“Practical Machine Learning” Project. Comparing Clustering with Supervised Baseline and Random Chance

STOICA ELIAS - VALERIU
GROUP 412

A. Introduction

The second project for the Machine Learning class involves the developing and evaluation of two cluster methods which can efficiently provide insights from data. Afterwards, they will be compared with a supervised baseline and a random chance approach, in order to prove their accuracy, relevance and practical utility.

For this task, I opted for sports analytics, specifically focusing on football. The objectives are to analyze and compute the similarities between football teams play styles and to identify some patterns in shot placement. The analysis has the potential to provide insights into football differences between leagues or top and bottom teams and their scoring tendencies.

The clustering methods which helped me through this process are **BIRCH** (Balanced Iterative Reducing and Clustering using Hierarchies) and **OPTICS** (Ordering Points to Identify the Clustering Structure), the objective being to find meaningful clusters without using the original labels and to understand the ability of unsupervised learning in uncovering hidden patterns.

B. Dataset Overview

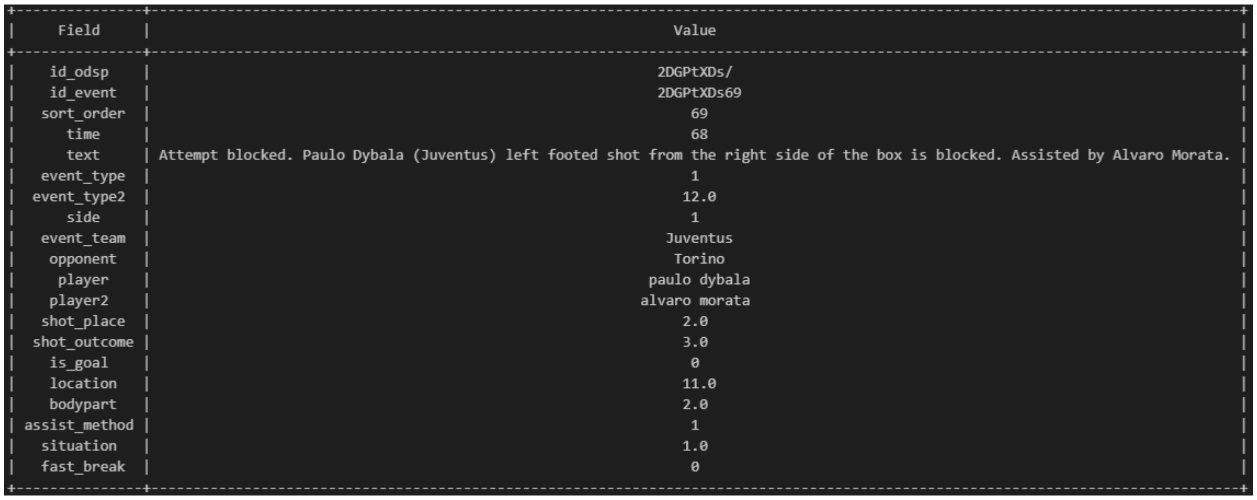
The dataset I chose for this task was found on Kaggle and it’s named Football Events. It provides detailed, event-level data from 9,074 football matches played in Europe’s top 5 leagues (England, Spain, Italy, Germany, and France), between the 2011/2012 and 2016/2017 seasons. It has around 900.000 events, making it a rich source for analyzing football performance.

With this dataset, I had the possibility of going deeper inside match events, to analyze the context behind actions, such as shot attempts, fouls, passes or set pieces. The dataset is aimed to go beyond the traditional statistics like goals or shots, which do not provide enough context for understanding similarities between teams.

I found it appropriate for clustering because it offers the possibility of discovering patterns in playstyle and shot placement, since the detailed event descriptions and contextual information fit the feature engineering and interpretation.

The dataset is organized into three primary files:

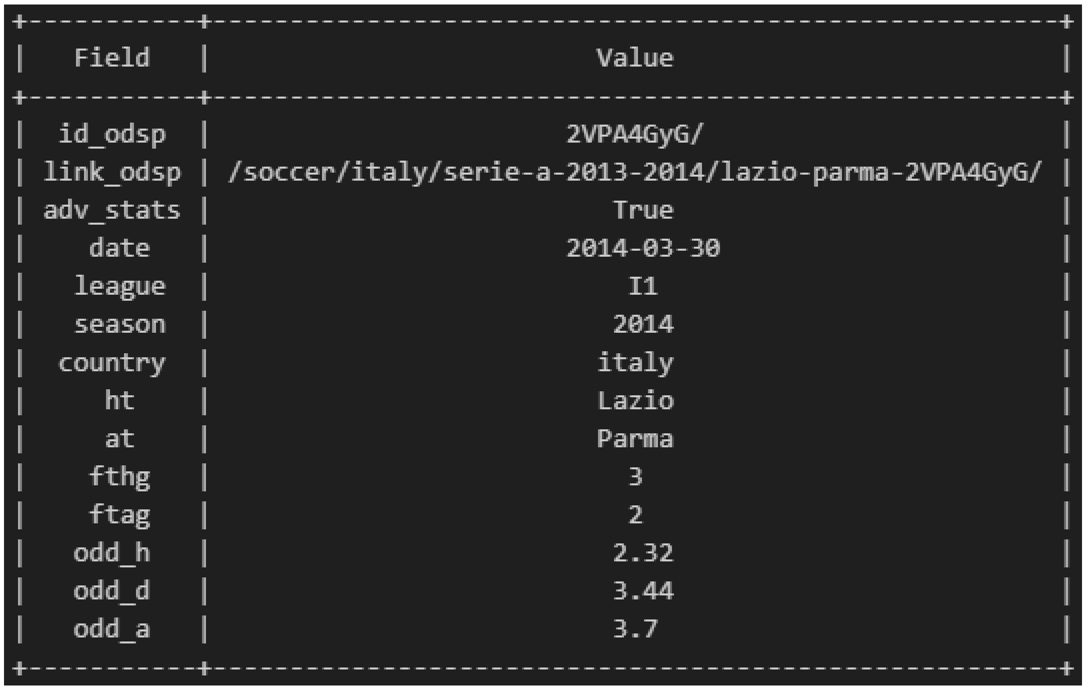
1. **events.csv** - it contains detailed event-level data for each match, derived from text commentary scraped from BBC, ESPN and OneFootball.



```
+--------------+------------------------------------------------------------------------------------------------------------------+
|    Field     |                                                      Value                                                       |
+--------------+------------------------------------------------------------------------------------------------------------------+
|   id_odsp    |                                                    2DGPtXDs/                                                     |
|   id_event   |                                                   2DGPtXDs69                                                     |
|  sort_order  |                                                        69                                                        |
|     time     |                                                        68                                                        |
|     text     | Attempt blocked. Paulo Dybala (Juventus) left footed shot from the right side of the box is blocked. Assisted by Alvaro Morata. |
|  event_type  |                                                        1                                                         |
| event_type2  |                                                       12.0                                                       |
|     side     |                                                        1                                                         |
|  event_team  |                                                     Juventus                                                     |
|   opponent   |                                                      Torino                                                      |
|    player    |                                                   paulo dybala                                                   |
|   player2    |                                                  alvaro morata                                                   |
|  shot_place  |                                                       2.0                                                        |
| shot_outcome |                                                       3.0                                                        |
|   is_goal    |                                                        0                                                         |
|   location   |                                                       11.0                                                       |
|   bodypart   |                                                       2.0                                                        |
| assist_method|                                                        1                                                         |
|  situation   |                                                       1.0                                                        |
|  fast_break  |                                                        0                                                         |
+--------------+------------------------------------------------------------------------------------------------------------------+
```

Fig B.1: Sample from events.csv

2. **ginf.csv** - it contains metadata about each game, including market odds and other statistics; it was used only for computing each league and its teams.



```
+-----------+-----------------------------------------------------+
|   Field   |                        Value                        |
+-----------+-----------------------------------------------------+
|  id_odsp  |                      2VPA4GyG/                      |
| link_odsp | /soccer/italy/serie-a-2013-2014/lazio-parma-2VPA4GyG/ |
| adv_stats |                         True                        |
|    date   |                      2014-03-30                     |
|   league  |                          I1                         |
|   season  |                         2014                        |
|  country  |                        italy                        |
|     ht    |                        Lazio                        |
|     at    |                        Parma                        |
|    fthg   |                          3                          |
|    ftag   |                          2                          |
|   odd_h   |                         2.32                        |
|   odd_d   |                         3.44                        |
|   odd_a   |                         3.7                         |
+-----------+-----------------------------------------------------+
```

Fig B.2: Sample from ginf.csv

3. **dictionary.txt** - it provides a detailed mapping of categorical variables (e.g. event_type codes to descriptive labels: 1 - attempt, 2 - corner etc.)

C. BIRCH Clustering for Football Team Play Style Analysis

1. Overview of **BIRCH**

BIRCH (Balanced Iterative Reducing and Clustering using Hierarchies) is a scalable clustering algorithm designed to efficiently handle large datasets. It incrementally builds a hierarchical clustering tree, called the Clustering Feature (CF) tree, to summarize the dataset in a compact structure.

In this section, BIRCH is applied to analyze football team playstyles, aiming to uncover similarities and differences without relying on league labels.

2. Feature Engineering

The first step in preparing the Football Events dataset for clustering is feature engineering. I managed to transform raw event data into aggregated and meaningful team-level features, so that the clustering algorithm could identify the similarities efficiently.

Data Loading and Filtering

Firstly, I loaded the two primary files from the dataset which I described earlier: *events.csv* and *ginf.csv*. From the dataset containing the events, I kept only the most pertinent columns for my task:

- **event_team** - identifies the team involved in the event.
- **event_type** - describes the type of event (e.g., shot, foul, corner).
- **location** - indicates where the event occurred on the field.
- **assist_method** - details the type of assist leading to a shot.
- **situation**: Specifies the play context (e.g., open play, set piece).
- **bodypart**: Identifies whether the event used the right foot, left foot, or head.
- **shot_place** and **shot_outcome**: Provide critical information about shot accuracy and placement.

To maintain data integrity, the rows with missing values in critical columns (*event_team* and *event_type*) were dropped.

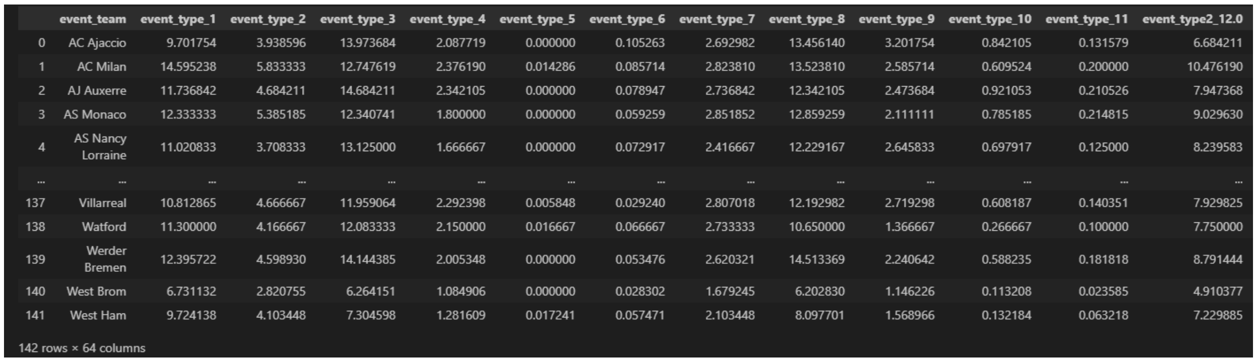
The categorical features were one-hot encoded to convert them into numerical format which is suitable for clustering. In this way, each category will be represented as a separate binary column, to process the features efficiently. The resulting dataset contained 63 binary columns representing event categories.

Aggregating Data at the Team Level

To capture team styles better, the event-level data was aggregated at the team level. All one-hot encoded features were summed for each team and in this way a general view of their statistics across all matches can be observed

Each row in the new dataset represented a team, with the columns describing the frequency of each event and characteristics.

Since the number of games played by each team is different, all event-related features were normalized by the total number of games played. This step prevented teams with more matches from altering the results.



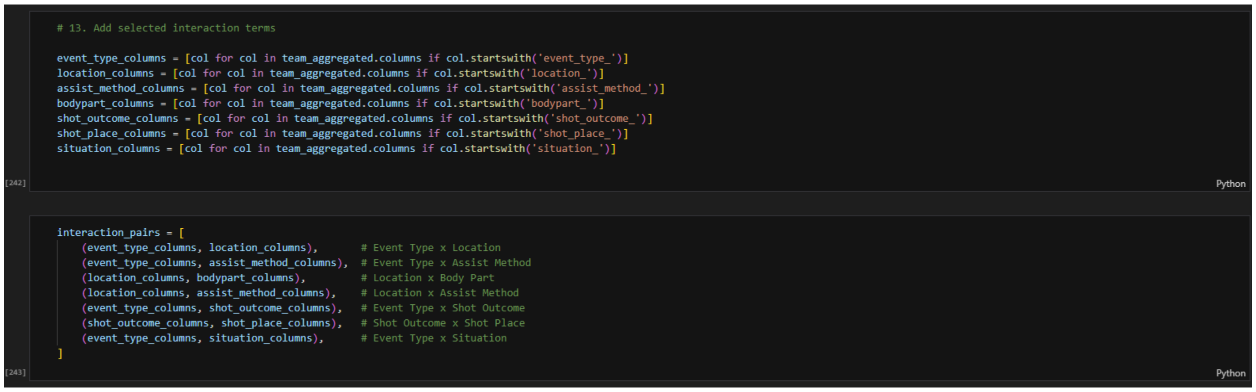
	event_team	event_type_1	event_type_2	event_type_3	event_type_4	event_type_5	event_type_6	event_type_7	event_type_8	event_type_9	event_type_10	event_type_11	event_type2_12.0
0	AC Ajaccio	9.701754	3.938596	13.973684	2.087719	0.000000	0.105263	2.692982	13.456140	3.201754	0.842105	0.131579	6.684211
1	AC Milan	14.595238	5.833333	12.747619	2.376190	0.014286	0.085714	2.823810	13.523810	2.585714	0.609524	0.200000	10.476190
2	AJ Auxerre	11.736842	4.684211	14.684211	2.342105	0.000000	0.078947	2.736842	12.342105	2.473684	0.921053	0.210526	7.947368
3	AS Monaco	12.333333	5.385185	12.340741	1.800000	0.000000	0.059259	2.851852	12.859259	2.111111	0.785185	0.214815	9.029630
4	AS Nancy Lorraine	11.020833	3.708333	13.125000	1.666667	0.000000	0.072917	2.416667	12.229167	2.645833	0.697917	0.125000	8.239583
...	...	...	...	...	...	...	...	...	...	...	...	...	...
137	Villarreal	10.812865	4.666667	11.959064	2.292398	0.005848	0.029240	2.807018	12.192982	2.719298	0.608187	0.140351	7.929825
138	Watford	11.300000	4.166667	12.083333	2.150000	0.016667	0.066667	2.733333	10.650000	1.366667	0.266667	0.100000	7.750000
139	Werder Bremen	12.395722	4.598930	14.144385	2.005348	0.000000	0.053476	2.620321	14.513369	2.240642	0.588235	0.181818	8.791444
140	West Brom	6.731132	2.820755	6.264151	1.084906	0.000000	0.028302	1.679245	6.202830	1.146226	0.113208	0.023585	4.910377
141	West Ham	9.724138	4.103448	7.304598	1.281609	0.017241	0.057471	2.103448	8.097701	1.568966	0.132184	0.063218	7.229885

142 rows × 64 columns

Fig C.1: Computed dataset at team-level

Derived Features and Ratio

In order to capture team behavior more accurately, I decided to add several new features. For example, *ratio_shots_on_target* and *ratio_cards_per_foul* aim to provide important insights into aspects like shooting accuracy and discipline.



```python
# 13. Add selected interaction terms

event_type_columns = [col for col in team_aggregated.columns if col.startswith('event_type_')]
location_columns = [col for col in team_aggregated.columns if col.startswith('location_')]
assist_method_columns = [col for col in team_aggregated.columns if col.startswith('assist_method_')]
bodypart_columns = [col for col in team_aggregated.columns if col.startswith('bodypart_')]
shot_outcome_columns = [col for col in team_aggregated.columns if col.startswith('shot_outcome_')]
shot_place_columns = [col for col in team_aggregated.columns if col.startswith('shot_place_')]
situation_columns = [col for col in team_aggregated.columns if col.startswith('situation_')]
```

```python
interaction_pairs = [
    (event_type_columns, location_columns),        # Event Type x Location
    (event_type_columns, assist_method_columns),   # Event Type x Assist Method
    (location_columns, bodypart_columns),          # Location x Body Part
    (location_columns, assist_method_columns),     # Location x Assist Method
    (event_type_columns, shot_outcome_columns),    # Event Type x Shot Outcome
    (shot_outcome_columns, shot_place_columns),    # Shot Outcome x Shot Place
    (event_type_columns, situation_columns),       # Event Type x Situation
]
```

Fig C.2: Adding interaction terms

Additionally, interaction terms were created to explore relationships between different feature categories, such as:

- **Event Type x Location** - capturing how event types vary by location on the pitch.
- **Shot Outcome x Shot Place** - linking shot success to specific goal placements.
- **Event Type x Situation** - understanding how event types vary under different play contexts (e.g., open play vs. set piece).

The aim of these features is to enrich the data space and to allow the clustering algorithm to consider more relationships than the default ones.

To reduce noise and computational complexity, I opted to remove low-variance features (the ones with almost no variation across teams). A threshold of 0.001 was used to identify and drop such features, so if the variation is as little as possible, the feature is taken out since it does not help clustering.

Furthermore, highly correlated features were also identified using a correlation matrix. Then, a hierarchical clustering approach was applied to group the correlated features into clusters, so that from each cluster of features, the one with the highest variance was retained as the representative features, to minimize redundancy.

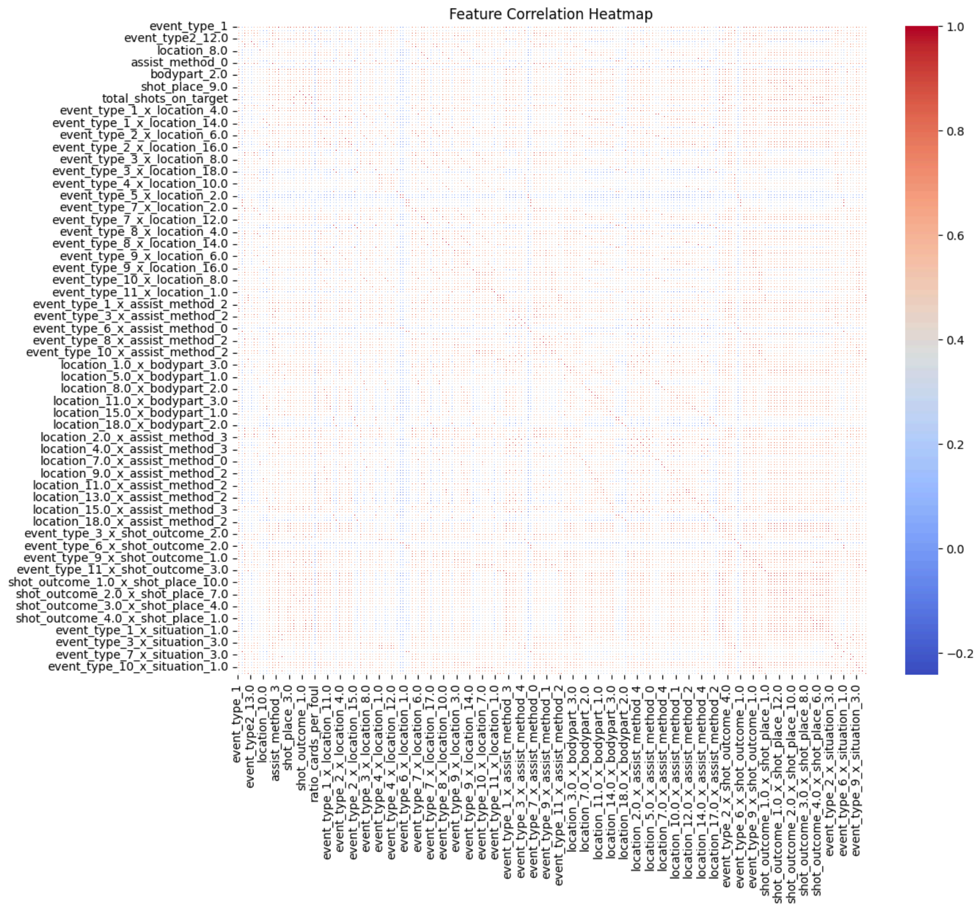


Fig C.3: Feature Correlation Heatmap

Scaling and Saving the Dataset

Finally, I scaled all numerical features using *StandardScaler* to normalize the feature values. The final processed dataset containing 161 scaled and cleaned features was saved for the clustering process. Each row represents a team, and the columns capture various play style characteristics derived from the raw event data.

3. Algorithm Implementation

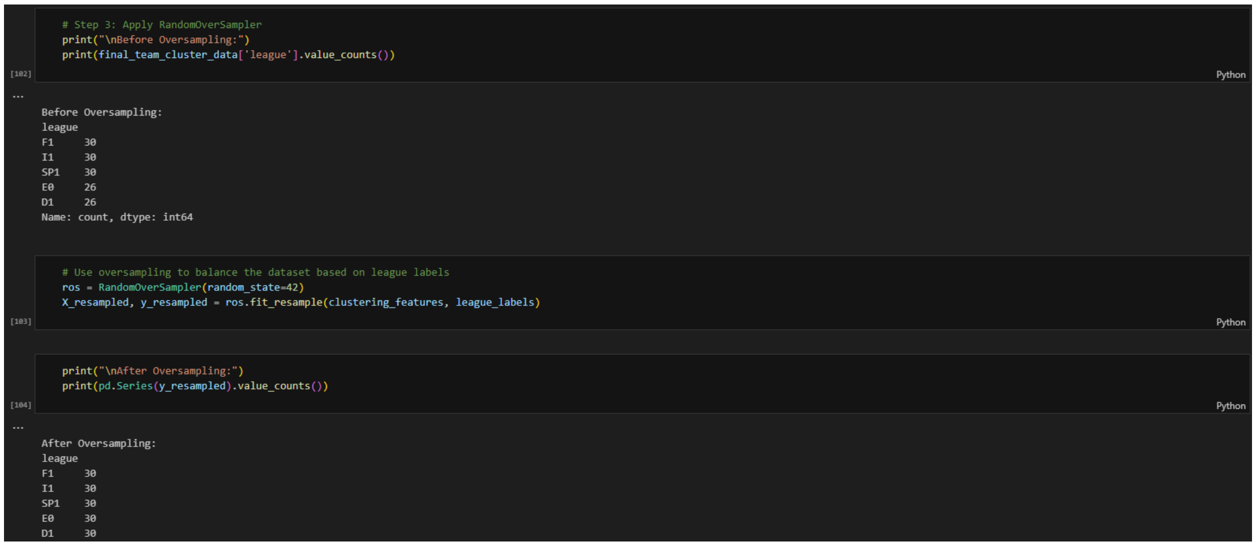
Based on the features engineered from event-level data, the BIRCH algorithm was applied to cluster football teams based on their play styles.

Loading and Balancing the Data

The prepared dataset was loaded for clustering, including the team name and the actual league of the team, which is used for evaluation purposes only.

The league column was mapped and verified to ensure that each team had the correct league label. Since the distribution of the teams across leagues showed a slight imbalance, which needed to be addressed before clustering. Therefore, oversampling was applied to ensure a fair representation of all leagues. I used *RandomOverSampler* to balance the number of teams from each league by duplicating teams from underrepresented leagues.

After oversampling, each league now has 30 teams, ensuring equal representation during clustering.



```
# Step 3: Apply RandomOverSampler
print("\nBefore Oversampling:")
print(final_team_cluster_data['league'].value_counts())
```
[102] Python

```
...
Before Oversampling:
league
F1     30
I1     30
SP1    30
E0     26
D1     26
Name: count, dtype: int64
```

```
# Use oversampling to balance the dataset based on league labels
ros = RandomOverSampler(random_state=42)
X_resampled, y_resampled = ros.fit_resample(clustering_features, league_labels)
```
[103] Python

```
print("\nAfter Oversampling:")
print(pd.Series(y_resampled).value_counts())
```
[104] Python

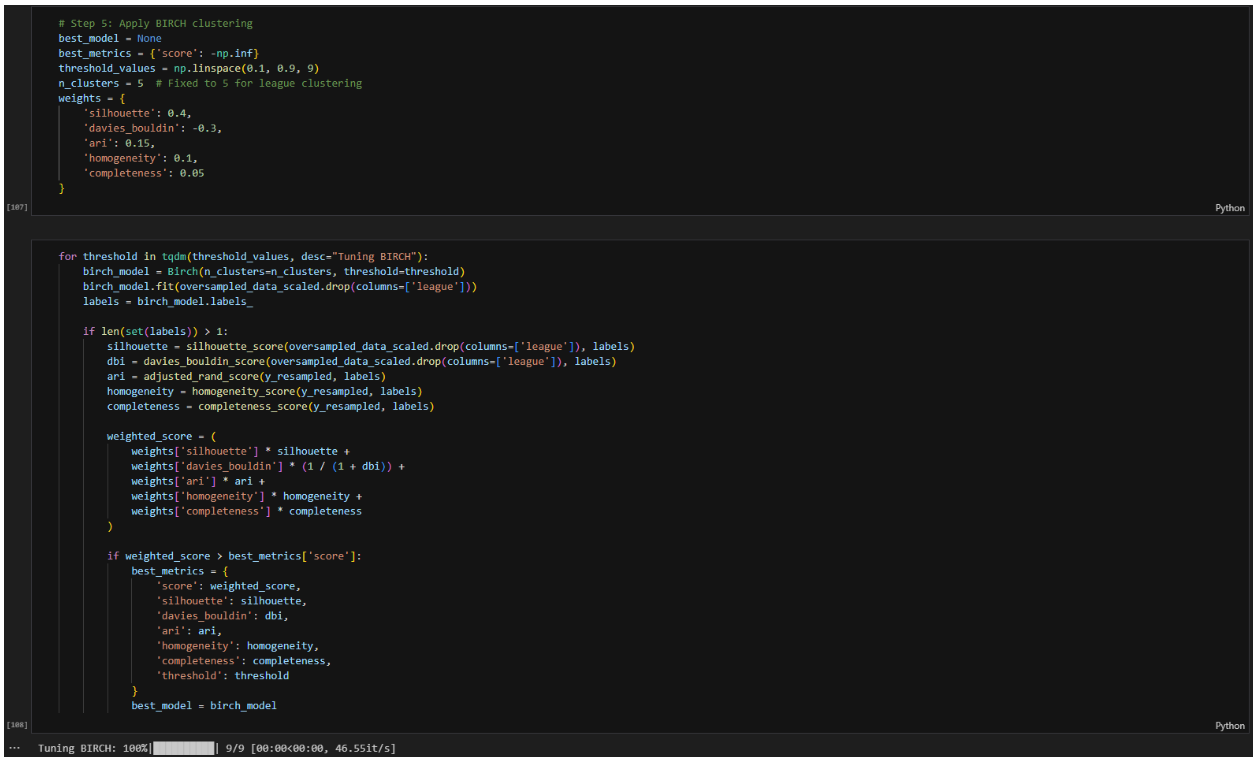
```
...
After Oversampling:
league
F1     30
I1     30
SP1    30
E0     30
D1     30
```

Fig C.4: Oversampling to balance the dataset

BIRCH Clustering and Hyperparameter Tuning

The two main hyperparameters of the BIRCH algorithm were tuned to optimize clustering performance:

- **threshold** - controls the maximum radius of sub-clusters in the Clustering Feature (CF) tree. Lower values create finer, more granular clusters, while higher values produce broader clusters.
- **Number of Clusters (n_clusters)** - fixed to 5, corresponding to the number of leagues. This allowed for direct comparison of the identified clusters to actual league labels.



```
# Step 5: Apply BIRCH clustering
best_model = None
best_metrics = {'score': -np.inf}
threshold_values = np.linspace(0.1, 0.9, 9)
n_clusters = 5  # Fixed to 5 for league clustering
weights = {
    'silhouette': 0.4,
    'davies_bouldin': -0.3,
    'ari': 0.15,
    'homogeneity': 0.1,
    'completeness': 0.05
}
```
[107] Python

```
for threshold in tqdm(threshold_values, desc="Tuning BIRCH"):
    birch_model = Birch(n_clusters=n_clusters, threshold=threshold)
    birch_model.fit(oversampled_data_scaled.drop(columns=['league']))
    labels = birch_model.labels_

    if len(set(labels)) > 1:
        silhouette = silhouette_score(oversampled_data_scaled.drop(columns=['league']), labels)
        dbi = davies_bouldin_score(oversampled_data_scaled.drop(columns=['league']), labels)
        ari = adjusted_rand_score(y_resampled, labels)
        homogeneity = homogeneity_score(y_resampled, labels)
        completeness = completeness_score(y_resampled, labels)

        weighted_score = (
            weights['silhouette'] * silhouette +
            weights['davies_bouldin'] * (1 / (1 + dbi)) +
            weights['ari'] * ari +
            weights['homogeneity'] * homogeneity +
            weights['completeness'] * completeness
        )

        if weighted_score > best_metrics['score']:
            best_metrics = {
                'score': weighted_score,
                'silhouette': silhouette,
                'davies_bouldin': dbi,
                'ari': ari,
                'homogeneity': homogeneity,
                'completeness': completeness,
                'threshold': threshold
            }
            best_model = birch_model
```
[108] Python

```
Tuning BIRCH: 100%|██████████| 9/9 [00:00<00:00, 46.55it/s]
```

Fig C.5: Tuning process

In order to find the optimal configuration, I opted for a grid search which was conducted over a range of *threshold* values (0.1 to 0.9). For each try, the following metrics were computed:

- **Silhouette Score** - it measures the separation and cohesion of clusters (higher is better).
- **Davies-Bouldin Index** - it evaluates cluster compactness and separation (lower is better).
- **Adjusted Rand Index (ARI)** - it assesses the similarity of clustering results to ground truth labels.

- **Homogeneity** - it measures how well each cluster contains only members of a single class.
- **Completeness** - it measures how well all members of a given class are assigned to the same cluster.

In order to find the best configuration, I chose to combine the metrics into a weighted score, with higher weights given to Silhouette Score and Davies-Bouldin Index due to their importance in unsupervised clustering.

The best result were found for the optimal **threshold value 0.1**:

- Silhouette Score: 0.18
- Davies-Bouldin Index: 1.74
- Adjusted Rand Index (ARI): 0.19
- Homogeneity: 0.32
- Completeness: 0.37
- Weighted Score: 0.04

The moderate performance of BIRCH clustering indicates that football play styles are complex and can't be strictly tied to leagues.

The lower Silhouette Score and ARI imply that football strategies quite overlap and some teams do not represent the exact stereotype of their leagues. For example, an English team might adopt a style based on possession and domination, similar to Spanish teams, which makes the clustering harder.

The Homogeneity and Completeness scores show that the clusters partially align with leagues but also reveal inter-league similarities, suggesting that some clusters capture cross-league play style trends.

Assigning Clusters and Analyzing Results

I wanted to try to compute a mapping between clusters and leagues, in order to find the best correspondence based on league proportions in each cluster. The best mapping was:

1. Cluster 0: I1 (Italy)
2. Cluster 1: E0 (England)
3. Cluster 2: F1 (France)
4. Cluster 3: D1 (Germany)
5. Cluster 4: SP1 (Spain)

The accuracy of the clustering approach was computed by comparing predicted leagues (based on clusters) to actual leagues. The resulting accuracy was 43%, highlighting that BIRCH identified playstyle similarities that partially aligned with league labels.

Visualizing Clusters

In order to visualize the clusters, I utilised PCA (Principal Component Analysis) to reduce the high-dimensional feature space to two dimensions. The plot below shows the resulting clusters, with teams labeled and colored by their assigned one.

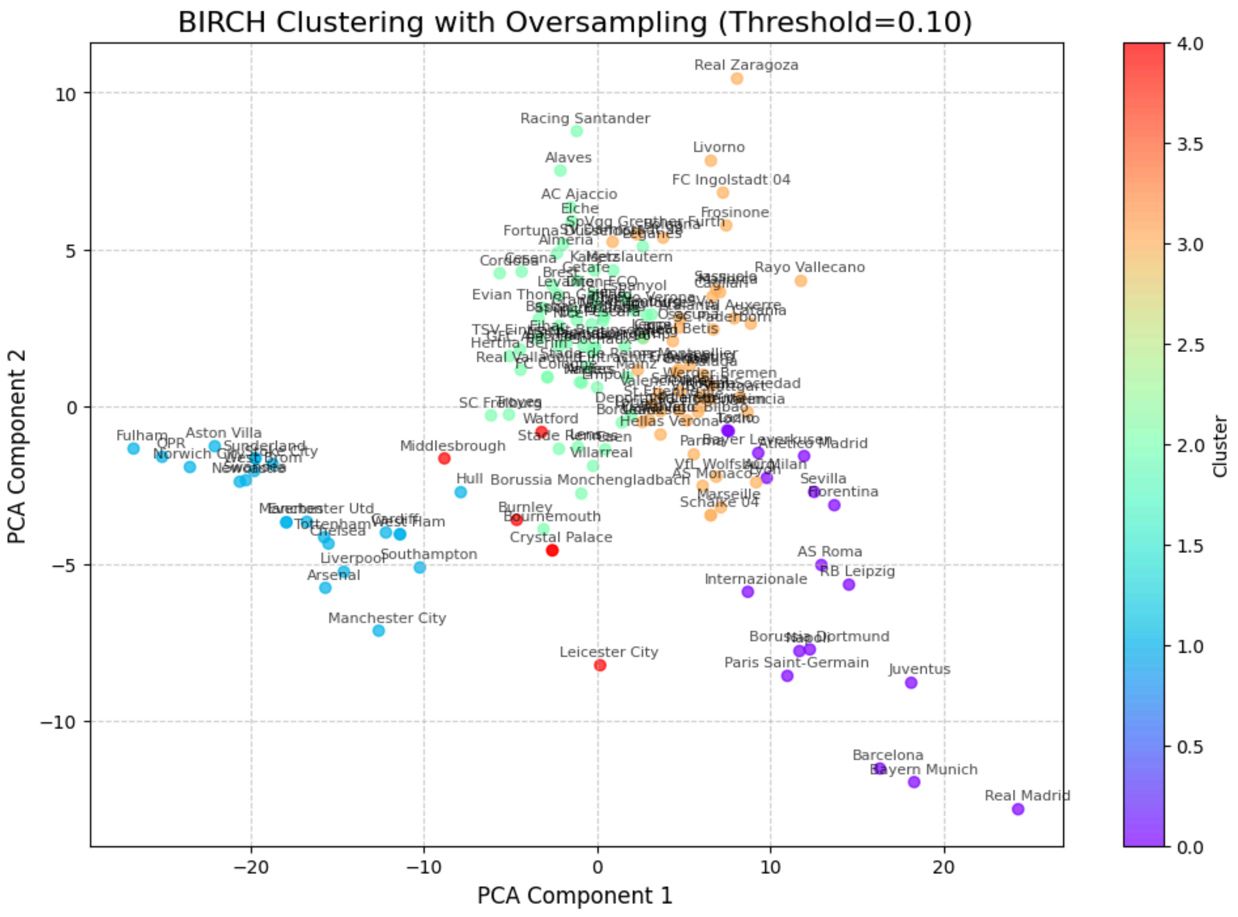


Fig C.6: Clusters Visualization

At first sight, we can see that teams like Real Madrid, Barcelona, Bayern Munich, Juventus, PSG and Dortmund stand out, indicating some unique play styles with dominant performance metrics.

On the left side, we can see that the Cluster 1 perfectly groups the English Premier League which is marked by high-paced and physically intense football. Even though there is a big difference of value between bottom and top teams in England, the inclusion of teams with diverse styles suggests that all teams share some common traits.

The orange cluster contains a niche group potentially related to bottom teams which struggle to compete and are found in the relegation zone.

The little red cluster stands out by separating some English teams. This might be due to Leicester City's historical performance metrics (its Premier League winning season from 2015-2016). Similarly, Crystal Palace and Burnley diverge from the main English cluster.

The overlapping clusters, 2 and 3, have teams from multiple leagues, indicating that some patterns are found beyond league boundaries, mostly for the mid-table teams from every league, which tend to have a balanced football.

Conclusions

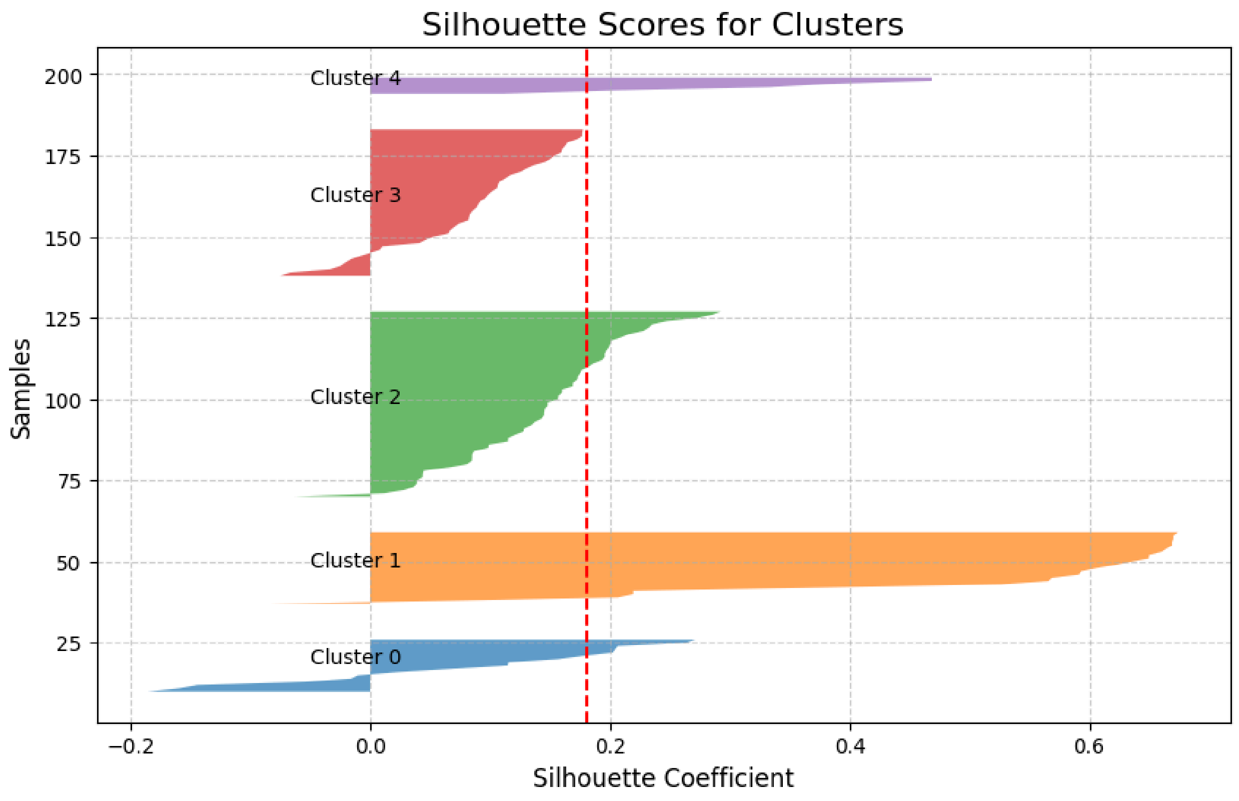


Fig C.7: Silhouette score for each cluster

As we saw in the previous scatter plot, Cluster 1 and Cluster 4 have the highest silhouette scores, suggesting that they are the most well-separated ones. The other ones present a little variation and overlap with neighboring clusters, with teams from different leagues being grouped together, fact indicated by the lower silhouette values.

England shows stronger associations with specific clusters, which means that the teams have more distinctive play style and characteristics.

In the other countries, where the teams are spread across multiple clusters, it is clear that the play style and strategy of top teams are totally opposite to the bottom ones, so they are grouped better on value rather than league.

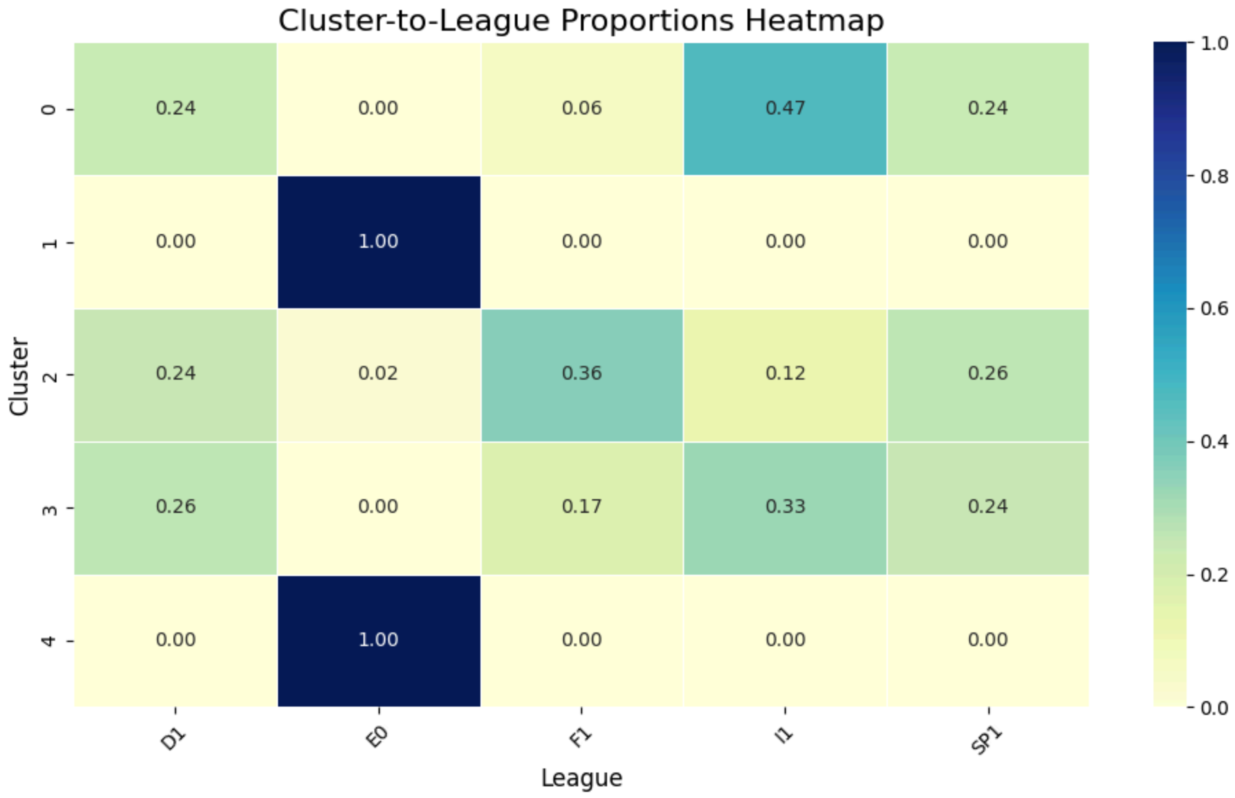


Fig C.8: Cluster-to-League proportions heatmap

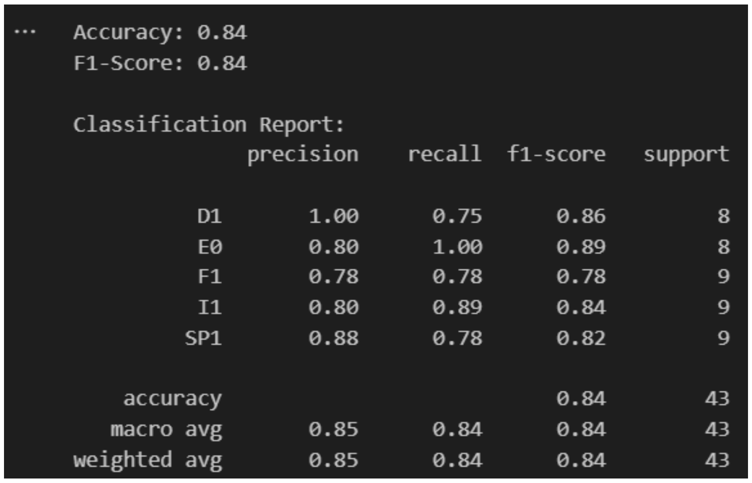
4. Comparison of Clustering Results with Supervised Baseline and Random Chance

The BIRCH algorithm achieved moderate accuracy, reflecting its ability to group teams based on interesting characteristics. To understand and evaluate better the clustering results, I will compare it with a supervised Random Forest classifier and a general approach using random chance.

Supervised Method (Random Forest Classifier)

Accuracy: 84%
F1-Score (Weighted): 0.84
Classification Report Highlights:

- High precision and recall for leagues like E0 (Premier League) and D1 (Bundesliga).
- Slightly lower performance for F1 (Ligue 1) and SP1 (La Liga), possibly due to more variability in playstyles within these leagues.



```
...  Accuracy: 0.84
     F1-Score: 0.84

     Classification Report:
                   precision    recall  f1-score   support

               D1       1.00      0.75      0.86         8
               E0       0.80      1.00      0.89         8
               F1       0.78      0.78      0.78         9
               I1       0.80      0.89      0.84         9
              SP1       0.88      0.78      0.82         9

         accuracy                           0.84        43
        macro avg       0.85      0.84      0.84        43
     weighted avg       0.85      0.84      0.84        43
```

Fig C.9: Classification report for supervised baseline

The supervised approach significantly outperformed the BIRCH clustering, demonstrating that the league labels allow the model to understand clear patterns in the data. Therefore, it looks like the labeled data is crucial and offers better accuracy.

There is a minimal number of wrong classifications, highlighting the perfect combination of feature engineering and supervised learning for identifying complex patterns.

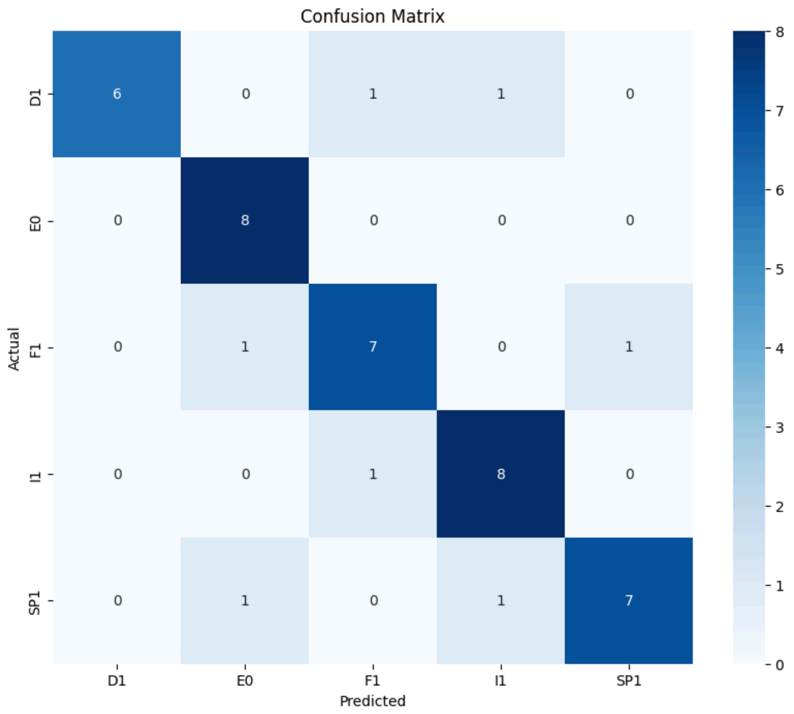
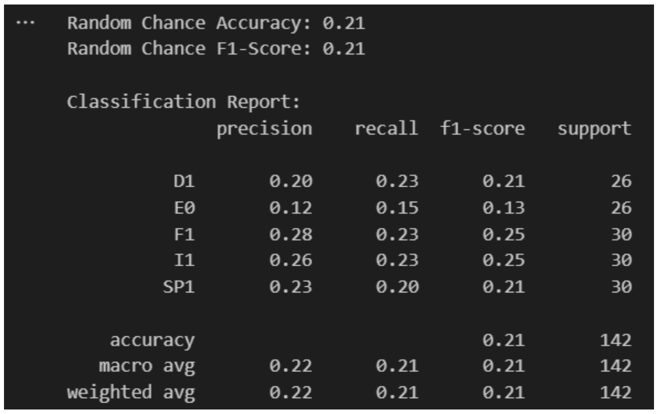


Fig C.10: Confusion Matrix for Supervised Baseline

Random Chance Approach

Accuracy: 21%
F1-Score (Weighted): 0.21
Classification Report Highlights: Precision and recall are uniformly poor across all leagues, reflecting the randomness of the predictions.



```
...  Random Chance Accuracy: 0.21
     Random Chance F1-Score: 0.21

     Classification Report:
                   precision    recall  f1-score   support

              D1       0.20      0.23      0.21        26
              E0       0.12      0.15      0.13        26
              F1       0.28      0.23      0.25        30
              I1       0.26      0.23      0.25        30
             SP1       0.23      0.20      0.21        30

        accuracy                           0.21       142
       macro avg       0.22      0.21      0.21       142
    weighted avg       0.22      0.21      0.21       142
```

Fig C.11: Classification report for random chance

The random chance approach establishes a lower bound for performance, its score showing that random predictions are far from being meaningful. Having 5 labels, it does not capture anything, achieving an accuracy of 0.21.

The predictions are spread across all classes, accuracy is low across all leagues and in this way I understood the BIRCH's accuracy which is a significant improvement and a relevant approach.

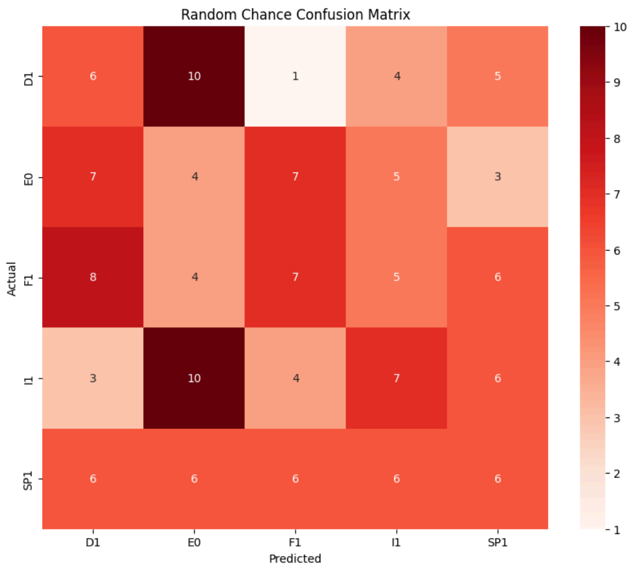


Fig C.12: Confusion Matrix for random approach

D. OPTICS Clustering for Shot Placement Analysis

1. Overview of **OPTICS**

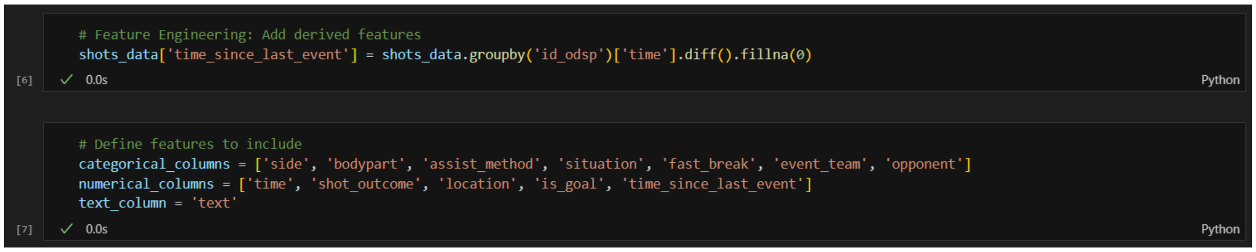
OPTICS (Ordering Points To Identify the Clustering Structure) is a density-based clustering algorithm designed to identify clusters of varying shapes and densities in large datasets. Unlike traditional clustering methods, OPTICS does not require the number of clusters to be specified. Instead, it generates a reachability plot to reveal the hierarchical structure of the data, making it particularly effective for complex, high-dimensional datasets.

In this section, OPTICS is applied to cluster football shot events based on their placement on the goal, without relying on the original shot_place labels. The algorithm leverages a combination of numerical, categorical, and textual features to group shots based on their spatial and contextual characteristics.

2. Preprocessing and Feature Engineering

In order to prepare the dataset for clustering, I filtered it and retained only rows with valid *shot_place* values, which represents the location of the shot on the goal. Since the dataset has more than 900.000 rows, I randomly selected 1000 samples to optimize execution time while retaining statistical integrity.

To enhance the clustering, I decided to also add some new derived features. Some of them are *time_since_last_event* which computes the time elapsed since the previous event, capturing some context for the shot and other categorical, numerical and text features. Shot descriptions were processed using TF-IDF Vectorization to capture textual nuances in shot commentaries.



```
# Feature Engineering: Add derived features
shots_data['time_since_last_event'] = shots_data.groupby('id_odsp')['time'].diff().fillna(0)
```
[6] ✓ 0.0s Python

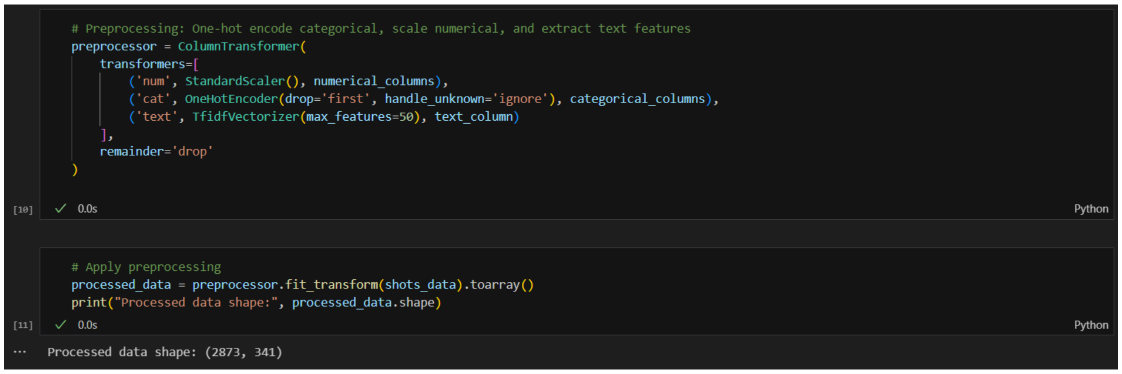
```
# Define features to include
categorical_columns = ['side', 'bodypart', 'assist_method', 'situation', 'fast_break', 'event_team', 'opponent']
numerical_columns = ['time', 'shot_outcome', 'location', 'is_goal', 'time_since_last_event']
text_column = 'text'
```
[7] ✓ 0.0s Python

Fig D.1: Feature engineering

To ensure the fair representation of all *shot_place* categories, I used *RandomOverSampler* similar to the previous BIRCH approach. In this way, I was assured that the dataset is balanced across shot placement classes. This step increased the dataset size to 2873 samples from 13 different classes.

3. OPTICS Clustering Implementation

The preprocessing stage consists of 3 different stages. Firstly, I applied one-hot encoding to categorical variables, converting them into binary format. Afterwards, I used *StandardScaler* to standardize numerical features, to ensure uniform contribution to clustering. As mentioned before, the textual shot descriptions were processed using *TF-IDF Vectorization* into a vectorized numerical format. In this way, I computed the new dataset with 341 features, exceeding the requirement of at least 100.



```
# Preprocessing: One-hot encode categorical, scale numerical, and extract text features
preprocessor = ColumnTransformer(
    transformers=[
        ('num', StandardScaler(), numerical_columns),
        ('cat', OneHotEncoder(drop='first', handle_unknown='ignore'), categorical_columns),
        ('text', TfidfVectorizer(max_features=50), text_column)
    ],
    remainder='drop'
)
```
[10] ✓ 0.0s Python

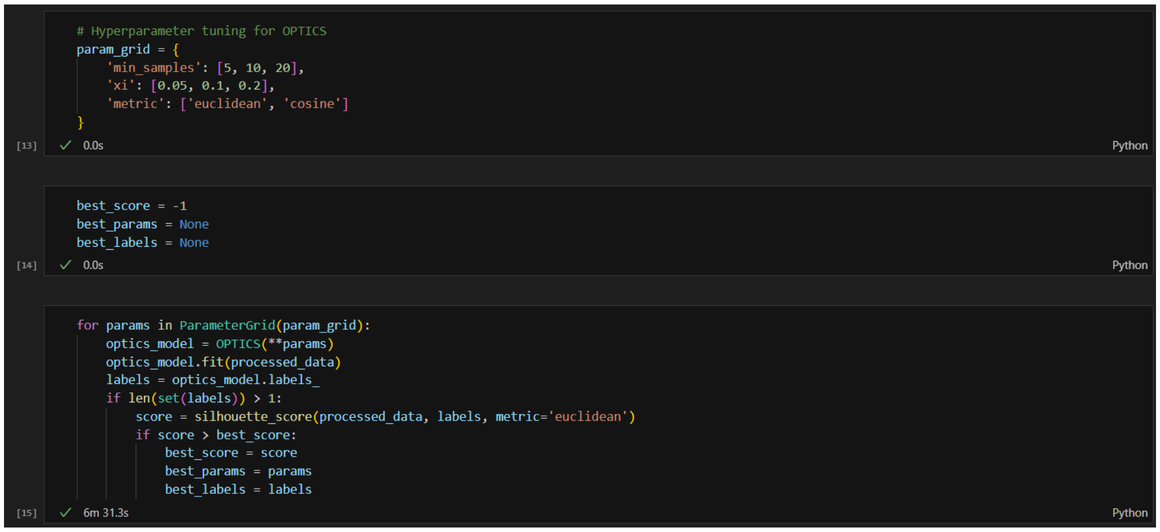
```
# Apply preprocessing
processed_data = preprocessor.fit_transform(shots_data).toarray()
print("Processed data shape:", processed_data.shape)
```
[11] ✓ 0.0s Python

```
Processed data shape: (2873, 341)
```

Fig D.2: Preprocessing

To optimize the key parameters of OPTICS, I performed a grid search:

- min_samples - minimum points in a cluster (values: 5, 10, 20).
- xi - steepness of the reachability plot for cluster identification (values: 0.05, 0.1, 0.2).
- metric - distance metric for clustering (values: Euclidean, Cosine).



```
# Hyperparameter tuning for OPTICS
param_grid = {
    'min_samples': [5, 10, 20],
    'xi': [0.05, 0.1, 0.2],
    'metric': ['euclidean', 'cosine']
}
```
[13] ✓ 0.0s Python

```
best_score = -1
best_params = None
best_labels = None
```
[14] ✓ 0.0s Python

```
for params in ParameterGrid(param_grid):
    optics_model = OPTICS(**params)
    optics_model.fit(processed_data)
    labels = optics_model.labels_
    if len(set(labels)) > 1:
        score = silhouette_score(processed_data, labels, metric='euclidean')
        if score > best_score:
            best_score = score
            best_params = params
            best_labels = labels
```
[15] ✓ 6m 31.3s Python

Fig D.3: Hyperparameter tuning

The best configuration was found with the following parameters:

- **min_samples: 5**
- **xi: 0.05**
- **metric: Cosine**

4. Evaluation Metrics

In order to evaluate the approach, I used different plots and visualizations. The reachability plot visualizes the clustering structure of the data. Plateaus in the plot indicate potential clusters, while the spikes represent noise.

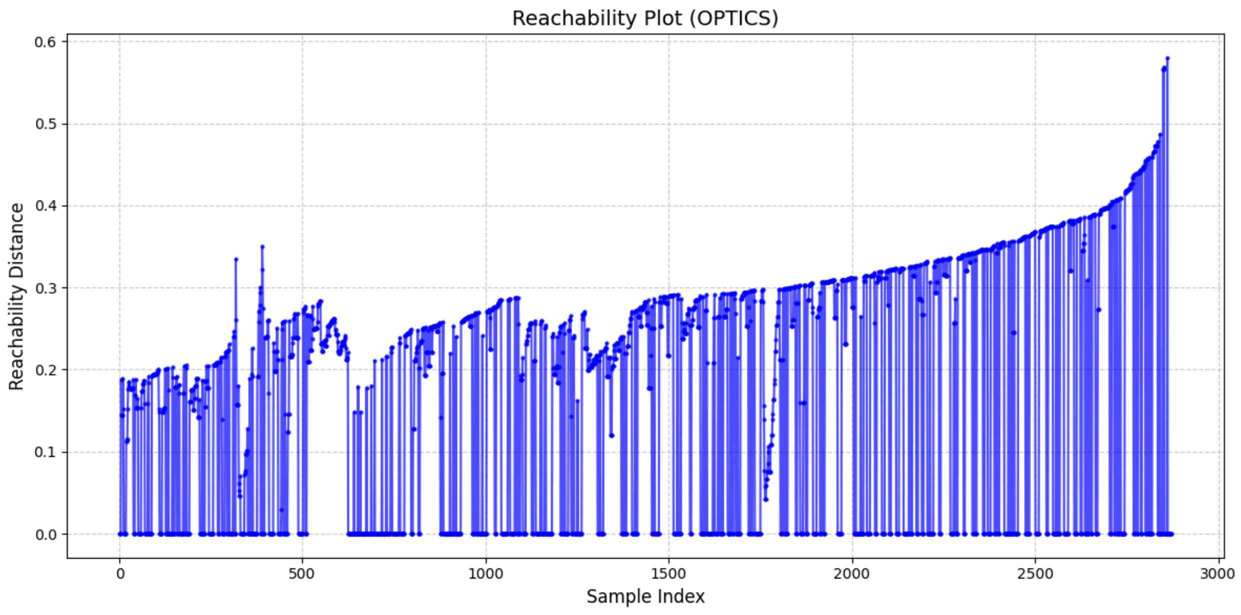


Fig D.4: Reachability plot

I also computed the most relevant metrics for this clustering method:

- Silhouette Score: 0.3483 - it indicates moderate separation and cohesion among clusters. A higher score would reflect better-defined clusters.
- Davies-Bouldin Index: 1.1776 - it suggests compact and well-separated clusters, with lower values being better.
- Calinski-Harabasz Score: 22.0580 - it reflects the ratio of intra-cluster variance to inter-cluster variance, with higher values indicating better-defined clusters.

In the same way I did for BIRCH, I also tried to compute a mapping between clusters and *shot_place* labels. Even though OPTICS found 204 clusters, the proportion of *shot_place* within each cluster was analyzed, and the most common *shot_place* for each cluster was assigned as its label. This mapping provided a basis for evaluating the clustering accuracy. I found the clustering accuracy for predicting *shot_place* at 62.16%.

This demonstrates the ability of OPTICS to group shots into meaningful clusters that align with actual shot placements.

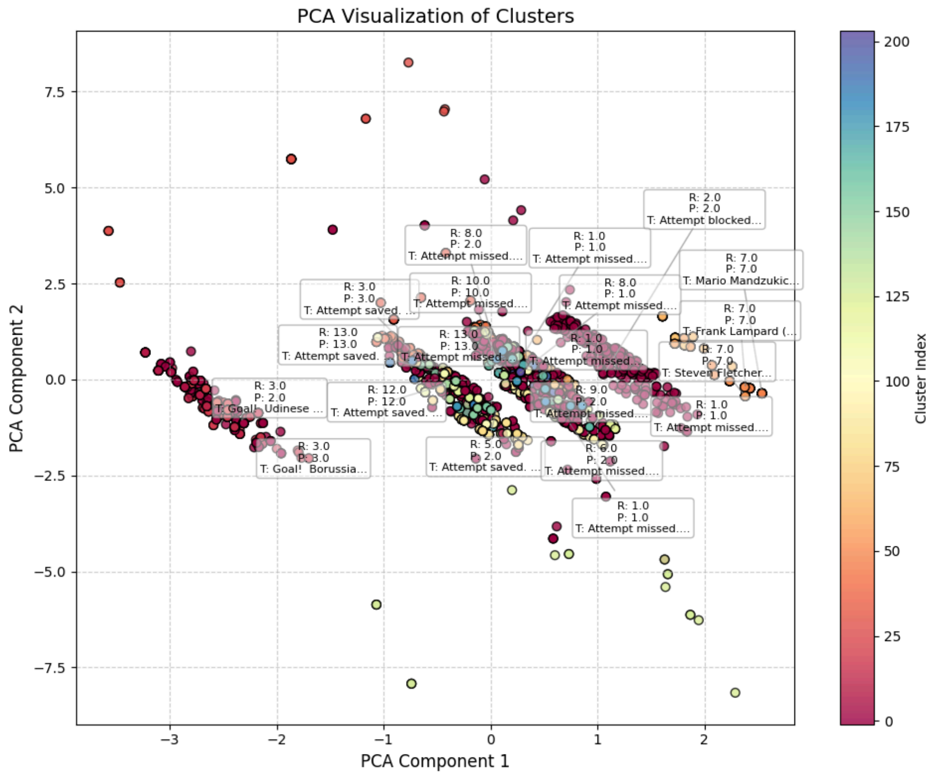


Fig D.5: Visualization of clusters

5. Results and Analysis

The clusters were projected into a 2D space using PCA (Principal Component Analysis), resulting in a plot that showcased distinct groups with some overlap but generally good accuracy. While the goal was to group the data into 13 classes, OPTICS identified more intricate feature similarities, forming 204 clusters instead. This approach effectively captured outliers and unique shot characteristics in smaller clusters.

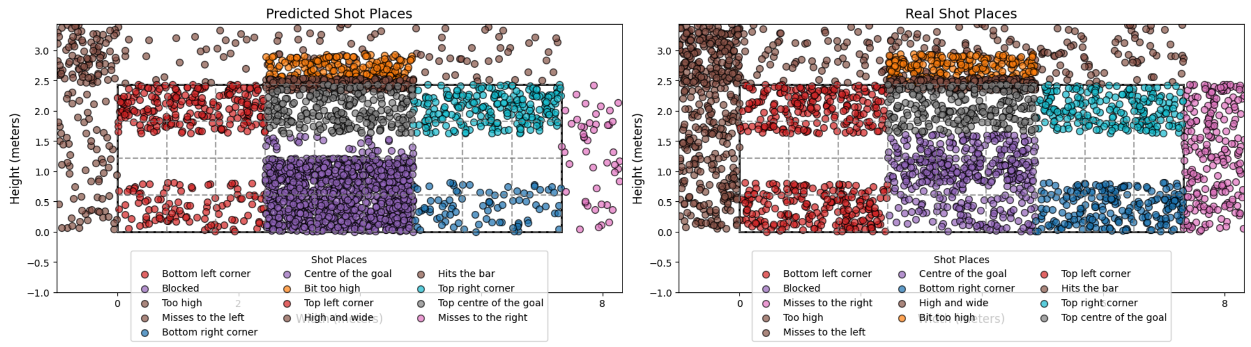


Fig D.6: Predicted vs Real shot places

The previous visualisation compares the predicted shot placement with the real shot placements in the context of a soccer goal. In general, the predicted placements are very similar to the real ones, particularly for top and bottom corners, but also for the top centre of the goal. Some dispersion can be seen for categories like center of the goal and misses or too high shots. The algorithm tends to attribute lots of the shots to the overall central location, suggesting slight inaccuracies in distinguishing these events.

I wanted to visualize the clustering in a "football" view to effectively understand the interpretation of shot categories in a direct comparison. The plots provide valuable insights into how well the algorithm handles less common categories like "Bit too high" or "Hits the bar".

6. Feature importance

The loadings from PCA revealed the top features driving the clustering:

- *shot_outcome* and *is_goal* - major contributors, as expected, since shot placement often correlates with outcomes.
- *time* and *location* - capture the spatial and temporal context of the shot.
- text features - certain TF-IDF features influenced clustering, indicating the value of commentary in distinguishing shots.

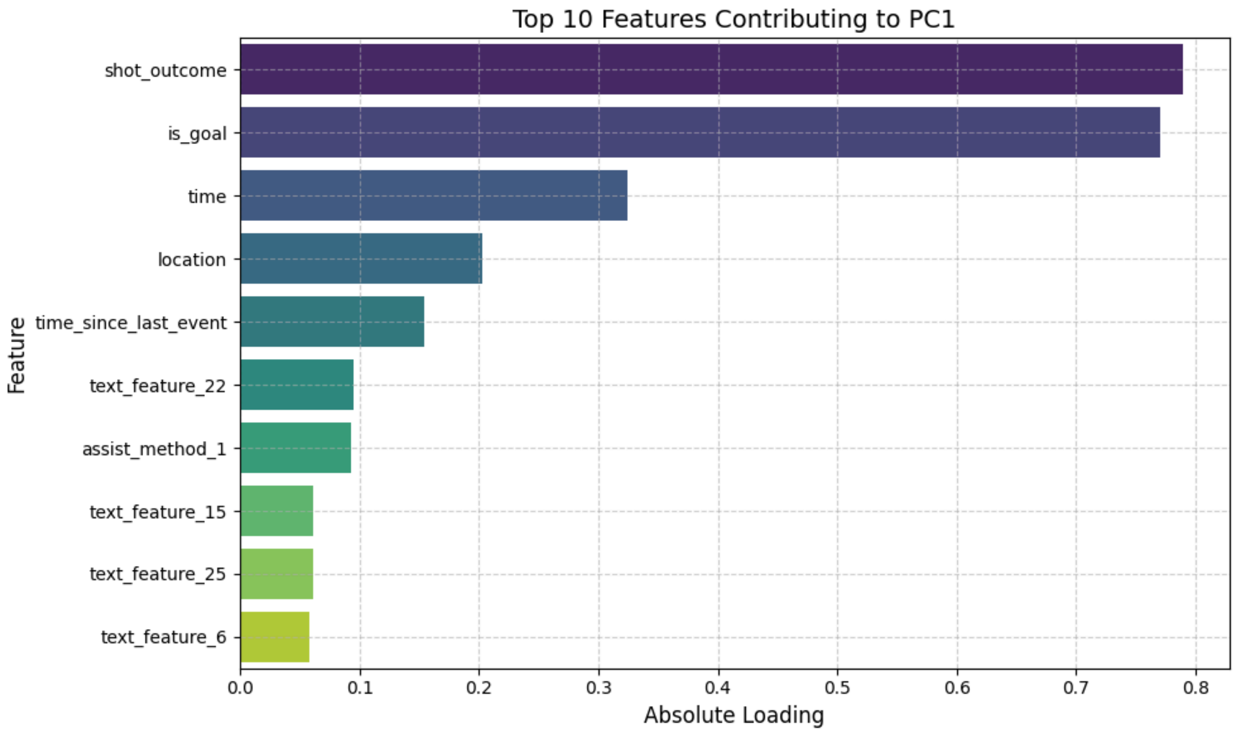


Fig D.7: Top 10 features contributing to clustering

7. Comparison of Clustering Results with Supervised Baseline and Random Chance

The OPTICS clustering approach achieved moderate success in grouping shots based on their placement on the goal. To better understand the effectiveness of this clustering technique, we compared its results with two baselines: **a supervised Random Forest classifier**, which leverages labeled data for training and **a random chance baseline**, where predictions are assigned randomly to establish a lower bound for performance.

Supervised Method (Random Forest Classifier)

Accuracy: 91.3%
Feature Importance:

- The top contributing features were *shot_outcome*, *time*, and *location*.
- Contextual features like *side* and *assist_method* also played significant roles.

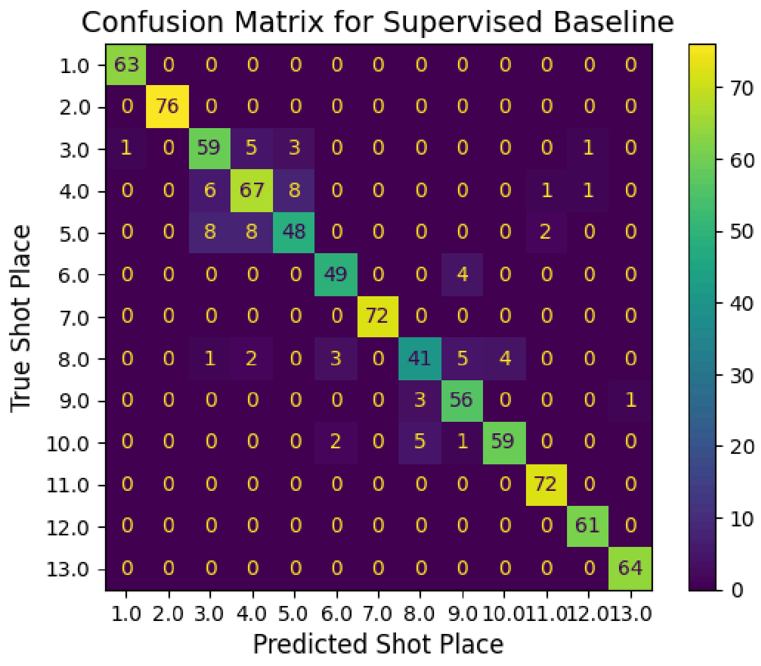


Fig D.8: Confusion Matrix for supervised baseline

The supervised approach significantly outperformed the OPTICS clustering, achieving an impressive accuracy of 91.3%. The Random Forest classifier effectively captured subtle patterns in the data due to its ability to leverage labeled information. The confusion matrix highlights the model's exceptional precision, with most shot placements correctly classified. Rare misclassifications occurred, likely due to overlap in feature distributions for similar shot placements.

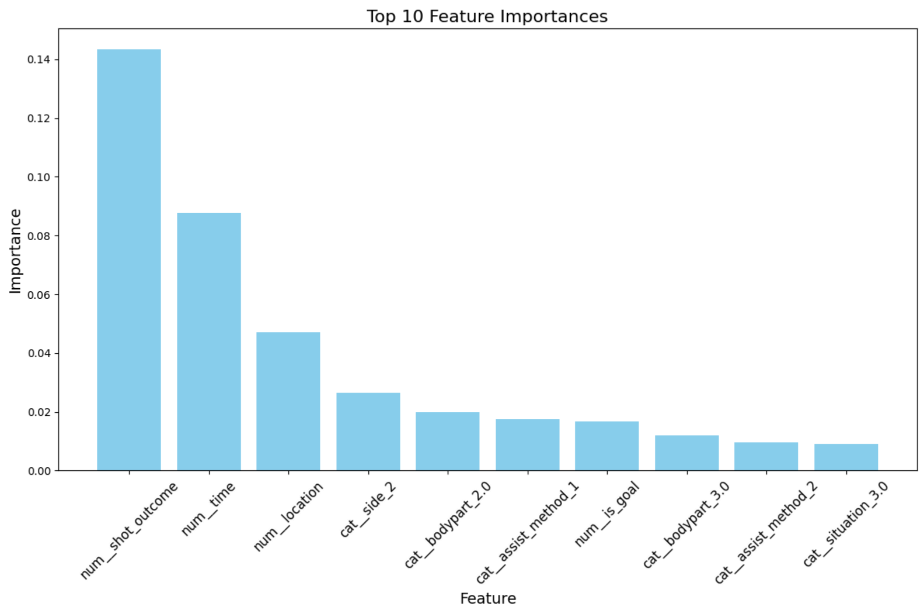


Fig D.9: Top 10 feature importances for supervised baseline

Random Chance Approach

Accuracy: 8.21%
Confusion Matrix - the predictions were uniformly spread across all categories, reflecting the randomness of the approach.

The random chance approach has a lower bound for performance with a small accuracy. This result highlights the importance of structured approaches like OPTICS and supervised methods for meaningful classification. Random chance fails to capture any underlying patterns in the data, and it reflects the total randomness of predictions.

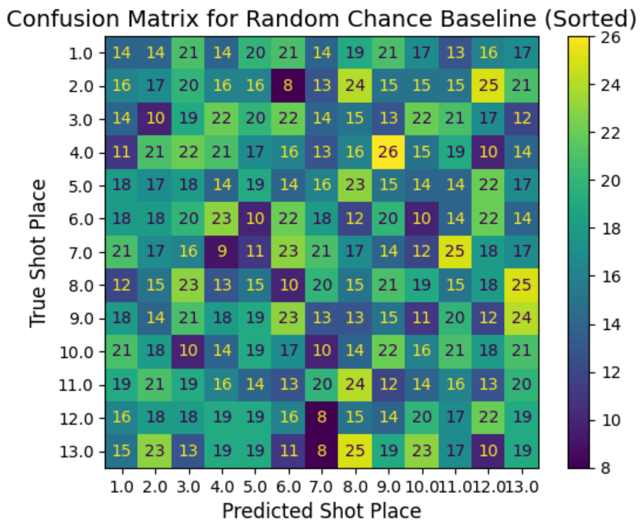


Fig D.10: Confusion Matrix for random chance

E. Third Additional Approach - DBSCAN Clustering

Overview of DBSCAN

DBSCAN (Density-Based Spatial Clustering of Applications with Noise) is a robust clustering algorithm designed to identify dense regions of data points in a feature space. Unlike OPTICS or hierarchical clustering methods, DBSCAN does not require the user to specify the number of clusters beforehand. Instead, it relies on two parameters:

- *eps*: The maximum distance between two points for them to be considered neighbors.
- *min_samples*: The minimum number of points required to form a dense region.

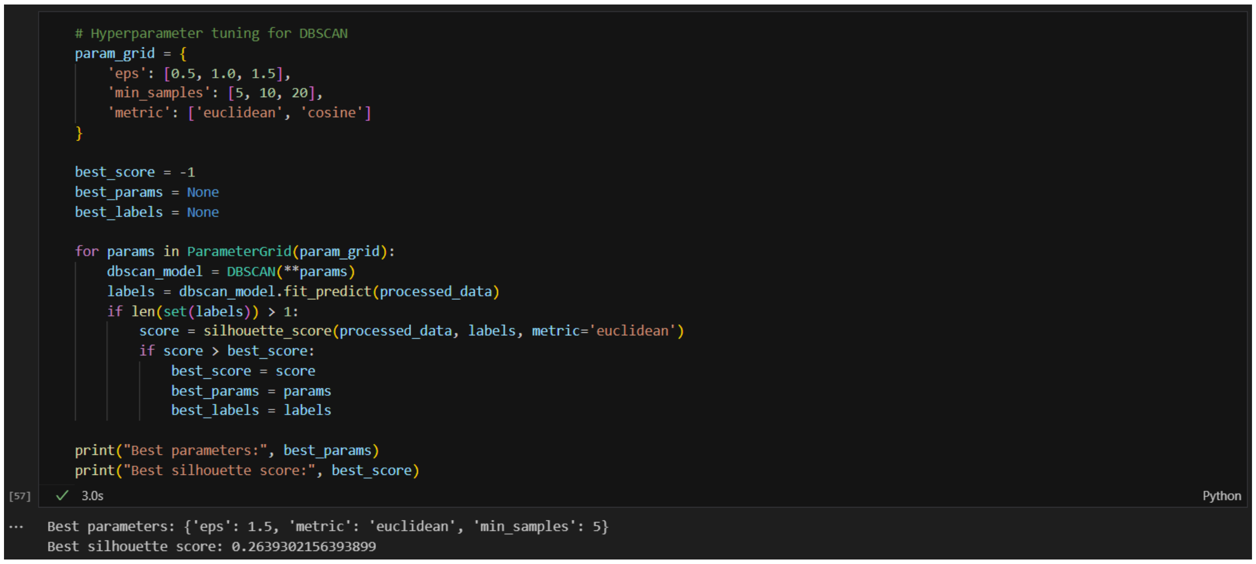
I thought of using DBSCAN due to its ability to identify clusters of arbitrary shape while labeling outliers as noise. It was used on the same task as the previous OPTICS approach, clustering football shots based on multiple feature representations.

Differences

Since the preprocessing was the same, because I wanted to compare its performance with OPTICS, I will start to describe hyperparameter tuning directly and then review the results.

A systematic grid search was conducted to determine the optimal DBSCAN parameters. The hyperparameter combinations tested included:

- *eps*: [0.5, 1.0, 1.5]
- *min_samples*: [5, 10, 20]
- *metric*: ['euclidean', 'cosine']



```python
# Hyperparameter tuning for DBSCAN
param_grid = {
    'eps': [0.5, 1.0, 1.5],
    'min_samples': [5, 10, 20],
    'metric': ['euclidean', 'cosine']
}

best_score = -1
best_params = None
best_labels = None

for params in ParameterGrid(param_grid):
    dbscan_model = DBSCAN(**params)
    labels = dbscan_model.fit_predict(processed_data)
    if len(set(labels)) > 1:
        score = silhouette_score(processed_data, labels, metric='euclidean')
        if score > best_score:
            best_score = score
            best_params = params
            best_labels = labels

print("Best parameters:", best_params)
print("Best silhouette score:", best_score)
```

[57] ✓ 3.0s Python

```
Best parameters: {'eps': 1.5, 'metric': 'euclidean', 'min_samples': 5}
Best silhouette score: 0.2639302156393899
```

Fig E.1: Hyperparameter tuning for DBSCAN

The best parameter combination was: **eps = 1.5, min_samples = 5, metric = euclidean**. The configuration obtained the highest silhouette score of 0.2639, indicating moderate separation between clusters. DBSCAN identified 161 clusters, with noise points labeled as -1. The cluster metrics are:

- Silhouette Score: 0.2639
- Davies-Bouldin Index: 1.0463 (lower values indicate better clustering compactness and separation).
- Calinski-Harabasz Score: 15.0454 (higher values indicate better-defined clusters).

By mapping clusters back to shot_place using majority voting, an accuracy of 54.61% was achieved for predicting shot placement categories.

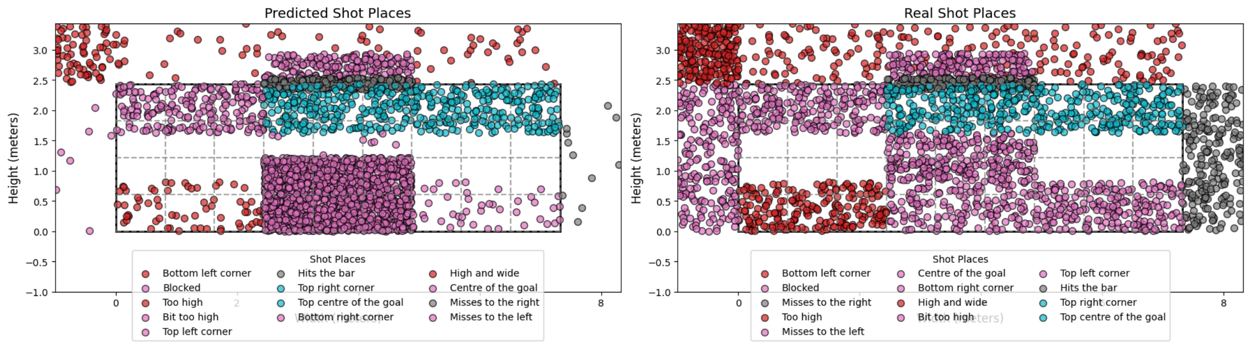


Fig E.2: Predicted vs Real shot places for DBSCAN

In the predicted plot from above, clusters capture distinct regions like the top left and right corners and also the shots marked as “Bit too high”, over the post. However, more discrepancies arise here than for the OPTICS approach. It looks like DBSCAN tends to group a vast number of shots to “Centre of the goal”, especially the ones which are outside the goal, because we can see in the left plot that there is a really small number of outside shots.

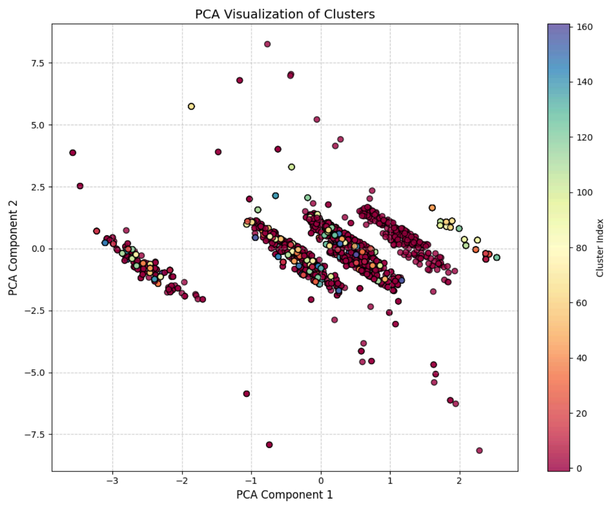


Fig E.3: Visualization of DBSCAN clusters

F. Conclusions

In conclusion, this project was a great opportunity for me to discover and uncover patterns in football data, focusing on team play styles with BIRCH and shot placement with OPTICS or DBSCAN. I found the methods efficient in handling a large dataset and identifying clusters which correspond to different characteristics. For all methods, preprocessing, feature engineering and hyperparameter tuning played an important role in extracting relevant patterns. Finally, by comparing the unsupervised with supervised training, I found the limitations of the clustering methods, but I am happy to understand the domain relevance in analyzing data.