Neil Grogan C00205522

Introduction

For this tutorial we will use a dataset of video games and how well they sold. There are 16720 rows of data in the dataset which I sourced from: https://www.kaggle.com/rush4ratio/video-game-sales-with-ratings (https://www.kaggle.com/rush4ratio/video-game-sales-with-ratings) (renamed to VGS.csv)

In this tutorial I will go over some aspects of data preprocessing and how to get your dataset ready for machine learning.

Once we have a clean and prepared dataset I will go over a few different machine learning techniques for regression.

(This tutorial is inspired by the following tutorial: https://www.youtube.com/watch?v=0Lt9w-BxKFQ (https://www.youtube.com/watch?v=0Lt9w-BxKFQ))

I will be using Scikit-learn for this tutorial: https://scikit-learn.org/stable/ (https://scikit-learn.org/stable/)

Imports

Python has a large number of helpful libraries for machine learning.

Pandas and Numpy are two of the most popular.

Pandas provide data structures and operations for manipulating data whereas Numpy helps with the large arrays of data we'll be working with

https://pandas.pydata.org/ (https://pandas.pydata.org/)

http://www.numpy.org/ (http://www.numpy.org/)

In [1]:

```
import pandas as pd
import numpy as np
```

Let's look at our dataset

(Note: It is always a good idea to open the dataset with an application like excel or sheets first and familiarize yourself with the data and its columns. This will make it easier to plan and understand what you need to do.)

We can use Pandas to read our csv file. The read_csv method takes in a few parameters.

1: The name of the file we wish to read.

2: The delimiter tells us what seperates the data, it is usually either a ',' or ';'

3: We can assign names to each of the columns in the dataset if we wish to change them

We assign this to a variable called VGS (video game sales)

(If you're having any issues with the delimter this may help https://support.clever.com/hc/en-us/articles/218518698-How-do-I-troubleshoot-CSV-formatting-errors- (https://support.clever.com/hc/en-us/articles/218518698-How-do-I-troubleshoot-CSV-formatting-errors-))

In [2]:

```
VGS = pd.read_csv('VGS.csv', delimiter = ',', names=['Name', 'Platform', 'Release Year'
, 'Genre', 'Publisher', 'North American Sales', 'European Sales','Japanese Sales','Othe
r Sales', 'Global Sales', 'Critic Score', 'Number of Critics', 'User Score', 'Number of
User Scores', 'Developer', 'Rating'])
```

We can use the pandas method .info() to describe the data

In [3]:

```
VGS.info()
# As you can see we have 16 columns of data
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 16720 entries, 0 to 16719
Data columns (total 16 columns):
Name                     16718 non-null object
Platform                 16720 non-null object
Release Year             16451 non-null object
Genre                    16718 non-null object
Publisher                16666 non-null object
North American Sales     16720 non-null object
European Sales           16720 non-null object
Japanese Sales           16720 non-null object
Other Sales              16720 non-null object
Global Sales             16720 non-null object
Critic Score             8138 non-null object
Number of Critics        8138 non-null object
User Score               10016 non-null object
Number of User Scores    7591 non-null object
Developer                10097 non-null object
Rating                   9951 non-null object
dtypes: object(16)
memory usage: 2.0+ MB
```

Having null values in our dataset could break our functions later so we need to check if there is any missing values.

The method .isnull() tells is if there are any null values, combining it to .isnull.sum() gives us how many null values exist in each column

In [4]:

```
# Get the number of rows that have null values for each column
VGS.isnull().sum()
```

Out[4]:

```
Name                       2
Platform                   0
Release Year             269
Genre                      2
Publisher                 54
North American Sales       0
European Sales             0
Japanese Sales             0
Other Sales                0
Global Sales               0
Critic Score            8582
Number of Critics       8582
User Score              6704
Number of User Scores   9129
Developer               6623
Rating                  6769
dtype: int64
```

As you can see we have quite a lot of null values, this will cause an issue so we need to deal with this when we preprocess our data.

Data preprocessing

Drop any empty values

There are a few different ways to handle missing values in a dataset. The best one to use depends on your situation.

1: You can replace all null values with 0's but this could skew any machine learning results.

2: You could replace all null values with an average of all values in that column.

3: You can drop all rows that contain any null data, if you have a small dataset this has the pitfall of making your machine learning worse

In this case we will drop all rows with null values as we have a large enough dataset that it will be fine.

(Don't worry, the data will still be in your file, just not in our panda dataframe)

In [5]:

```
# dropna removes rows with null values
# axis 0 means the rows, inplace will keep our changes after running the drop, any tell
s it to drop a row if any date is empty
VGS.dropna(axis=0, how='any', inplace=True)
```

In [6]:

```
# Now let's see how many null values there are
VGS.isnull().sum()
```

Out[6]:

```
Name                     0
Platform                 0
Release Year             0
Genre                    0
Publisher                0
North American Sales     0
European Sales           0
Japanese Sales           0
Other Sales              0
Global Sales             0
Critic Score             0
Number of Critics        0
User Score               0
Number of User Scores    0
Developer                0
Rating                   0
dtype: int64
```

Drop any columns we don't need

We want to predict the global sales column, to do these we don't need some of the existing columns, we'll drop the name, year of release, EU sales, NA sales, JP sales and other sales columns as they won't be of use to us.

(Don't worry, the data will still be in your file, just not in our panda dataframe)

In [7]:

```
# lets drop the columns we don't want to use for our machine learning
# axis 1 means columns, inplace keeps our changes
VGS.drop(['Name', 'Release Year', 'North American Sales', 'European Sales', 'Japanese S
ales', 'Other Sales'], axis=1, inplace=True)
```

In [8]:

```
# Let's look at our data now that we've removed those rows
VGS.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 6826 entries, 0 to 16707
Data columns (total 10 columns):
Platform                 6826 non-null object
Genre                    6826 non-null object
Publisher                6826 non-null object
Global Sales             6826 non-null object
Critic Score             6826 non-null object
Number of Critics        6826 non-null object
User Score               6826 non-null object
Number of User Scores    6826 non-null object
Developer                6826 non-null object
Rating                   6826 non-null object
dtypes: object(10)
memory usage: 586.6+ KB
```

In [9]:

```
# As the first row contains the column names, lets drop it
# [0] means row, [1] would drop the first column
VGS.drop([0], inplace=True)
```

In [10]:

```
# .head() displays the top 5 rows in our dataset
VGS.head()
```

Out[10]:

	Platform	Genre	Publisher	Global Sales	Critic Score	Number of Critics	User Score	Number of User Scores	Develope
1	Wii	Sports	Nintendo	82.53	76	51	8	322	Nintendo
3	Wii	Racing	Nintendo	35.52	82	73	8.3	709	Nintendo
4	Wii	Sports	Nintendo	32.77	80	73	8	192	Nintendo
7	DS	Platform	Nintendo	29.8	89	65	8.5	431	Nintendo
8	Wii	Misc	Nintendo	28.92	58	41	6.6	129	Nintendo

Binning

We want to predict the "Global Sales" column, however there as so many different values in this column it will be very hard to predict one. Therefore we will divide them into sections. By only having 5 different values, we make it easier to predict an outcome. The number of bins you choose (if you need to) will depend on the values in your dataset.

The bins represent the sections or intervals we have used to divide our data into

Problems can occur during the binning process if not all data is the same type, therefore we want to ensure that all the data in the "Global Sales" column is of the same type (float)

https://pandas.pydata.org/pandas-docs/version/0.23.4/generated/pandas.cut.html (https://pandas.pydata.org/pandas-docs/version/0.23.4/generated/pandas.cut.html)

In [11]:

```
# first ensure all values in global sales are the same data type
VGS['Global Sales'] = VGS['Global Sales'].astype(float)

# now lets define the bins we will use, the bins must be in equal increments so we will
go up in 10's for this tutorial
bins = (.001,.01, .1, 1, 10, 100)

# define the names we will associate with each bin
group_names = ['poor','okay','good','amazing', 'incredible']

# Now we use the pandas method .cut() to overwrite the global sales with our new bins
VGS['Global Sales'] = pd.cut(VGS['Global Sales'], bins=bins, labels=group_names)

# let see how many unique values are in the 'Global Sales' column now and what they are
VGS['Global Sales'].unique()
```

Out[11]:

```
[incredible, amazing, good, okay, poor]
Categories (5, object): [poor < okay < good < amazing < incredible]
```

Label Encoder

We need to work with numbers for the machine learning, so we have to convert the data in the remaining columns to numbers rather than strings. To do this we use the label encoder which assigns a number to each unique string in our column.

https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.LabelEncoder.html (https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.LabelEncoder.html)

In [12]:

```
from sklearn.preprocessing import LabelEncoder
# Get the LabelEncoder so we can convert them to numbers
label_quality = LabelEncoder()

# Now lets convert all our columns using the label qualities fit and transform method.
VGS['Global Sales'] = label_quality.fit_transform(VGS['Global Sales'])
VGS['Platform'] = label_quality.fit_transform(VGS['Platform'])
VGS['Genre'] = label_quality.fit_transform(VGS['Genre'])
VGS['Publisher'] = label_quality.fit_transform(VGS['Publisher'])
VGS['Developer'] = label_quality.fit_transform(VGS['Developer'])
VGS['Rating'] = label_quality.fit_transform(VGS['Rating'])
```

In [13]:

```
# Let's look at our data now as you can see we are now working with all numbers
VGS.head()
```

Out[13]:

	Platform	Genre	Publisher	Global Sales	Critic Score	Number of Critics	User Score	Number of User Scores	Developer
1	12	10	160	2	76	51	8	322	771
3	12	6	160	2	82	73	8.3	709	771
4	12	10	160	2	80	73	8	192	771
7	2	4	160	2	89	65	8.5	431	771
8	12	3	160	2	58	41	6.6	129	771

In [14]:

```
# If you wanted to get a count of all the different values in a column you could do so
 using the .value_counts() method like so

VGS['Global Sales'].value_counts()
```

Out[14]:

```
1    3917
3    1502
0    1267
4      99
2      40
Name: Global Sales, dtype: int64
```

Data visualization

If you would like to visualize some of your data so that it is easier to interpret for yourself or someone else, seaborn and maplotlib are great libraries for this.

(Note: We use '%matplotlib' inline to tell the notebook to display the graphs inline with our code)

https://matplotlib.org/ (https://matplotlib.org/)

https://seaborn.pydata.org/ (https://seaborn.pydata.org/)

In [15]:

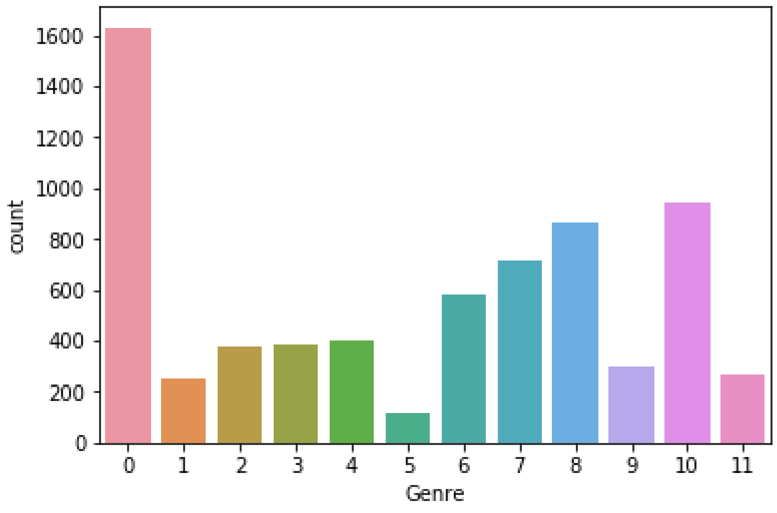
```
import seaborn as sns
import matplotlib.pyplot as plt

%matplotlib inline

# lets visualize the 'Genre' column to see how many categories we have and how many of
 each there are
sns.countplot(VGS['Genre'])
```

Out[15]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1ff61f50a58>
```



Machine learning

As we want to predict something we are better off using regression, however if you are trying to classify data then classification would be best for your dataset, if you'd like to learn more about classification and regression please read this article (https://medium.com/quick-code/regression-versus-classification-machine-learning-whats-the-difference-345c56dd15f7 (https://medium.com/quick-code/regression-versus-classification-machine-learning-whats-the-difference-345c56dd15f7))

Lets set up our training and testing sets

As they sound, the training set will be used to train our model by showing it data from that set.

They test set will not be shown to the model during its training, but shown to it after it is trained to see how accurate it is.

In [16]:

```
# we need to assign just the global sales column to the y variable and all other column
s to the X variable
X = VGS.drop('Global Sales', axis=1)
y = VGS['Global Sales']
```

In [17]:

```
# next we will use the train_test_split method to split up our data into the training a
nd testing sets for X and y
# we will use 20% of the data for testing (0.2)
# random state simply tells it to start at a random point when assigning data to each s
et

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=4
2)
```

Scaling

We need to scale our data so that all the values are closer together and we're not comparing 0.002 to 400.9, this would change the weighting during the machine learning and skew our results.

https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html (https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html)

In [18]:

```
# Let's scale our data so there is no bias
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()

# The fit_transform() method combines both the fit() and the transform() methods togeth
er
X_train = sc.fit_transform(X_train)
X_test = sc.fit_transform(X_test)
```

Metrics

The metrics package of sklearn allows us to test how well our models have performed.

Let's import the r2, mean_asolute_error and the explained_variance_score libraries

https://scikit-learn.org/stable/modules/generated/sklearn.metrics.r2_score.html#sklearn.metrics.r2_score (https://scikit-learn.org/stable/modules/generated/sklearn.metrics.r2_score.html#sklearn.metrics.r2_score)

https://scikit-learn.org/stable/modules/generated/sklearn.metrics.mean_absolute_error.html (https://scikit-learn.org/stable/modules/generated/sklearn.metrics.mean_absolute_error.html)

https://scikit-learn.org/stable/modules/generated/sklearn.metrics.explained_variance_score.html#sklearn.metrics.explained_ (https://scikit-learn.org/stable/modules/generated/sklearn.metrics.explained_variance_score.html#sklearn.metrics.explained_

explained variance

Measures the proportion to which a mathematical model accounts for the variation of a given data set

mean absolute error

Tells us how big an error we can expect in our model on average

r2 score

This is the proportion of the variance in the dependent variable that is predictable from the independent variable, essentially it's a measure of how well the regression predictions approximate the real data points.

In [19]:

```
from sklearn.metrics import mean_absolute_error, explained_variance_score, r2_score
```

Linear Regression

https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html (https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html)

Linear regression is pretty simple, you plot a line based off of a group of X and y variables and once you have the slope of that line you can plug an X in to get the corresponding y or vice versa.

image.png
(https://miro.medium.com/max/642/1*xxxqZtZExBJoxmYKIY-waw.png (https://miro.medium.com/max/642/1*xxxqZtZExBJoxmYKIY-waw.png))

In [20]:

```
from sklearn.linear_model import LinearRegression

lrg = LinearRegression().fit(X_train, y_train)

pred_lrg = lrg.predict(X_test)
```

Metrics

In [21]:

```
print(mean_absolute_error(y_test, pred_lrg))
```

```
0.7835290295556591
```

In [22]:

```
print(explained_variance_score(y_test, pred_lrg))
```

```
0.14824627786219557
```

In [23]:

```
print(r2_score(y_test, pred_lrg))
```

```
0.1481836207456264
```

Multi-layer Perceptron Regressor

https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPRegressor.html#sklearn.neural_network.MLPI
(https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPRegressor.html#sklearn.neural_network.MLPI

An MLP regressor is a from of neural network. It consists of, at least, three layers of nodes: an input layer, a hidden layer and an output layer. Each node has an activation function which defines how it's output will be calculated.

It learns by assigning weights or biases to pieces of data, this affects how important data will be determined to be when deciding on an outcome. These weights are tuned until deemed correct.

image.png
(http://rasbt.github.io/mlxtend/user_guide/classifier/NeuralNetMLP_files/neuralnet_mlp_1.png
(http://rasbt.github.io/mlxtend/user_guide/classifier/NeuralNetMLP_files/neuralnet_mlp_1.png))

In [24]:

```
from sklearn.neural_network import MLPRegressor

MLPR = MLPRegressor(hidden_layer_sizes=(10,),
                    activation='relu',
                    solver='adam',
                    learning_rate='adaptive',
                    max_iter=1000,
                    learning_rate_init=0.01,
                    alpha=0.01)

MLPR.fit(X_train, y_train)

pred_MLPR = lrg.predict(X_test)
```

Metrics

In [25]:

```
print(mean_absolute_error(y_test, pred_MLPR))
```

```
0.7835290295556591
```

In [26]:

```
print(explained_variance_score(y_test, pred_MLPR))
```

```
0.14824627786219557
```

In [27]:

```
print(r2_score(y_test, pred_MLPR))
```

```
0.1481836207456264
```

Linear SVR

https://scikit-learn.org/stable/modules/generated/sklearn.svm.LinearSVR.html#sklearn.svm.LinearSVR (https://scikit-learn.org/stable/modules/generated/sklearn.svm.LinearSVR.html#sklearn.svm.LinearSVR)

This one may be a bit confusing but what we are essentially tryng to do is try and fit points in margins either side of a hyper plane and decide on what distance those margins should be from the hyperplane to see which points give the least error.

image.png

(https://www.researchgate.net/profile/Nittaya_Kerdprasop/publication/323588842/figure/fig1/AS:612636878184 of-linear-support-vector-regression.png (https://www.researchgate.net/profile/Nittaya_Kerdprasop/publication/323588842/figure/fig1/AS:612636878184 of-linear-support-vector-regression.png))

In [28]:

```
from sklearn.svm import LinearSVR
```

In [29]:

```
LSVR = LinearSVR()

LSVR.fit(X_train, y_train)

pred_LSVR = LSVR.predict(X_test)
```

Metrics

In [30]:

```
print(mean_absolute_error(y_test, pred_LSVR))
```

```
0.6872444576866795
```

In [31]:

```
print(explained_variance_score(y_test, pred_LSVR))
```

```
0.007241438236222142
```

In [32]:

```
print(r2_score(y_test, pred_LSVR))
```

```
-0.0822293491501711
```

Decision Trees

https://scikit-learn.org/stable/modules/tree.html#regression (https://scikit-learn.org/stable/modules/tree.html#regression)

A decision tree is what it sounds like, when a decision is made, the path branches off for each different option. This process repeats for all the decisions until all the branches are formed.

image.png

(https://cdn-images-1.medium.com/max/1200/0*Yclq0kqMAwCQcIV_.jpg (https://cdn-images-1.medium.com/max/1200/0*Yclq0kqMAwCQcIV_.jpg))

In [33]:

```
from sklearn import tree

dtr = tree.DecisionTreeRegressor()

dtr.fit(X_train, y_train)

pred_dtr = dtr.predict(X_test)
```

Metrics

In [34]:

```
print(mean_absolute_error(y_test, pred_dtr))
```

```
0.8249084249084249
```

In [35]:

```
print(explained_variance_score(y_test, pred_dtr))
```

```
-0.4182273697655845
```

In [36]:

```
print(r2_score(y_test, pred_dtr))
```

```
-0.4537967133085139
```

Random Forest Regressor

https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestRegressor.html (https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestRegressor.html)

This is very similar to the decision trees we mentioned earlier, however, we combine many decision trees to make up a 'forest'. This is known as an ensemble technique.

image.png

(https://cdn-images-1.medium.com/max/1600/1*jEGFJCm4VSG0OzoqFUQJQg.jpeg (https://cdn-images-1.medium.com/max/1600/1*jEGFJCm4VSG0OzoqFUQJQg.jpeg))

In [37]:

```
from sklearn.ensemble import RandomForestRegressor

rfr = RandomForestRegressor(max_depth=2, random_state=0,n_estimators=100)

rfr.fit(X_train, y_train)

pred_rfr = rfr.predict(X_test)
```

```
C:\Users\elite\AppData\Local\Continuum\anaconda3\lib\site-packages\sklearn
\ensemble\weight_boosting.py:29: DeprecationWarning: numpy.core.umath_test
s is an internal NumPy module and should not be imported. It will be remov
ed in a future NumPy release.
  from numpy.core.umath_tests import inner1d
```

In [38]:

```
print(mean_absolute_error(y_test, pred_rfr))
```

```
0.7931712519368798
```

In [39]:

```
print(explained_variance_score(y_test, pred_rfr))
```

```
0.13001110394774773
```

In [40]:

```
print(r2_score(y_test, pred_rfr))
```

```
0.13000775266962983
```

Results

We have tried to use 5 different machine learning techniques and based off their results most did not perform well.

We will focus on the r2 score mainly as it is a very easy way to determine how good your model is.

- Linear Regression = 0.1481836207456264
- Multi-layer Perceptron Regressor = 0.1481836207456264
- Linear SVR = -0.08197588091941
- Decision Trees = -0.4841919292896082
- Random Forest Regressor = 0.13000775266962983

A score between -0.1 to 0.1 is considered weak at best

-0.3 to -0.1 or 0.1 to 0.3 is still weak

-0.5 to -0.3 or 0.3 to 0.5 is okay

and -1.0 to -0.5 or 1.0 to 0.5 is strong

Therefore our best model was the Decision tree with -0.4841919292896082 which is a moderate score.

There other models are perhaps still worth trying but would require some tuning to try and improve them.

Conclusions

This tutorial has gone over a few different aspects of data preprocessing, machine learning and some useful packages for sklearn.

Hopefully this has given you a good understanding of how to use scikit-learn and data science in python in general.

This tutorial did not go into how to fine tune your machine learning algorithms but there are ways to do so and possibly improve your results, you can find out how by going to https://scikit-learn.org/ (https://scikit-learn.org/) and searching for the algorithm you wish to improve.

If you'd like to learn more check out scikit's offical documentation: https://scikit-learn.org/ (https://scikit-learn.org/)

or check out python data science handbook:
https://tanthiamhuat.files.wordpress.com/2018/04/pythondatasciencehandbook.pdf
(https://tanthiamhuat.files.wordpress.com/2018/04/pythondatasciencehandbook.pdf)

Here are some great websites for finding datasets:

https://www.kaggle.com/ (https://www.kaggle.com/)

https://data.gov.ie/ (https://data.gov.ie/)

https://www.reddit.com/r/datasets/ (https://www.reddit.com/r/datasets/)

https://www.kdnuggets.com/datasets/index.html (https://www.kdnuggets.com/datasets/index.html)