Department of Computer Science Course: CS-505-740

Semester: Fall 2022

Under the Guidance Of: Dr.Juefei Yuan

South East Missouri State University

Group Name: Team_Coders

Team Member	SEMO ID
Sai Gajjala	S02035282
Usha Devulapally	S02035040
Kolli Pradeep Kumar	S02033678
Phanindra Kumar Perumalla	S02036172
Nachi Ketan Reddy	S02034793
Sharath Kumar Polisetty	S02036169

Abstract

Player position in ice hockey is very important for defense and attack. Unlike in other team games in ice hockey players usually play in two lines attack with three players and defense with two. Recently, the NHL has published new data about each game. As new statistical analysis tools such as scalable vector machines become available, it becomes possible to analyze player positions more holistically by using a wide range of statistical methods. The aims of this study were to predict players positions by looking at the statistics of an individual player.

Contents

Chapter 1

Introduction

1.1 Getting started

Brief

Players positions and thus so called line combinations are so important for team to achieve victory that teams have different line combinations of players for different games modes such as Power Play or killing a penalty. The usage of statistics in the NHL has been very rare until recently when Official NHL statistics API and statistics became available. Statistics development in NHL game is hard because players and puck are always in the move. In ice hockey games, there usually four forward lines and one defense, players usually play with linemates throughout a game. The statistics of players at the same line are often highly correlated.

1.1.1 Method

Support Vector Machine or SVM is one of the most popular supervised learning algorithms, which is used for classification and regression problems. The goal of the SVM algorithm is to create the best line or decision boundary that can segregate n-dimensional space into classes so that we can easily put the new data point in the correct category in the future. This best decision boundary is called a hyperplane. SVM chooses the extreme points/vectors that help in creating the hyperplane. These extreme cases are called as support vectors, and hence algorithm is termed as Support Vector Machine.

1.1.2 Data Resource

Data comes from Official NHL statistics API from season 2010 until now. From multiple statistics available following were chosen:

- person.id
- person.positionCode
- person.weight,
- person.height,
- person.shootsCatches,
- skaterStats.timeOnIce,
- skaterStats.assists,
- skaterStats.goals,
- skaterStats.shots,
- skaterStats.hits,
- skaterStats.powerPlayGoals
- skaterStats.powerPlayAssists,
- skaterStats.penaltyMinutes,
- skaterStats.faceOffWins,
- skaterStats.faceoffTaken,
- skaterStats.takeaways,
- skaterStats.giveaways,
- skaterStats.shortHandedGoals,
- skaterStats.shortHandedAssists,
- skaterStats.blocked,
- skaterStats.plusMinus,
- skaterStats.evenTimeOnIce,
- skaterStats.powerPlayTimeOnIce,
- skaterStats.shortHandedTimeOnIce

Chapter 2

Data and Methods

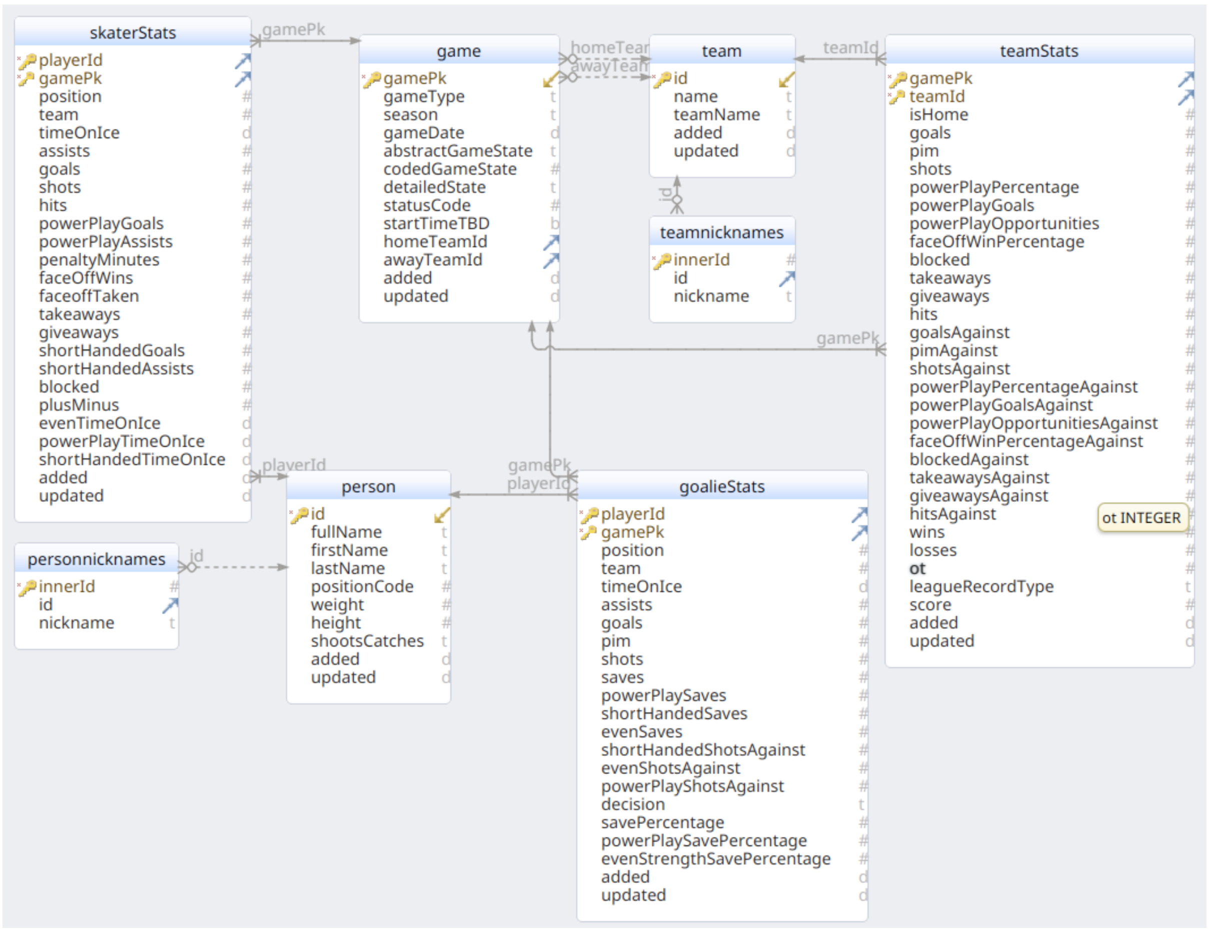
2.1 Data

2.1.1 Data Mining

Data was scraped from Official NHL statistics API using Python libraries requests and grequests which allow asynchronous HTTP requests. Since this is no trivial dataset with relations between game, teams, players and player stats storing it in CSV or other plain file format would be overkill. For that purpose SQL schema for SQlite database was created.

2.1.2 Data storage

Fetched data was stored into SQL database in raw format without any transformations just like returned by statistics API. For that purpose SQL schema was designed to fit data:



2.1.3 Data fetch from database

Thanks to storing data in database features was easy to shape and obtain with single SQL query:

```
SELECT
    person.id,person.positionCode,person.weight,person.height,person.shootsCatches,
skaterStats.timeOnIce,skaterStats.assists,skaterStats.goals,skaterStats.shots,
skaterStats.hits,skaterStats.powerPlayGoals,skaterStats.powerPlayAssists,
skaterStats.penaltyMinutes,skaterStats.faceOffWins,skaterStats.faceoffTaken,
skaterStats.takeaways,skaterStats.giveaways,skaterStats.shortHandedGoals,
skaterStats.shortHandedAssists,skaterStats.blocked,skaterStats.plusMinus,
skaterStats.evenTimeOnIce,skaterStats.powerPlayTimeOnIce,
skaterStats.shortHandedTimeOnIce FROM person,skaterStats WHERE person.positionCode
    != 'G' AND person.id = skaterStats.playerId
```

This SQL query performs join of players personal data like weight, height etc with players game statistics.

2.1.4 Data cleansing and transformation

Variables were classified into continuous variables and binary variables. This process is necessary to normalize variables and convert binary variables to 0's or 1's there is only one binary variable person.shootsCatches which is information about dominant hand.

Although four variables needed to be transformed shootsCatches and time statistics which were obtained as text and needed to be converted integer value. The example of time conversion from string to integer is presented below:

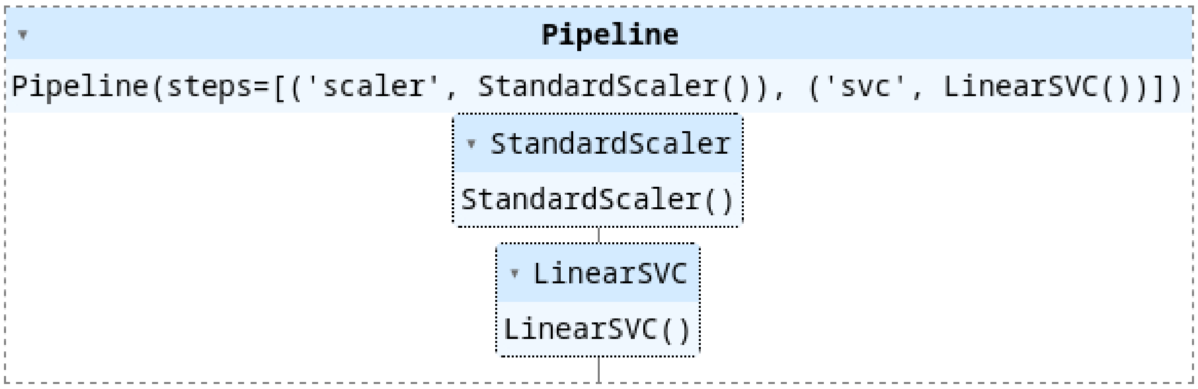
```
nhl['timeOnIce'] = nhl['timeOnIce'].str.split('.').str[0]
nhl['timeOnIce'] = nhl['timeOnIce'].str.split(':').apply(lambda x: int(x[0]) * 60 +
    int(x[1]))
```

shootsCatches representing a dominant hand was converted using python pandas python library function factorize which transformed this variable to 0,1:

```
nhl.shootsCatches = pd.Categorical(pd.factorize(nhl.shootsCatches)[0])
```

2.1.5 Data normalization

Data was normalized prior to using Linear Support Vector Classification, by using standard scaler which scales data to range of [0,1]. This is important because SVC tries to maximize the distance between the separating plane and the support vectors. If one feature (i.e. one dimension in this space) has very large values, it will dominate the other features when calculating the distance. If data is rescaled all features have the same influence on the distance metric. This was done by making a pipeline with standard scaler and SVC classifier.



```
Pipeline(steps=[('scaler', StandardScaler()), ('svc', LinearSVC())])
StandardScaler()
LinearSVC()
```

2.1.6 Testing data

Variance Threshold is a feature selector that removes all the low variance features from the dataset that are of no great use in modeling. It looks only at the features, not the desired outputs, and can thus be used for unsupervised learning. In other words it's used to find constant/quasi-constant predictors.

```python
from sklearn.feature_selection import VarianceThreshold
selector = VarianceThreshold()
selector.fit_transform(kb_features)
selector.get_feature_names_out()

array(['id', 'weight', 'height', 'shootsCatches', 'timeOnIce', 'assists',
       'goals', 'shots', 'hits', 'powerPlayGoals', 'powerPlayAssists',
       'penaltyMinutes', 'faceOffWins', 'faceoffTaken', 'takeaways',
       'giveaways', 'shortHandedGoals', 'shortHandedAssists', 'blocked',
       'plusMinus', 'evenTimeOnIce', 'powerPlayTimeOnIce',
       'shortHandedTimeOnIce'], dtype=object)
```

From the output of Variance Threshold it can be seen that there is no static or quasi-static predictors and all of the data is equally important.

2.1.7 Data split

Before attempting to use SVC data was splitted into train and test sets. With 75% of data being set as train data and 25% as test data.

```python
X_train, X_test, Y_train, Y_test = train_test_split(features, targets,
↪  test_size=0.25,random_state=0,shuffle=True)
```

2.2 Method

After preparing the data linear SVC was used on training set to prepare fit coefficients that would be then used on test set.

```python
import warnings
warnings.filterwarnings("ignore")
from sklearn.model_selection import cross_val_score
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn.svm import LinearSVC
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler

pipe = Pipeline([('scaler', StandardScaler()), ('svc', LinearSVC())])
pipe.fit(X_train, Y_train)
print("Accuracy {:.2f}%".format(pipe.score(X_test, Y_test)*100))
```

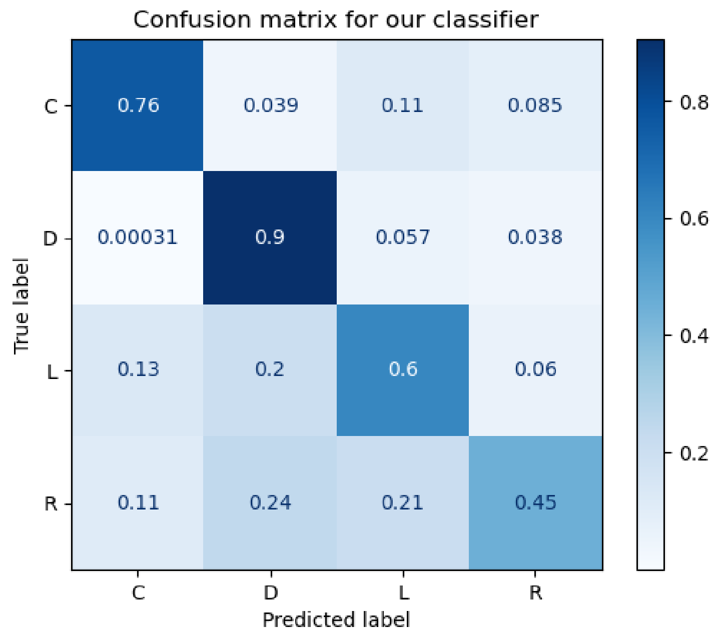
Which gave the accuracy of 73.52%

Chapter 3

Results and Discussion

3.1 Results

Even with not bad accuracy of 73.52% at first attempt it was worth to check confusion matrix to target low accuracy classification results.



- the highest accuracy is for defence (D) players it's 90% there would probably be not much space for optimization here
- center position (C) has 76% accuracy it can be optimized a little to achieve better accuracy
- the two worst cases are left wing (L) and right wing (R) players for whom accuracy is 60% and 45%

3.2 Discussion

- what stands opposite to domain knowledge is that there is about 20% probability of classifying wing players as defence players, stats of players from one line should be similar and thus probability of false positives should appear between those positions
- it's worth checking whether model could be optimized by dropping some of the player data from training set, especially the ones that for example have time on ice lower then average of other players at that position

- from the model it follows that defence players statistics vary significantly from centre and wing players thus accuracy for defence is very high

Chapter 4

Conclusion and References

4.1 Conclusion

- Sophisticated model would help NHL teams to figure out which players they should acquire and sell to keep necessary backup for line positions,
- NHL teams could use this model to figure out which players should play on which position to unleash their true potential based on statistics,
- According to Variance Treshold all of the columns are important for position classification,
- Defence position classification is almost certainly true,

4.2 References

- Matsuzawa, Takehiro. 2017. Using Machine Learning to Predict Future Points in the NHL
- Matt Eland, Predicting Hockey Penalties with Azure Machine Learning