sena_ap.py

Scanned on: 22:49 November 10, 2022 UTC

Overall Similarity Score

Results Found

Total Words in Text

Identical Words	286
Words with Minor Changes	16
Paraphrased Words	1
Omitted Words	0

sena_ap.py

Scanned on: 22:49 November 10, 2022 UTC

Results

Sources that matched your submitted document.

Sales-Copy1 *https://courses.grainger.illinois.edu/cs357/fa2019/assets/demos/upload/1...* Sales-Copy1.html	6%
| notebook.community *https://notebook.community/infilect/ml-course1/ml-notebooks/clustering*	6%
How to create clusters using DBSCAN in Python - Thinking N... *https://thinkingneuron.com/how-to-create-clusters-using-dbscan-in-python/*	0%

IDENTICAL

Identical matches are one to one exact wording in the text.

MINOR CHANGES

Nearly identical with different form, ie "slow" becomes "slowly".

PARAPHRASED

Close meaning but different words used to convey the same message.

Unsure about your report?

The results have been found after comparing your submitted text to online sources, open databases and the Copyleaks internal database. For any questions about the report contact us on support@copyleaks.com

Learn more about different kinds of plagiarism here

sena_ap.py

Scanned on: 22:49 November 10, 2022 UTC

Scanned Text

Your text is highlighted according to the matched content in the results above.

IDENTICAL MINOR CHANGES PARAPHRASED

```
# -*- coding: utf-8 -*-
"""SENA - AP.ipynb

Automatically generated by Colaboratory.

Original file is located at
https://colab.research.google.com/drive/1px8pogTV5KEzKpLHOkQkVf3cvGOEtcqT

# SENA - Assignment presentation

# Basic Imports
"""

# Commented out IPython magic to ensure Python compatibility.
import numpy as np
import pandas as pd
import seaborn as sb
import matplotlib.pyplot as plt
from matplotlib import gridspec
from IPython.display import display, HTML
from math import pi
plt.style.use('fivethirtyeight')

# %matplotlib inline

from sklearn.preprocessing import StandardScaler, PolynomialFeatures
from sklearn.model_selection import train_test_split, cross_val_score, GridSearchCV
from sklearn import linear_model, tree, svm
from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn.metrics import mean_squared_error, r2_score, accuracy_score
from sklearn.neighbors import KNeighborsClassifier

"""# Loading the dataset"""

from google.colab import drive

drive.mount('/content/drive')

DATA_PATH1 = '/content/fifa_data_2015_to_2020.csv'
DATA_PATH2 = '/content/players_20.csv'
```

```
df= pd.read_csv(DATA_PATH1, sep=',')
data= pd.read_csv(DATA_PATH2, sep=',')
display(df.head(5))

"""# Analysis 1 - EDA"""

df.columns

def get_year(df,year):
temp_data = df.query('year==@year').reset_index()
return temp_data

df_2015 = get_year(df,2015)
df_2016 = get_year(df,2016)
df_2017 = get_year(df,2017)
df_2018 = get_year(df,2018)
df_2019 = get_year(df,2019)
df_2020 = get_year(df,2020)
l_df=[df_2015,df_2016,df_2017,df_2018,df_2019,df_2020]

"""EDA 1 - Overall rating distribution by year"""

plt.figure(figsize = (20,7))
sb.countplot(data = df, x = 'overall', hue='year', palette = sb.cubehelix_palette(6, start=5, rot=0, dark=0.2,
light=.8, reverse=False))
plt.title('Overall rating distribution by year')
plt.legend(loc=1);

"""EDA 2 - Comparison of the top 5 players from 2015 to 2020"""

l=[]
for i in l_df:
temp = i.head(5)[['short_name','overall','year']]
l.append(temp)
fig = plt.figure(figsize=(22,12))
plt.suptitle('Comparaison of the top 5 players from 2015 to 2020',fontsize=22)
plt.tight_layout(pad=0.5, w_pad=0.5, h_pad=0.5)
plt.subplot(231)
sb.barplot(data=l[0], x='short_name', y='overall', palette='Reds_r')
plt.ylim(85, 100)
plt.xlabel('Player Name')
plt.title('Top 5 most rated players in FIFA 2015')
plt.subplot(232)
sb.barplot(data=l[1], x='short_name', y='overall', palette='Reds_r')
plt.ylim(85, 100)
plt.xlabel('Player Name')
plt.title('Top 5 most rated players in FIFA 2016')
plt.subplot(233)
sb.barplot(data=l[2], x='short_name', y='overall', palette='Reds_r')
plt.ylim(85, 100)
plt.xlabel('Player Name')
plt.title('Top 5 most rated players in FIFA 2017')
plt.subplot(234)
sb.barplot(data=l[3], x='short_name', y='overall', palette='Reds_r')
plt.ylim(85, 100)
plt.xlabel('Player Name')
plt.title('Top 5 most rated players in FIFA 2018')
plt.subplot(235)
sb.barplot(data=l[4], x='short_name', y='overall', palette='Reds_r')
plt.ylim(85, 100)
plt.xlabel('Player Name')
plt.title('Top 5 most rated players in FIFA 2019')
plt.subplot(236)
sb.barplot(data=l[5], x='short_name', y='overall', palette='Reds_r')
plt.ylim(85, 100)
plt.xlabel('Player Name')
```

```
plt.title('Top 5 most rated players in FIFA 2020');

"""EDA 3 - Best overall rating for each position by year"""

def get_player_noGK(df):
idx_df_gk = list(df.query('pace==0').index)
df_no_GK = df.drop(idx_df_gk)
return df_no_GK

def get_best_by_position(df):
lis=[]
temp = df.groupby(['team_position'])[['overall']].max()
for i in list(temp.index):
overall = temp.loc[i][0]
best_i = df.query('team_position==@i & overall==@overall').iloc[0]
lis.append(best_i)
best_pos = pd.DataFrame(lis)
return best_pos

lis=[]
lis.append(get_best_by_position(df_2015))
lis.append(get_best_by_position(df_2016))
lis.append(get_best_by_position(df_2017))
lis.append(get_best_by_position(df_2018))
lis.append(get_best_by_position(df_2019))
lis.append(get_best_by_position(df_2020))
best_pos = pd.concat(lis, ignore_index=True)

plt.figure(figsize=(15,6))
plt.title('Best overall rating for each position by year')
sb.barplot(data=best_pos, x='team_position', y='overall', hue='year')
plt.legend(loc='upper right', bbox_to_anchor=(1.1, 1))
plt.xlabel('Team position')
plt.ylabel('Overall')
plt.ylim(60,100);

"""EDA 4 - Best Players in every position in each year"""

gb = best_pos.groupby('team_position')
l = [gb.get_group(x) for x in gb.groups]
for i in range(15):
display(l[i].loc[:,['short_name','team_position','overall','club','nationality','year']])

"""# Analysis 2 - Player overall rating/brand value prediction"""

# Construction of the features dataframe for the regression model

years=[2019,2018,2017,2016,2015]
features_selected = ['age','wage_eur', 'value_eur','international_reputation', 'pace','skill_moves' ,'shooting',
'passing', 'dribbling',
'defending', 'physic','skill_oa', 'attack_oa', 'movements_oa', 'power_oa', 'mentality_oa', 'gk_oa',
'defending_oa', 'trait_coef','year']

ols_xdata = df.query('year in @years')[features_selected].copy()
ols_ydata = df.query('year in @years')['overall'].copy()

ols_xdata.loc[ols_xdata.value_eur==0,'value_eur'] = ols_xdata['value_eur'].mean()
ols_xdata.loc[ols_xdata.wage_eur==0,'wage_eur'] = ols_xdata['wage_eur'].mean()

ols_xdata['isGK'] = (ols_xdata['pace']==0).astype(int)
cols = list(ols_xdata.columns)
ols_xdata

# Adding 2nd degree polynomial features to make the regression model more accurate

polynomial = PolynomialFeatures(2, include_bias=False)
```

```
ols_xdata = polynomial.fit_transform(ols_xdata)

# Train/Test data split

#xtrain, xtest, ytrain, ytest = train_test_split( ols_xdata, ols_ydata, test_size=0.2, shuffle=True,
random_state=42)
xtrain,ytrain=ols_xdata,ols_ydata

display((xtrain.shape, ytrain.shape))

x_2020 = df.query('year ==2020')[features_selected].copy()
x_2020['isGK'] = (x_2020['pace']==0).astype(int)
x_2020 = polynomial.fit_transform(x_2020)
y_2020 = np.array(df.query('year==2020')['overall'].copy()).flatten()
regression = linear_model.LinearRegression()
regression.fit(xtrain, ytrain)
ypred = regression.predict(x_2020)
predicted_2020 = np.round(regression.predict(x_2020)).astype(int)

print('Mean squared error: %.2f' % mean_squared_error(y_2020, predicted_2020))
print('Coefficient of determination: %.2f'% r2_score(y_2020, predicted_2020))

# Choosing a player from the dataset

def get_player(df,features_selected):
player= df.sample(1)
player_x = player[features_selected].copy()
player_x['isGK'] = (player_x['pace']==0).astype(int)
player_x= polynomial.fit_transform(player_x)
player_y = np.array(player['overall'].copy())
return player_x,player_y

#Predicting the overall rating of the random player
player_x,player_y = get_player(df, features_selected)
estimated_y = regression.predict(player_x)
print('Actual value y=',int(player_y), 'Predicted value=', int(np.round(estimated_y)))

plt.figure(figsize=(10,10))
sb.scatterplot(predicted_2020, y_2020, alpha=0.15)
plt.plot(y_2020, y_2020, color='r', alpha=0.5)
plt.legend(['Correct pred','True-Vs-Pred scatter'])
plt.title('Prediction of overall ratings')
plt.xlabel(' Overall Prediction')
plt.ylabel(' Overall True');

"""# Analysis 3 - Community detection or Clustering with respect to Player position

## Preprocessing
"""

display(data.head())
print(f'The Data has {data.shape[0]} rows and {data.shape[1]} features')

"""## Data understanding and cleaning"""

# Taking the first preference of each player's preferred player position
data['player_positions'] = data['player_positions'].str.split(',').str[0]

# Making a copy of the original dataset before further analysis
copy_of_dataset = data.copy()

# function to generate samples of the desired player positions
def sample_generate(positions = ['CAM', 'RM', 'CDM', 'LM', 'CM'], n_samples = 10):
'''
available unique player positions to be clustered = ['RW', 'ST', 'LW', 'GK', 'CAM', 'CB', 'CM', 'CDM', 'CF', 'LB',
'RB','RM', 'LM', 'LWB', 'RWB']
```

```
'''
samples = copy_of_dataset[copy_of_dataset.player_positions.isin(positions) &
(copy_of_dataset.overall>=70)].sample(n_samples)
return samples.index.values

"""Player Ratings"""

plt.figure(figsize=(20,10))
ax = sb.distplot(data.overall, bins=20);
ax.set_title('Distributions of Player Ratings')
ax.set_xlabel('Overall Ratings');

data = data[data.overall>=70] # Removing players with rating below 70

pd.DataFrame(data.overall.value_counts().sort_index())

# Visualizing updated player ratings after removal
plt.figure(figsize=(20,10))
ax = sb.distplot(data.overall, bins=20,vertical=False,kde=False);
ax.set_title('Distributions of Player Ratings')
ax.set_xlabel('Overall Ratings');

# Plotting Age Vs Player rating
plt.figure(figsize=(20,10))
ax = sb.scatterplot('age','overall',hue='player_positions',data=data);
ax.set_title('Player Age vs Overall Rating')
ax.set_xlabel('Age')
ax.set_ylabel('Overall Rating')

def label_point(x, y, val, ax):
a = pd.concat({'x': x, 'y': y, 'val': val}, axis=1)
for i, point in a.iterrows():
if (point['y'] >=90) :
ax.text(point['x']+.1, point['y']+.1, str(point['val']),fontsize=12)

label_point( data.age,data.overall, data.short_name, plt.gca())

"""Dropping unecessary features"""

to_drop = ['sofifa_id','player_url','long_name','potential','dob',\
'work_rate','body_type','real_face','release_clause_eur','player_tags',\
'team_position','team_jersey_number','loaned_from','joined','contract_valid_until',\
'nation_position','nation_jersey_number','player_traits',\
'ls','st','rs','lw','lf','cf','rf','rw','lam','cam','ram','lm','lcm','cm','rcm',\
'rm','lwb','ldm','cdm','rdm','rwb','lb','lcb','cb','rcb','rb', 'value_eur','wage_eur']
data = data.drop(to_drop, axis=1)

pd.DataFrame(data.dtypes).T

"""Checking for correlated features using a heatmap"""

# Create correlation matrix as the data is still suspected to be highly correlated
corr_matrix = data.corr().abs()

# Plotting a heat map to detect correlated features
plt.figure(figsize=(20,10))
ax = sb.heatmap(corr_matrix)

"""Goalkeeper features are correlated"""

# The white squares in the heatmap show that the features related to goalkeepers are correlated
perfectly.
feature_goalkeep = ['gk_handling','gk_reflexes','gk_positioning','gk_diving','gk_kicking','gk_speed',\
'goalkeeping_diving','goalkeeping_handling','goalkeeping_kicking','goalkeeping_positioning','goalkeeping_re
flexes']
data = data.drop(feature_goalkeep, axis = 1)
```

```
# Goalkeepers form a isolated group because no main player skills apply to them. Thus, goalkeepers are removed from the dataset as we have a clear indication of them being a separate cluster.

data = data[data.player_positions !='GK']

"""Drop categorical features"""

# All the features that are categorical are dropped as PCA needs continuous features only.
feature_category =
['short_name','nationality','club','preferred_foot','player_positions','international_reputation','weak_foot','skill_moves']
data = data.drop(feature_category, axis =1)
data = data.fillna(0)
data.shape

"""Building a Decision Tree Regressor to eliminate more correlated features using R2 score"""

def features(data, feature, random_state, R2score): #regression model is built on remaining features acting as the response and all other features act as predictors.
new = data.drop(feature, axis = 1)

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(new, data[feature], test_size = 0.20, random_state = random_state)

from sklearn.tree import DecisionTreeRegressor #Importing a decision tree regressor from Skikitlearn
regressor = DecisionTreeRegressor(random_state=random_state)
regressor.fit(X_train, y_train)
y_pred = regressor.predict(X_test)

score = regressor.score(X_test, y_test)
if score >= 0.95: #The R2 scores for response > 0.95 is considered as the variance in these features can be explained by the remaining variables and they do not add a lot of further information to our analysis.
R2score.append(feature)
print("R2 Score for feature {} is {}".format(feature, round(score,3) ))
else:
pass
return R2score

R2score = []
for key in data:
features(data, key, random_state=13263600,R2score=R2score)

"""Dropping features with R2 score >= 0.95"""

data = data.drop(R2score, axis = 1) #Since the R2 scores are so high, these features can be calculated using the other independent features, thus drop them.
print(data.shape)
data.head()

"""## Feature Scaling and Transforming the data for PCA

Tweaked Dataset distribution
"""

data.describe()

"""We perform log scaling as it gave the best results from the rest"""

from sklearn.decomposition import PCA
def results_of_pca(data, n_components=8):
#PCA model
model_pca = PCA(n_components=n_components, random_state=1).fit(data)

# Creating a dataframe
dim = ['Dimension {}'.format(i) for i in range(1,len(model_pca.components_)+1)]
```

```
comp = pd.DataFrame(np.round(model_pca.components_, 4), columns = list(data.columns))
comp.index = dim

# PCA explained variance
ratios = model_pca.explained_variance_ratio_.reshape(len(model_pca.components_), 1)
ratio_variance = pd.DataFrame(np.round(ratios, 4), columns = ['Explained Variance'])
ratio_variance.index = dim

# Bar plot visualization
fig, ax = plt.subplots(figsize = (25,10))

# Plot the feature weights as a function of the components
comp.plot(ax = ax, kind = 'bar');
ax.set_ylabel("Feature Weights")
ax.set_xticklabels(dim, rotation=0)
plt.legend(loc='upper right')

# Display the explained variance ratios
for i, ev in enumerate(model_pca.explained_variance_ratio_):
ax.text(i-0.40, ax.get_ylim()[1] + 0.05, "Explained Variance\n%.4f"%(ev))

# Return a concatenated DataFrame
df = pd.concat([ratio_variance, comp], axis = 1)
print(f'Total Variance Explained by the first 2 dimensions: {df.iloc[:2,0].sum()}')
return df

logscaled_data = np.log(data)

def transformed_data(data):
plt.figure(figsize=(20,6))
for col in data.columns:
sb.kdeplot(data[col], shade=True)
plt.legend(loc='best');

transformed_data(data) #Unscaled data
# We can notice that the features are slightly left-skewed.

transformed_data(logscaled_data) #Scaled data using Log Scaling

pca_logscale = results_of_pca(logscaled_data, 5) #For every feature, the variance for every dimension is
calculated. An average of this variance is taken finally.

"""## PCA Analysis

Making a Biplot for PCA analysis
"""

def construct_pca(data, sample_ids):
pca = PCA(n_components=2).fit(data) #pca object that contains the components_ attribute
data_reduced = pca.transform(logscaled_data) ## the first two dimensions are plotted
samples_of_pca = pca.transform(logscaled_data[logscaled_data.index.isin(sample_ids)])
data_reduced = pd.DataFrame(data_reduced, columns = ['Dimension 1', 'Dimension 2'])
return pca, data_reduced, samples_of_pca

pca, data_reduced, samples_of_pca = construct_pca(logscaled_data, sample_ids =
sample_generate(['CAM','CM']))

def biplot(log_data, data_reduced, pca): #Biplot shows the scatterplot of the reduced data and the original
features' projections.

fig, ax = plt.subplots(figsize = (10,10))
# scatterplot of the reduced data
ax.scatter(x=data_reduced.loc[:, 'Dimension 1'], y=data_reduced.loc[:, 'Dimension 2'],
facecolors='b', edgecolors='b', s=70, alpha=0.5)

feature_vectors = pca.components_.T
```

```
# To see arrows and text easily
arrow_size, text_pos = 4.0, 3.0,

# Original features projections
for i, v in enumerate(feature_vectors):
ax.arrow(0, 0, arrow_size*v[0], arrow_size*v[1],
head_width=0.2, head_length=0.2, linewidth=2, color='red')
ax.text(v[0]*text_pos, v[1]*text_pos, log_data.columns[i], color='black',
ha='center', va='center', fontsize=18)

ax.set_xlabel("Dimension 1 (PC1)", fontsize=14)
ax.set_ylabel("Dimension 2 (PC2)", fontsize=14)
ax.set_title("PC plane with original feature projections.", fontsize=16);
return ax

biplot(logscaled_data, data_reduced, pca);

"""## Clustering

Silhouette score to determine the no. of clusters
"""

from sklearn.metrics import silhouette_score
from sklearn.cluster import KMeans

def clustering(reduced_data,n_clusters,pca_samples=samples_of_pca):
clusterer = KMeans(n_clusters=n_clusters, random_state=123).fit(reduced_data)
preds = clusterer.predict(reduced_data)
centers = clusterer.cluster_centers_
sample_preds = clusterer.predict(pca_samples)
return preds, centers, sample_preds

def score_silhouette(reduced_data,n_clusters):
preds,_,_ = clustering(reduced_data,n_clusters)
score = silhouette_score(reduced_data, preds)
return score

for n_clusters in range(2,10):
score = score_silhouette(data_reduced,n_clusters)
print ("Silhoutte Score for {} cluster is {}".format(n_clusters,score))

"""The highest Silhouette Score is 0.53 for 3 clusters

Alternate Elbow method to to determine the no. of clusters
"""

inertia = []
clusters = range(2,10)
for n_clusters in clusters:
clusterer = KMeans(n_clusters=n_clusters, random_state=123).fit(data_reduced)
preds = clusterer.predict(data_reduced)
inertia.append(clusterer.inertia_)

plt.plot(clusters, inertia)
plt.ylabel('Inertia')
plt.xlabel('No. of clusters')
plt.title('Elbow Method');

"""The value of K is derived from the elbow of the curve plotted above.

The elbow of the curve appears at **3 clusters** which coincides with the Silhouette score too.

## Visualizing the clusters
"""
```

```
def visualize_cluster_results(reduced_data, preds, centers, pca_samples):

import matplotlib.cm as cm
predictions = pd.DataFrame(preds, columns = ['Cluster'])
plot_data = pd.concat([predictions, reduced_data], axis = 1)

# Generating cluster plot
fig, ax = plt.subplots(figsize = (14,8))

# Color map
colormap = cm.get_cmap('gist_rainbow')

# Color the points according to the assigned cluster
for i, cluster in plot_data.groupby('Cluster'):
cluster.plot(ax = ax, kind = 'scatter', x = 'Dimension 1', y = 'Dimension 2', \
color = colormap((i)*1.0/(len(centers)-1)), label = 'Cluster %i'%(i), s=30);

# Indicate the cluster using indicators
for i, c in enumerate(centers):
ax.scatter(x = c[0], y = c[1], color = 'white', edgecolors = 'black', \
alpha = 1, linewidth = 2, marker = 'o', s=200);
ax.scatter(x = c[0], y = c[1], marker='$%d$'%(i), alpha = 1, s=100);

# Plotting transformed sample data points
ax.scatter(x = pca_samples[:,0], y = pca_samples[:,1], \
s = 150, linewidth = 4, color = 'black', marker = 'x');

# Plot title
ax.set_title("Clustering on PCA-Reduced Data");

copy_of_dataset.player_positions.unique()

"""## Results"""

sample_ids = sample_generate(['RW', 'ST', 'LW', 'GK', 'CAM', 'CB', 'CM', 'CDM', 'CF', 'LB', 'RB','RM', 'LM', 'LWB',
'RWB'], 10)
# Using the general unique player positions, the players are classified into generalized communities, i.e,
into 3 main clusters
_, _, pca_samples = construct_pca(logscaled_data, sample_ids)
preds, centers, sample_preds = clustering(data_reduced, 3)
visualize_cluster_results(data_reduced, preds, centers, pca_samples)

"""## Final Analysis

We should revisit the Biplot image from before in order to comprehend why K Means returned the
clusters that it did.
It is clear when comparing the clusters and the feature vectors after the Biplot transfers the original
features as vectors to the principal components.


Let's look at Cluster 0, also seen as the Red cluster, as an illustration. On the basis of the random samples,
we have deduced that there are several defenders. Now, if we look at the Biplot, we can see that some of
the feature vectors, such as defending_marking, mentality_interceptions, defending_sliding_tackle, have a
significant impact along the direction of this cluster.

Similar to cluster 1, we can see that skill_long_passing, short_passing, power_stamina, mental_composure
etc. are the most crucial characteristics. As a result, we can conclude that midfielders are the players that
exhibit these qualities, and we can pinpoint the individuals in this group who are most suited for a
midfielder position.

Finally, a forward player's primary responsibility is to score goals, so it goes without saying that the crucial
characteristics in this regard are attacking, volleys, mentality_positioning, attacking_header_accuracy are a
few of the important ones.
"""
```