PLAYER DETECTION IN FOOTBALL VIDEOS USING OPENCV

GOAL PREDICTION FROM SOCCER DATA USING MACHINE LEARNING

Introduction

Soccer player detection is a computer vision task that involves automatically identifying and localizing soccer players within images or videos. The objective of player detection is to enable automated analysis of soccer games, including tracking the movements of individual players, analysing team tactics, and providing performance metrics for players and teams.

Player detection has a range of practical applications in the field of sports analytics, including team performance analysis, player evaluation and scouting, and live game commentary.

Predicting goals in soccer has been a popular research topic for a long time, and machine learning is a powerful tool for analysing the spatial-temporal data in the game.

By accurately predicting the likelihood of a goal being scored, machine learning algorithms can provide valuable insights to coaches, players, and fans, which can improve team performance and enhance the viewing experience.

Methodology – Player Detection Project

The Project essentially revolves around the detected of players from a football video clip, and recognizing which country the players are from based on the colour of their jersey. In this case, we observe two teams and the football, and display a recognition of either 'France' (blue) or 'Belgium' (red/orange) or the football (white).

First, we import all required libraries and read the video from a saved title:

```
import cv2
import os
import numpy as np

#Reading the video
vidcap = cv2.VideoCapture('Vid4.mp4')
success,image = vidcap.read()
count = 0
success = True
idx = 0
```

Next, to read the video, we need to convert it frame by frame into HSV format (Hue, Saturation and Value). This is done so we can detect any specified colour background ranging from specific colour codes. We thus define colour ranges based on BGR and HSV colour codes

```
while success:

    #converting into hsv image
    hsv = cv2.cvtColor(image,cv2.COLOR_BGR2HSV)
    #green range
    lower_green = np.array([40,40, 40])
    upper_green = np.array([70, 255, 255])
    #blue range
    lower_blue = np.array([110,50,50])
    upper_blue = np.array([130,255,255])
    #Red range
    lower_red = np.array([0,31,255])
    upper_red = np.array([176,255,255])
    #White range
    lower_white = np.array([0,0,0])
    upper_white = np.array([0,0,255])
```

Our first approach is to isolate the green colour of the field, which is done by developing a mask of the green colour to detect the field:

```
    #Define a mask ranging from lower to uppper
    mask = cv2.inRange(hsv, lower_green, upper_green)
    res = cv2.bitwise_and(image, image, mask=mask)
    #convert to hsv to gray
    res_bgr = cv2.cvtColor(res,cv2.COLOR_HSV2BGR)
    res_gray = cv2.cvtColor(res,cv2.COLOR_BGR2GRAY)
```

What this does is isolate the green colour in the image and blackens everything else.

Next, to reduce false detections, we fill out all noise in the crowd by using morphological closing operations:

```
  #Defining a kernel to do morphological operation in threshold image to
    #get better output.
    kernel = np.ones((13,13),np.uint8)
    thresh = cv2.threshold(res_gray,127,255,cv2.THRESH_BINARY_INV |
cv2.THRESH_OTSU)[1]
    thresh = cv2.morphologyEx(thresh, cv2.MORPH_CLOSE, kernel)
```

Next, we find contours in the frames (player outlines) and then detect the jersey colour and assign a name (country) to the detection:

```
   #find contours in threshold image
    contours,hierarchy =
cv2.findContours(thresh,cv2.RETR_TREE,cv2.CHAIN_APPROX_SIMPLE)

    prev = 0
    font = cv2.FONT_HERSHEY_SIMPLEX

    for c in contours:
        x,y,w,h = cv2.boundingRect(c)

        #Detect players
        if(h>=(1.5)*w):
            if(w>15 and h>= 15):
                idx = idx+1
                player_img = image[y:y+h,x:x+w]
                player_hsv = cv2.cvtColor(player_img,cv2.COLOR_BGR2HSV)
                #If player has blue jersy
                mask1 = cv2.inRange(player_hsv, lower_blue, upper_blue)
                res1 = cv2.bitwise_and(player_img, player_img, mask=mask1)
                res1 = cv2.cvtColor(res1,cv2.COLOR_HSV2BGR)
                res1 = cv2.cvtColor(res1,cv2.COLOR_BGR2GRAY)
                nzCount = cv2.countNonZero(res1)
```

```
                #If player has red jersy
                mask2 = cv2.inRange(player_hsv, lower_red, upper_red)
                res2 = cv2.bitwise_and(player_img, player_img, mask=mask2)
                res2 = cv2.cvtColor(res2,cv2.COLOR_HSV2BGR)
                res2 = cv2.cvtColor(res2,cv2.COLOR_BGR2GRAY)
                nzCountred = cv2.countNonZero(res2)

                if(nzCount >= 20):
                    #Mark blue jersy players as france
                    cv2.putText(image, 'France', (x-2, y-2), font, 0.8,
(255,0,0), 2, cv2.LINE_AA)
                    cv2.rectangle(image,(x,y),(x+w,y+h),(255,0,0),3)
                else:
                    pass
                if(nzCountred>=20):
                    #Mark red jersy players as belgium
                    cv2.putText(image, 'Belgium', (x-2, y-2), font, 0.8,
(0,0,255), 2, cv2.LINE_AA)
                    cv2.rectangle(image,(x,y),(x+w,y+h),(0,0,255),3)
                else:
                    pass
```

To detect the football, we check smaller dimensions and analyse a white colour that matched the football colour from the video:

```
        if((h>=1 and w>=1) and (h<=30 and w<=30)):
            player_img = image[y:y+h,x:x+w]

            player_hsv = cv2.cvtColor(player_img,cv2.COLOR_BGR2HSV)
            #white ball  detection
            mask1 = cv2.inRange(player_hsv, lower_white, upper_white)
            res1 = cv2.bitwise_and(player_img, player_img, mask=mask1)
            res1 = cv2.cvtColor(res1,cv2.COLOR_HSV2BGR)
            res1 = cv2.cvtColor(res1,cv2.COLOR_BGR2GRAY)
            nzCount = cv2.countNonZero(res1)


            if(nzCount >= 3):
                # detect football
                cv2.putText(image, 'football', (x-2, y-2), font, 0.8,
(0,255,0), 2, cv2.LINE_AA)
                cv2.rectangle(image,(x,y),(x+w,y+h),(0,255,0),3)
```

To validate success of video reading and to display the sub-window:

```
    cv2.imwrite("./Cropped/frame%d.jpg" % count, res)
    print('Read a new frame: ', success)    # save frame as JPEG file
    count += 1
    cv2.imshow('Match Detection',image)
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break
    success,image = vidcap.read()

vidcap.release()
cv2.destroyAllWindows()
```

It is also to be noted that there is no specific training *dataset* of sorts as this project works solely based on colour identification from image processing a video frame-by-frame.

Methodology – Goal Prediction Project

We perform binary classification to predict whether a goal will be scored in a football match, using XGBoost algorithm. The dataset used in this code is comprised of 3 HDF5 files, one containing the game and team information, features, and labels.

The XGBoost algorithm is a gradient boosting algorithm that builds an ensemble of decision trees to improve the predictions.

It uses the gradient descent optimization technique to minimize the loss function, and it also includes regularization to prevent overfitting. The algorithm can handle both numeric and categorical features, and it can also perform feature selection and missing value imputation.

Evaluate the performance of the model by computing some metrics (accuracy, recall, precision) and plotting actual vs predicted values.

Perform feature selection using two methods: variance threshold and Lasso regularization.

Train and test the XGBoost model on the reduced feature set obtained by the feature selection methods.

Snippets of the Code:

First, we split the dataset into testing and training and train the data using the XGB Classifier

```
#split train and test data, using the same class ratio on the two sets
X_train, X_test, y_train, y_test = train_test_split(
    df_features,
    df_labels['goal'],
    test_size=0.10,
    stratify=df_labels['goal']
)
```

```
[ ]  model = XGBClassifier()
     model.fit(X_train[features], y_train)
```

```
features = [
    'start_dist_to_goal_a0',
    'end_dist_to_goal_a0',
    'start_dist_to_goal_a1',
    'end_dist_to_goal_a1',
    'start_dist_to_goal_a2',
    'end_dist_to_goal_a2',
    'start_angle_to_goal_a0',
    'end_angle_to_goal_a0',
    'start_angle_to_goal_a1',
    'end_angle_to_goal_a1',
    'start_angle_to_goal_a2',
    'end_angle_to_goal_a2',
    'team_1',
    'team_2'
]
```

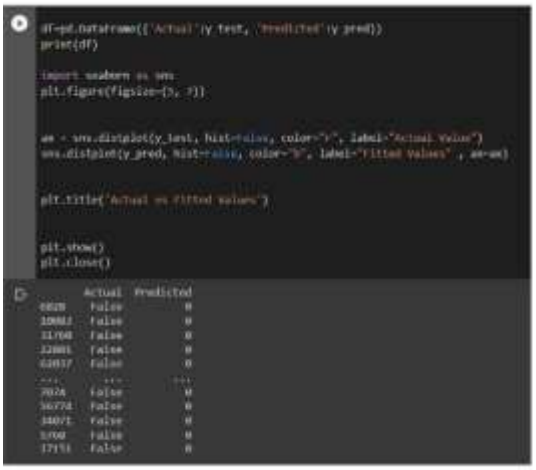
Next, we obtain the 2 class confusion matrix along with the accuracy, precision and recall values.

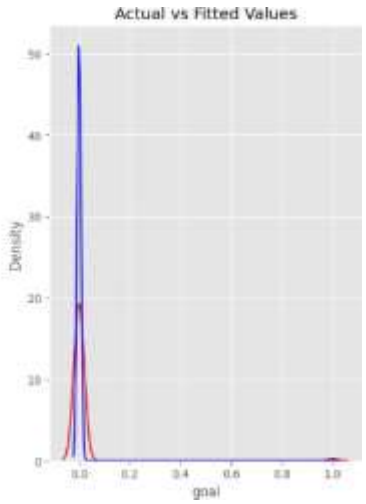
```
def evaluate_results(y_test, y_pred):
    cm = confusion_matrix(y_test, y_pred)
    print("Confusion Matrix:\n", cm)
    print("\nTrue Negatives:", cm[0][0])
    print("False Negatives:", cm[1][0])
    print("Falso Positives:", cm[0][1])
    print("True Positives:", cm[1][1])



    print("Accuracy: ", round(accuracy_score(y_test, y_pred) ,3))
    print("Recall: ", round(recall_score(y_test, y_pred) ,3))
    print("Precision: ", round(precision_score(y_test, y_pred) ,3))

evaluate_results(y_test, y_pred)
```

```
Confusion Matrix:
 [[6179    4]
 [  76   11]]

True Negatives: 6179
False Negatives: 76
Falso Positives: 4
True Positives: 11
Accuracy:  0.987
Recall:  0.126
Precision:  0.733
```

We then test the model and plot the graph between actual and predicted values, that is, goal or no goal

```python
def make_predictions(m, X_data):

    #binary classification
    y_pred = m.predict(X_data)

    #Action value
    probabilities = m.predict_proba(X_data)

    #probability of Goal (class 1)
    y_pred_prob = probabilities[:, 1]

    return y_pred, y_pred_prob


y_pred, y_pred_prob = make_predictions(model, X_test[features])
```





Next, we perform feature selection using Lasso Regression method to selet features with positive coefficients and select features with minimum variance.

```python
#Cross Validation and Grid Search
grid_search = GridSearchCV(
    estimator=XGBClassifier(),
    param_grid=parameters,
    scoring=make_scorer(brier_score_loss),
    refit=True,
    cv = 3
)
```

Finally, We test the model again and predict goal or no goal and this time we obtain a much higher accuracy of 98% .

```python
#Cross Validation and Grid Search
grid_search = GridSearchCV(
    estimator=XGBClassifier(),
    param_grid=parameters,
    scoring=make_scorer(brier_score_loss),
    refit=True,
    cv = 3
)


grid_search.fit(X_train[relevant_features], y_train)
print("Best Parms", grid_search.best_params_)
best_model = grid_search.best_estimator_

#predict and evaluate results
y_pred_tuned, y_pred_prob_tuned = make_predictions(best_model, X_test[relevant_features])
evaluate_results(y_test, y_pred_tuned)
```

```
Best Parms {'learning_rate': 0.05, 'max_depth': 3, 'n_estimators': 20}
Confusion Matrix:
 [[6183    0]
 [  74   13]]

True Negatives: 6183
False Negatives: 74
Falso Positives: 0
True Positives: 13
Accuracy:  0.988
Recall:  0.149
Precision:  1.0
```

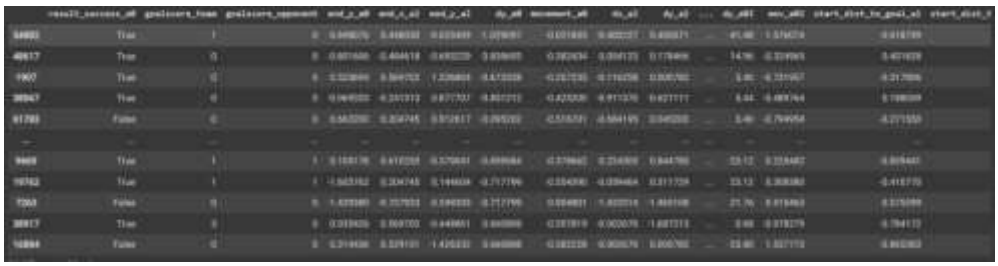
The Dataset for this project involves the following:

- Soccer analytics is attracting increasing interest in academia and industry, thanks to the availability of sensing technologies that provide high-fidelity data streams for every match.

- contains all the spatial-temporal events (passes, shots, fouls, etc.) that occurred during each match for an entire season of seven prominent soccer competitions. Each match event contains information about its position, time, outcome, player, and characteristics.

- The nature of team sports like soccer, halfway between the abstraction of a game and the reality of complex social systems, combined with the unique size and composition of this dataset, provide an ideal ground for tackling a wide range of data science problems, including the measurement and evaluation of performance, both at individual and at collective level, and the determinants of success and failure.

Examples: (Dataset)



Conclusions

The Player Detection using OpenCV was a success, with an example screenshot as shown below:

The Goal Prediction Machine Learning Algorithm/Project was a success as well, with some graphs and predictions shown below:

```
Best Parms {'learning_rate': 0.05, 'max_depth': 3, 'n_estimators': 20}
Confusion Matrix:
 [[6183    0]
 [  74   13]]

True Negatives: 6183
False Negatives: 74
False Positives: 0
True Positives: 13
Accuracy:  0.988
Recall:  0.149
Precision:  1.0


Calibration Diagram:
```