier(random_state=42), n_jobs=-1,
             param_grid={'max_depth': [None, 10, 20],
                         'min_samples_split': [2, 5],
                         'n_estimators': [100, 200]},
             scoring='roc_auc', verbose=1)
```

[45]:
```
#print the best score and parameters
print("Best AUC Score:", grid_search.best_score_)
print("Best Parameters:", grid_search.best_params_)
```

```
Best AUC Score: 0.8479580156940711
Best Parameters: {'max_depth': 20, 'min_samples_split': 2, 'n_estimators': 200}
```

[47]:
```
from sklearn.metrics import accuracy_score, roc_auc_score, classification_report
from sklearn.model_selection import train_test_split

#split data
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42, stratify=y)
#fit on x train
grid_search.fit(X_train, y_train)

#get the best model from grid search
best_model = grid_search.best_estimator_

#predict on the test set
#for ROC AUC
y_pred = best_model.predict(X_test)
y_proba = best_model.predict_proba(X_test)[:, 1]
```

```
Fitting 5 folds for each of 12 candidates, totalling 60 fits
```

[49]:
```
#print results of the model
print("Accuracy:", accuracy_score(y_test, y_pred))
print("AUC Score:", roc_auc_score(y_test, y_proba))
print("\nClassification Report:\n", classification_report(y_test, y_pred))
```

```
Accuracy: 0.7538095238095238
AUC Score: 0.8458629988378008
```

```
Classification Report:
              precision    recall  f1-score   support

           0       0.74      0.73      0.74       989
           1       0.77      0.77      0.77      1111

    accuracy                           0.75      2100
   macro avg       0.75      0.75      0.75      2100
weighted avg       0.75      0.75      0.75      2100
```

[ ]:
```
#auc score of 85% means my model is learning from the data and the model is␣
 ↪ranking winners over loses 85% of the time
#we can see that blue(1) performanc is slightly higher than red(0) which is in␣
 ↪line with expected results
#this is mainly due a few reasons. First camera position giving a slight␣
 ↪competitive advantage, first pick in draft
#allows enemy team to be forced to ban the best champion in the current meta as␣
 ↪blue can pick it up if its available
#and better map and camera access to baron
#seeing the model pick up the slight edge that blue has means the model is␣
 ↪learning properly from the data
```

[ ]:
```
#now trying XGBoost to compare model performance
!pip install xgboost
```

[52]:
```
from xgboost import XGBClassifier
from sklearn.metrics import accuracy_score, roc_auc_score, classification_report
from sklearn.model_selection import train_test_split

#train/test split 80/20
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42, stratify=y)
#initialize model
xgb_model = XGBClassifier(
    n_estimators=300,
    max_depth=5,
    learning_rate=0.05,
    subsample=0.8,
    colsample_bytree=0.8,
    eval_metric='logloss',
    random_state=42)

#train the model
xgb_model.fit(X_train, y_train)
```

[52]:
```
XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=0.8, device=None, early_stopping_rounds=None,
              enable_categorical=False, eval_metric='logloss',
              feature_types=None, feature_weights=None, gamma=None,
              grow_policy=None, importance_type=None,
              interaction_constraints=None, learning_rate=0.05, max_bin=None,
              max_cat_threshold=None, max_cat_to_onehot=None,
              max_delta_step=None, max_depth=5, max_leaves=None,
              min_child_weight=None, missing=nan, monotone_constraints=None,
              multi_strategy=None, n_estimators=300, n_jobs=None,
              num_parallel_tree=None, …)
```

[55]:
```python
#predict
y_pred = xgb_model.predict(X_test)
y_proba = xgb_model.predict_proba(X_test)[:, 1]

#print model metrics
print("Accuracy:", accuracy_score(y_test, y_pred))
print("AUC Score:", roc_auc_score(y_test, y_proba))
print("\nClassification Report:\n", classification_report(y_test, y_pred))
```

```
Accuracy: 0.7690476190476191
AUC Score: 0.8570413158606054

Classification Report:
               precision    recall  f1-score   support

           0       0.76      0.74      0.75       989
           1       0.78      0.79      0.78      1111

    accuracy                           0.77      2100
   macro avg       0.77      0.77      0.77      2100
weighted avg       0.77      0.77      0.77      2100
```

[ ]:
```python
#we can see a slightly higher performance on the model for the blue team
#higher accuracy, AUC score, precision and recall
#xgboost model is learning from mistakes in earlier trees
#regularization from the XGBoost model is helping prevent overfitting
#slightly higher metrics for blue(1) also shows model is learning and seeing␣
↪blue win rate being
#higher compared to red
```

[59]:
```python
#XGBoost model with GridSearchCV
from xgboost import XGBClassifier
from sklearn.model_selection import GridSearchCV
```

```
param_grid = {
    'n_estimators': [200, 300],
    'max_depth': [4, 5, 6],
    'learning_rate': [0.03, 0.05, 0.1],
    'subsample': [0.8, 1.0],
    'colsample_bytree': [0.8, 1.0]}


#base model
xgb_base = XGBClassifier(
    eval_metric='logloss',
    random_state=42)

#grid search
xgb_grid = GridSearchCV(
    estimator=xgb_base,
    param_grid=param_grid,
    scoring='roc_auc',
    cv=5,
    verbose=1,
    n_jobs=-1)

#fit on training data only
xgb_grid.fit(X_train, y_train)
```

```
Fitting 5 folds for each of 72 candidates, totalling 360 fits
```

[59]:

```
GridSearchCV(cv=5,
             estimator=XGBClassifier(base_score=None, booster=None,
                                     callbacks=None, colsample_bylevel=None,
                                     colsample_bynode=None,
                                     colsample_bytree=None, device=None,
                                     early_stopping_rounds=None,
                                     enable_categorical=False,
                                     eval_metric='logloss', feature_types=None,
                                     feature_weights=None, gamma=None,
                                     grow_policy=None, importance_type=None,
                                     interaction_constraint…
                                     max_delta_step=None, max_depth=None,
                                     max_leaves=None, min_child_weight=None,
                                     missing=nan, monotone_constraints=None,
                                     multi_strategy=None, n_estimators=None,
                                     n_jobs=None, num_parallel_tree=None, …),
             n_jobs=-1,
             param_grid={'colsample_bytree': [0.8, 1.0],
                         'learning_rate': [0.03, 0.05, 0.1],
                         'max_depth': [4, 5, 6], 'n_estimators': [200, 300],
```

```
                        'subsample': [0.8, 1.0]},
             scoring='roc_auc', verbose=1)
```

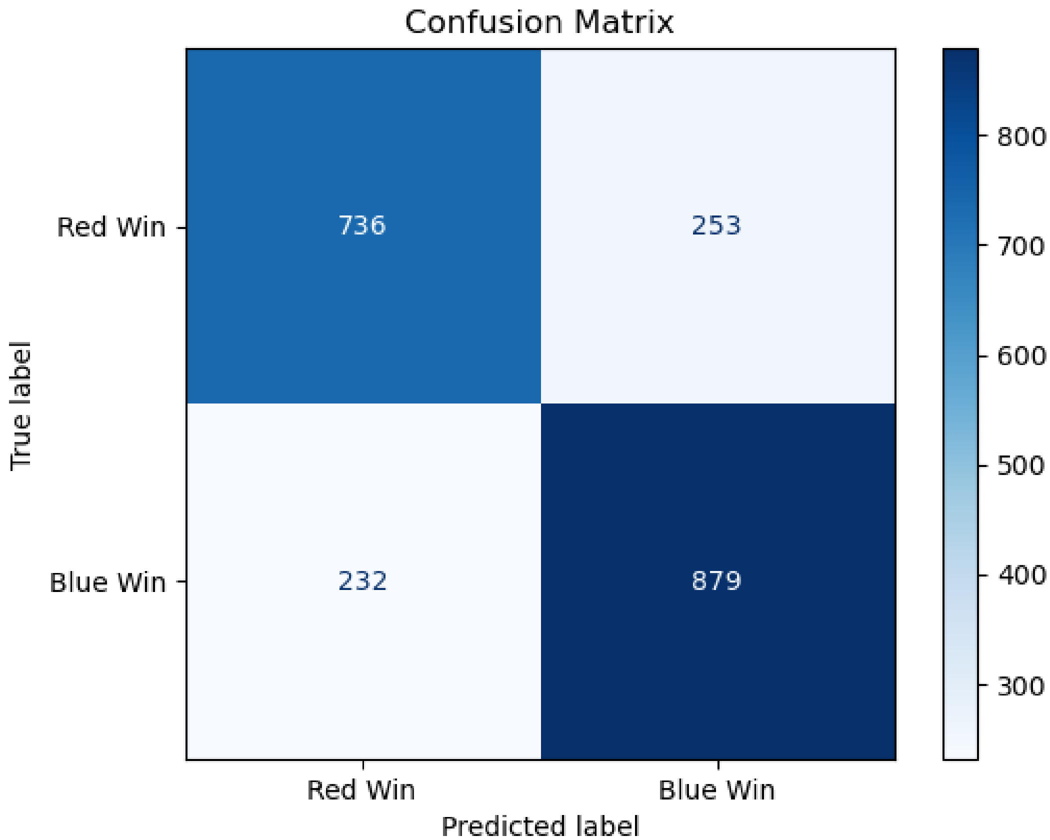
[61]:
```
#print best AUC score and best parameters
print("Best AUC Score:", xgb_grid.best_score_)
print("Best Parameters:", xgb_grid.best_params_)
```

```
Best AUC Score: 0.8529987720790245
Best Parameters: {'colsample_bytree': 0.8, 'learning_rate': 0.05, 'max_depth':
4, 'n_estimators': 200, 'subsample': 0.8}
```

[67]:
```
#final model
XGBClassifier(
    n_estimators=200,
    max_depth=4,
    learning_rate=0.05,
    subsample=0.8,
    colsample_bytree=0.8,
    eval_metric='logloss',
    random_state=42)
```

[67]:
```
XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=0.8, device=None, early_stopping_rounds=None,
              enable_categorical=False, eval_metric='logloss',
              feature_types=None, feature_weights=None, gamma=None,
              grow_policy=None, importance_type=None,
              interaction_constraints=None, learning_rate=0.05, max_bin=None,
              max_cat_threshold=None, max_cat_to_onehot=None,
              max_delta_step=None, max_depth=4, max_leaves=None,
              min_child_weight=None, missing=nan, monotone_constraints=None,
              multi_strategy=None, n_estimators=200, n_jobs=None,
              num_parallel_tree=None, …)
```

[77]:
```
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
#confusion matrix to show how well model performed, TN and TPs vs FN and FPs
cm = confusion_matrix(y_test, y_pred)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=['Red Win',␣
 ↪'Blue Win'])
disp.plot(cmap='Blues')
plt.title("Confusion Matrix")
plt.show()
```



Confusion Matrix
Red Win
Blue Win
True label
736
253
232
879
Red Win
Blue Win
Predicted label
800
700
600
500
400
300