Interceptions

October 10, 2025

[1]:
```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, accuracy_score,␣
 ↪confusion_matrix
from sklearn.ensemble import RandomForestClassifier
from xgboost import XGBClassifier
```

1 Probability QB Will Throw a Touchdown

[2]:
```
# Train & evaluate logistic regression model

player_stats_with_defense = pd.read_csv('data/PlayerStats.csv',␣
 ↪low_memory=False)

# Step 1: Remove rows where all relevant stats are zeros
stats_columns = ['attempts', 'completions', 'passing_yards', 'passing_tds',␣
 ↪'sacks']
player_stats_with_defense_cleaned = player_stats_with_defense[
    ~((player_stats_with_defense[stats_columns] == 0).all(axis=1))
]

# Step 2: Filter data to only include seasons 2020, 2021, 2022, 2023
seasons_to_keep = [2020, 2021, 2022, 2023]
player_stats_with_defense_cleaned = player_stats_with_defense_cleaned[
    player_stats_with_defense_cleaned['season'].isin(seasons_to_keep)
]

# Step 3: Filter to only keep rows where position is QB
player_stats_with_defense_cleaned = player_stats_with_defense_cleaned[
    player_stats_with_defense_cleaned['position'] == 'QB'
]

# Basic feature selection
features = ['attempts', 'completions', 'passing_yards', 'passing_tds', 'sacks',␣
 ↪'opponent_defense_strength']
```

```
# Target variable: whether the QB threw an interception
player_stats_with_defense_cleaned['threw_interception'] =␣
 ↪player_stats_with_defense_cleaned['interceptions'].apply(lambda x: 1 if x >␣
 ↪0 else 0)

# Drop rows with missing values
player_stats_with_defense_cleaned = player_stats_with_defense_cleaned.
 ↪dropna(subset=features + ['threw_interception'])

# Split the data
X = player_stats_with_defense_cleaned[features]
y = player_stats_with_defense_cleaned['threw_interception']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)

# Train the logistic regression model
model = LogisticRegression(max_iter=1000)
model.fit(X_train, y_train)

# Make predictions
y_pred = model.predict(X_test)

# Evaluation
accuracy = accuracy_score(y_test, y_pred)
report = classification_report(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)

# Print the results
print(f"Accuracy: {accuracy:.4f}")
print("\nClassification Report:\n", report)
print("\nConfusion Matrix:\n", conf_matrix)
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
/var/folders/wx/m1tmq8gx4_v321trybkxcg9m0000gn/T/ipykernel_26896/894251445.py i␣
 ↪?()
     25 # Target variable: whether the QB threw an interception
     26 player_stats_with_defense_cleaned['threw_interception'] =␣
 ↪player_stats_with_defense_cleaned['interceptions'].apply(lambda x: 1 if x > 0␣
 ↪else 0)
     27
     28 # Drop rows with missing values
---> 29 player_stats_with_defense_cleaned = player_stats_with_defense_cleaned.
 ↪dropna(subset=features + ['threw_interception'])
     30
```

```
     31 # Split the data
     32 X = player_stats_with_defense_cleaned[features]

~/.pyenv/versions/3.12.0/lib/python3.12/site-packages/pandas/core/frame.py in ?
 ↪(self, axis, how, thresh, subset, inplace, ignore_index)
   6666             ax = self._get_axis(agg_axis)
   6667             indices = ax.get_indexer_for(subset)
   6668             check = indices == -1
   6669             if check.any():
-> 6670                 raise KeyError(np.array(subset)[check].tolist())
   6671             agg_obj = self.take(indices, axis=agg_axis)
   6672
   6673         if thresh is not lib.no_default:

KeyError: ['opponent_defense_strength']
```

[3]:
```python
# Compare Random Forest and XGBoost

# Random Forest Model
rf_model = RandomForestClassifier(n_estimators=100, random_state=42)
rf_model.fit(X_train, y_train)
y_pred_rf = rf_model.predict(X_test)

# XGBoost Model
xgb_model = XGBClassifier(eval_metric='logloss', random_state=42)
xgb_model.fit(X_train, y_train)
y_pred_xgb = xgb_model.predict(X_test)

# Evaluation for Random Forest
accuracy_rf = accuracy_score(y_test, y_pred_rf)
report_rf = classification_report(y_test, y_pred_rf)
conf_matrix_rf = confusion_matrix(y_test, y_pred_rf)

print("Random Forest Model")
print(f"Accuracy: {accuracy_rf:.4f}")
print("\nClassification Report:\n", report_rf)
print("\nConfusion Matrix:\n", conf_matrix_rf)

# Evaluation for XGBoost
accuracy_xgb = accuracy_score(y_test, y_pred_xgb)
report_xgb = classification_report(y_test, y_pred_xgb)
conf_matrix_xgb = confusion_matrix(y_test, y_pred_xgb)

print("\nXGBoost Model")
print(f"Accuracy: {accuracy_xgb:.4f}")
print("\nClassification Report:\n", report_xgb)
```

```
print("\nConfusion Matrix:\n", conf_matrix_xgb)
```

```
Random Forest Model
Accuracy: 0.6130

Classification Report:
              precision    recall  f1-score   support

           0       0.62      0.69      0.66       277
           1       0.60      0.52      0.56       245

    accuracy                           0.61       522
   macro avg       0.61      0.61      0.61       522
weighted avg       0.61      0.61      0.61       522


Confusion Matrix:
 [[192  85]
 [117 128]]

XGBoost Model
Accuracy: 0.5747

Classification Report:
              precision    recall  f1-score   support

           0       0.59      0.64      0.62       277
           1       0.55      0.50      0.52       245

    accuracy                           0.57       522
   macro avg       0.57      0.57      0.57       522
weighted avg       0.57      0.57      0.57       522


Confusion Matrix:
 [[178  99]
 [123 122]]
```

[7]:
```
# Here's the full code that includes both the shifting of data and the use of␣
 ↪historical averages
# to estimate values for upcoming games:

average_stats = player_stats_with_defense_cleaned.
 ↪groupby('player_display_name').agg({
    'attempts': 'mean',
    'completions': 'mean',
    'passing_yards': 'mean',
```

```
    'passing_tds': 'mean',
    'sacks': 'mean',
    'opponent_defense_strength': 'mean'
}).reset_index()

# Step 2: Shift data by one game to predict the next game based on the previous␣
 ↪one
shifted_stats = player_stats_with_defense_cleaned.
 ↪groupby('player_display_name')[features].shift(1)

# Combine shifted stats with historical averages
# If shifted stats are available, they take precedence; otherwise, use␣
 ↪historical averages
predictions_data = shifted_stats.combine_first(average_stats.
 ↪set_index('player_display_name')).reset_index()

# Fill NaNs in numeric columns only
predictions_data[numeric_features] = predictions_data[numeric_features].
 ↪fillna(predictions_data[numeric_features].mean())

# Ensure the predictions_data has the same structure as your training data
predictions_data = predictions_data[features]

# Step 3: Predict probabilities for upcoming games
interception_probabilities = model.predict_proba(predictions_data)

# Print the probabilities
interception_probabilities
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[7], line 21
     18 predictions_data = shifted_stats.combine_first(average_stats.
 ↪set_index('player_display_name')).reset_index()
     20 # Fill NaNs in numeric columns only
---> 21 predictions_data[numeric_features] = predictions_data[numeric_features]
 ↪fillna(predictions_data[numeric_features].mean())
     23 # Ensure the predictions_data has the same structure as your training␣
 ↪data
     24 predictions_data = predictions_data[features]

NameError: name 'numeric_features' is not defined
```

[ ]:

[ ]:

[ ]: