

|  |
| --- |
| **Video Games Sales** |
| Teacher: Azamat Kibekbaev |
| Done by com16:  Zarlyk Zhusubaliev  Azamatzhon Akhunzhanov |
|  |





|  |  |  |
| --- | --- | --- |
| Content |  | • • • |
|  |
|  |
| For our Data Mining project we choose Video Game Sales. This data gives information about video games, which was published from 1980 till nowadays. It consists 11 columns, exactly:   * Rank (int) * Name (Polynomial-String) * Platform (Polynomial-String) * Year (int) * Genre (Polynomial-String) * Publisher (Polynomial-String) * North America sales (Float) * Europe sales (Float) * Japan Sales (Float) * Other Sales (Float) * Global Sales (Float)   We want use Decision Tree Regression and K neighbors Regression for ('Rank', 'Genre', 'Platform', 'Year', 'Publisher') and predict one of (‘’NA\_Sales, 'EU\_Sales','JP\_Sales', and 'Other\_Sales') |

|  |  |  |
| --- | --- | --- |
| (We build all charts in python PL)  Literature Review |  | • • •  Platform |
| First diagram shows number of games in different platforms.In second Diagram we can see Global sale in platfowms in millions. |
|  |
|  |

|  |  |  |
| --- | --- | --- |
| Literature Review |  | Genre • • • |
| Here Collected Information about top genres. Sales of top game genres, and top popular genres. |
|  |
|  |

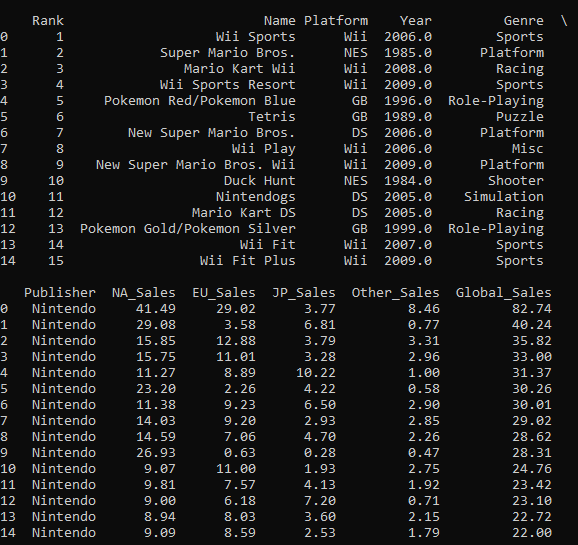
|  |  |  |
| --- | --- | --- |
| Literature Review |  | Sales • • • |
| Sales of Genres according to regions, in bottom percentage, at top in billions.. |
|  |
|  |

|  |  |  |
| --- | --- | --- |
| Literature Review |  | Sales • • • |
| Publishers Global sales |
|  |
|  |

Methodology

* A predictive model was built to estimate the value of the result of interest (namely, “EU\_Sales”)
* Such models can be used by manufacturers and suppliers to estimate the number of copies of games that will be produced and stored accordingly, and what the profit from sales may be

display(data[:15])



**Processing Data**

To simplify the analysis process, records whose ‘Year’ value is missing have been removed from the data frame.

data = data[np.isfinite(data['Year'])]

**Setting our Y-value (to-be-predicted) and our X-value (features)**

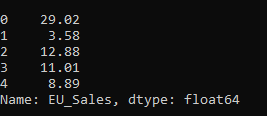
In the next block of code, I set our “y-value” column under the name of the variable “euSales”. We are interested in predicting this value. In addition, I set the “x-value” columns under the name of the “features” variable. We will use these functions to predict our euSales values. The “features” variable will store the following data columns: “Rank”, “Genre”, “Platform”, “Year”, “Publisher”, “NA\_Sales” and “JP\_Sales”, “Other\_Sales”. I do not include the “Global\_Sales” column in the “functions”, since its inclusion will reduce our forecasting problem of euSales to the simple problem of subtraction. euSales will simply be “Global\_Sales” - “NA\_Sales” - “JP\_Sales” - “Other\_Sales”.

euSales = data['EU\_Sales']

features = data.drop(['Name', 'Global\_Sales', 'EU\_Sales'], axis = 1)

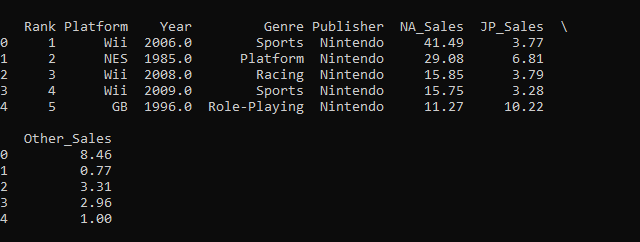
*# Display our features*

display(euSales[:5])



*# Display our target columns*

display(features[:5])



**Principal Component Analysis**

“NA\_Sales”, “JP\_Sales”, “Other\_Sales” are likely to be observable, controlled by a hidden function. In this document, I perform an analysis of the main components of these three functions in order to obtain one hidden function.

*# Firstly, I am dividing the features data set into two as follows.*

salesFeatures = features.drop(['Rank', 'Platform', 'Year', 'Genre', 'Publisher'],

axis = 1)

otherFeatures = features.drop(['NA\_Sales', 'JP\_Sales', 'Other\_Sales', 'Rank'],

axis = 1)

*# Secondly, I am obtaining the PCA transformed features...*

from sklearn.decomposition import PCA

pca = PCA(n\_components = 1)

pca.fit(salesFeatures)

salesFeaturesTransformed = pca.transform(salesFeatures)

*# Finally, I am merging the new transfomed salesFeatures*

*# (...cont) column back together with the otherFeatures columns...*

salesFeaturesTransformed = pd.DataFrame(data = salesFeaturesTransformed,

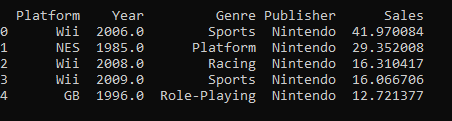
index = salesFeatures.index,

columns = ['Sales'])

rebuiltFeatures = pd.concat([otherFeatures, salesFeaturesTransformed],

axis = 1)

display(rebuiltFeatures[:5])



**Processing our data**

Most machine learning models expect numeric values. The next block of code converts non-numeric values to numeric values, adding columns of dummy variables. For example, the “Genre” function with, say, 2 values, namely “a” and “b”, will be divided into 2 functions: “Genre\_a” and “Genre\_b”, each of which will take binary values.

temp = pd.DataFrame(index = rebuiltFeatures.index)

for col, col\_data **in** rebuiltFeatures.iteritems():

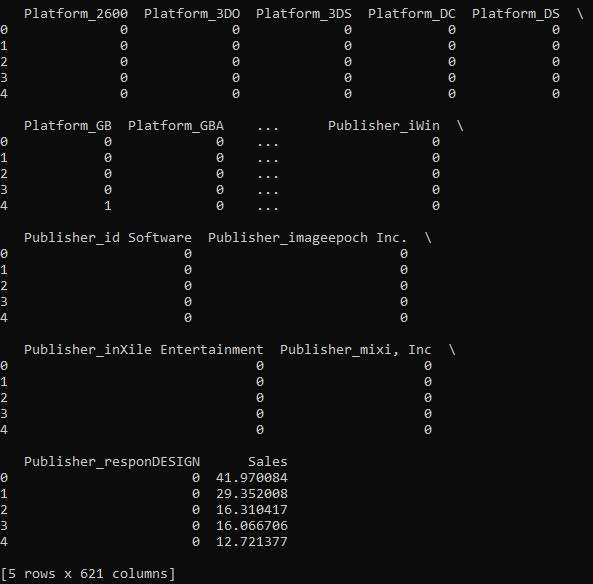
if col\_data.dtype == object:

col\_data = pd.get\_dummies(col\_data, prefix = col)

temp = temp.join(col\_data)

rebuiltFeatures = temp

display(rebuiltFeatures[:5])



**Dividing our data into Training and Testing sets.**

*# Dividing our data to training and testing sets...*

from sklearn.model\_selection import train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(rebuiltFeatures,

euSales,

test\_size = 0.2,

random\_state = 2)

**Model Selection**

We believe that the regression of the decision tree and the regression of the neighbors will correspond well to the data. Here we build both of these models and analyze the results to establish the best of the two. The metric we use to define the “perfection” of a model is the R-square.

*# Creating & fitting a Decision Tree Regressor*

from sklearn.tree import DecisionTreeRegressor

regDTR = DecisionTreeRegressor(random\_state = 4)

regDTR.fit(X\_train, y\_train)

y\_regDTR = regDTR.predict(X\_test)

from sklearn.metrics import r2\_score

print ('The following is the r2\_score on the DTR model...')

print (r2\_score(y\_test, y\_regDTR))

*# Creating a K Neighbors Regressor*

from sklearn.neighbors import KNeighborsRegressor

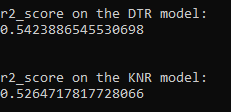
regKNR = KNeighborsRegressor()

regKNR.fit(X\_train, y\_train)

y\_regKNR = regKNR.predict(X\_test)

print ('The following is the r2\_score on the KNR model...')

print (r2\_score(y\_test, y\_regKNR))



The results above show that the regression model of the decision tree is the best of the two with an excellent R-square.

**Optimizing Decision tree regression model**

from sklearn.metrics import make\_scorer

from sklearn.model\_selection import GridSearchCV

from sklearn.model\_selection import ShuffleSplit

cv\_sets = ShuffleSplit(n\_splits = 10,

test\_size = 0.2, random\_state = 2)

regressor = DecisionTreeRegressor(random\_state = 4)

params = {'max\_depth': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10],

'splitter': ['best', 'random']}

scoring\_func = make\_scorer(r2\_score)

grid = GridSearchCV(regressor, params, cv = cv\_sets,

scoring = scoring\_func)

grid = grid.fit(X\_train, y\_train)

optimizedReg = grid.best\_estimator\_

y\_optimizedPrediction = optimizedReg.predict(X\_test)

print ('The r2\_score of the optimal regressor is:')

print (r2\_score(y\_test, y\_optimizedPrediction))

Oddly enough, the optimization code does not give better results than the default model. Perhaps the model parameters I chose to optimize are not correct.



**Conclusion**

In conclusion, we were able to build a model that evaluates target values based on the selected set of functions. Our decision tree regression model performed quite well with an R square of 0.65 ~ 0.7.

|  |  |  |
| --- | --- | --- |
| Result |  | • • • |
|  |
|  |
| Firstly we had no imagine what to do. We had only data and laptop, so first task for us is come what to do? Do descriptive or predict something? And in progress of searching of new idea, we learned different things related to data, as libraries in python, R language which is very comfortable for use, but python has more space for coding, building charts. Our data undarstanding improoved to next level. However it stays difficult for us to write report this was the most difficult part of project. We're going to learn more about documentation and data processing. For future we're going do it better and add machine learning and automotized learning for our projects. |

|  |  |  |
| --- | --- | --- |
| References |  | • • • |
|  |
|  |
| <https://www.kaggle.com/gregorut/videogamesales> source  <http://josephcslater.github.io/scipy-numpy-matplotlib-pylab.html> pylab  <http://pandas.pydata.org/pandas-docs/version/0.15/tutorials.html> pandos  <https://towardsdatascience.com/a-quick-introduction-to-the-pandas-python-library-f1b678f34673> pandos  <https://plot.ly/python/> python chart building  <https://www.analyticsvidhya.com/blog/2016/01/complete-tutorial-learn-data-science-python-scratch-2/> python data mining  <https://www.saedsayad.com/decision_tree_reg.htm> decision tree regression  <https://www.analyticsvidhya.com/blog/2018/08/k-nearest-neighbor-introduction-regression-python/> k neigbors regression python |