Descriptions and Implementations of algorithms

Steam Games Predictor: by: Kornel Zieliński, Krystian Rodzaj, Krystian Wojakiewicz

Introduction

To ensure our algorithms can correctly distinguish our input parameters, we first have to encode them. This process has been explained in the previous presentation. Now, that we have our encoded datasets, we can start implementing the chosen algorithms. For the implementation of our algorithms we will use the **sklearn** Python library, which provides a wide range of machine learning tools. In this presentation we will cover: **Support Vector Machines**, **Linear Regression**, **Decision Trees**. We will look at the different parameter used for building the algorithms, how they were achieved, and what they represent. The results for some of the runs of the algorithms will also be shown. In the future, neural networks will be implemented and compared to the results presented here.

Data preparation

Below you can see methods used for storing achieved figures ("save_fig") and for loading the csv files containing encoded datasets ("load_data"). Some useful constants are also set here (paths, folder names).

In [2]:

```
from __future__ import division, print_function, unicode_literals

import numpy as np
import os
import matplotlib
import matplotlib.pyplot as plt
plt.rcParams['axes.labelsize'] = 14
plt.rcParams['xtick.labelsize'] = 12
plt.rcParams['ytick.labelsize'] = 12

# Location, in which the figures will be saved
PROJECT_ROOT_DIR = "."
CHAPTER_ID = "machine_learning_part"
IMAGES_PATH = os.path.join(PROJECT_ROOT_DIR, "pictures", CHAPTER_ID)

def save_fig(fig_id, tight_layout=True, fig_extension="png", resolution=300):
    path = os.path.join(IMAGES_PATH, fig_id + "." + fig_extension)
    print("Saving image", fig_id)
    if tight_layout:
        plt.tight_layout()
    plt.savefig(path, format=fig_extension, dpi=resoflution)
```

In [3]:

```
import os
import pandas as pd

def load_data(steam_path, file):
    csv_path = os.path.join(steam_path, file)
    return pd.read_csv(csv_path, error_bad_lines=False)
```

In [4]:

```
steam = load_data('./data_shuffle/data/', "enc_steam.csv")
```

Using the **head** method of the **Pandas** DataFrame structure, we can look at the first few rows of the dataset.

In [5]:

```
steam.head()
```

Out[5]:

	Unnamed: 0	Accounting	Action	Adventure	Animation & Modeling	Audio Production	Casual	Design & Illustration	Documentary	Early Access	...	Year	English	Dev
0	0	0	1	0	0	0	0	0	0	0	...	2000	1	
1	1	0	1	0	0	0	0	0	0	0	...	1999	1	
2	2	0	1	0	0	0	0	0	0	0	...	2003	1	
3	3	0	1	0	0	0	0	0	0	0	...	2001	1	
4	4	0	1	0	0	0	0	0	0	0	...	1999	1	

5 rows × 73 columns

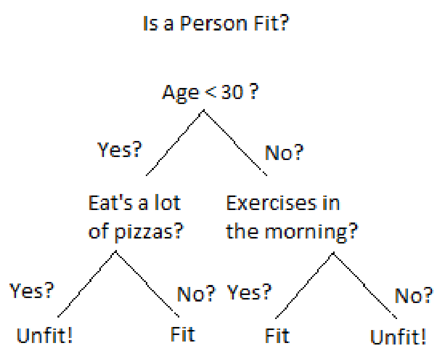
In [6]:

```
steam_cl = steam.copy()
steam_lin = steam.copy()
```

DecisionTreeClassifier

The decision tree learning method is a predictive model commonly used in machine learning and data mining. The idea is based on decision trees, in which the features of the examined subject are represented as the branches, and the target values (output classes) are represented as leaves. The algorithms traverses the tree from the root, making a decision about the given feature at each branch and continuing depending on that decision. When a leaf is reached, the final verdict can be made in terms of the target value residing in that particular leaf. If the target values are a finite set, the tree may be called a classification tree.

In our program, we used the **sklearn.tree.DecisionTreeClassifier** class to build our algorithm.



First, we need to divide our dataset into one training set and one verification set. The goal for this algorithm is to predict the amount of potential buyers for a new game, based on features like: genre, developers, release date. The **Owners** feature is represented by ranges of owners, e.g. [2000 - 5000]. Thus, we remove the **Owners** feature from our training set and we'll use it for validation.

In [7]:

```
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score
```

```
from sklearn.model_selection import cross_val_score

x,y = steam_cl.loc[:,steam_cl.columns != 'Owners'], steam_cl.loc[:,'Owners']
x_train,x_test,y_train,y_test = train_test_split(x,y,test_size = 0.3,random_state = 42)
```

In [8]:

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score

tree_clf = DecisionTreeClassifier()
```

Below, the process of finding the optimal hyperparameters is shown. We used the **sklearn.model_selection.RandomizedSearchCV** tool with ten iterations to find more suitable hyperparamers.

In [18]:

```
from sklearn.model_selection import RandomizedSearchCV
from scipy.stats import randint as sp_randint

tree_clf = DecisionTreeClassifier()

param_dist = {"max_depth": sp_randint(1,22),
              "max_features": sp_randint(1, 22),
              "min_samples_split": sp_randint(2, 100),
              "random_state": sp_randint(2, 100),
              "criterion": ["gini", "entropy"]}

# run randomized search
n_iter_search = 10
random_search = RandomizedSearchCV(tree_clf, param_distributions=param_dist,
                                   n_iter=n_iter_search, cv=5, iid=False)
random_search.fit(x_train, y_train)
```

```
/home/korni/anaconda3/lib/python3.7/site-packages/sklearn/model_selection/_split.py:667: UserWarning: T
he least populated class in y has only 1 members, which is less than n_splits=5.
  % (min_groups, self.n_splits)), UserWarning)
/home/korni/anaconda3/lib/python3.7/site-packages/sklearn/model_selection/_search.py:823: FutureWarning
: The parameter 'iid' is deprecated in 0.22 and will be removed in 0.24.
  "removed in 0.24.", FutureWarning)
```

Out[18]:

```
RandomizedSearchCV(cv=5, error_score=nan,
                   estimator=DecisionTreeClassifier(ccp_alpha=0.0,
                                                    class_weight=None,
                                                    criterion='gini',
                                                    max_depth=None,
                                                    max_features=None,
                                                    max_leaf_nodes=None,
                                                    min_impurity_decrease=0.0,
                                                    min_impurity_split=None,
                                                    min_samples_leaf=1,
                                                    min_samples_split=2,
                                                    min_weight_fraction_leaf=0.0,
                                                    presort='deprecated',
                                                    random_state=None,
                                                    splitter='best'),
                   i...
                                        'max_features': <scipy.stats._distn_infrastructure.rv_frozen ob
ject at 0x7f4b955323d0>,
                                        'min_samples_split': <scipy.stats._distn_infrastructure.rv_froz
en object at 0x7f4b95532650>,
                                        'random_state': <scipy.stats._distn_infrastructure.rv_frozen ob
ject at 0x7f4b955325d0>},
                   pre_dispatch='2*n_jobs', random_state=None, refit=True,
                   return_train_score=False, scoring=None, verbose=0)
```

In [19]:

```
print(random_search.best_score_)
print(random_search.best_params_)
```

```
0.7220872097433931
{'criterion': 'entropy', 'max_depth': 12, 'max_features': 10, 'min_samples_split': 62, 'random_state': 14}
```

These are the results of the "RandomizedSearchCV".

- **best score**: best prediction rate achieved,
- **best_params**: the parameters, which generated the highest prediction rate.

In [21]:

```
param_dist = {"max_depth": sp_randint(10,14),
              "max_features": sp_randint(8, 12),
              "min_samples_split": sp_randint(50, 70),
              "random_state": sp_randint(2, 20),
              "criterion": ["gini", "entropy"]}

# run randomized search
n_iter_search = 10
random_search = RandomizedSearchCV(tree_clf, param_distributions=param_dist,
                                   n_iter=n_iter_search, cv=5, iid=False)
random_search.fit(x_train, y_train)
```

```
/home/korni/anaconda3/lib/python3.7/site-packages/sklearn/model_selection/_split.py:667: UserWarning: The least populated class in y has only 1 members, which is less than n_splits=5.
  % (min_groups, self.n_splits)), UserWarning)
/home/korni/anaconda3/lib/python3.7/site-packages/sklearn/model_selection/_search.py:823: FutureWarning: The parameter 'iid' is deprecated in 0.22 and will be removed in 0.24.
  "removed in 0.24.", FutureWarning)
```

Out[21]:

```
RandomizedSearchCV(cv=5, error_score=nan,
                   estimator=DecisionTreeClassifier(ccp_alpha=0.0,
                                                    class_weight=None,
                                                    criterion='gini',
                                                    max_depth=None,
                                                    max_features=None,
                                                    max_leaf_nodes=None,
                                                    min_impurity_decrease=0.0,
                                                    min_impurity_split=None,
                                                    min_samples_leaf=1,
                                                    min_samples_split=2,
                                                    min_weight_fraction_leaf=0.0,
                                                    presort='deprecated',
                                                    random_state=None,
                                                    splitter='best'),
                   i...
                                        'max_features': <scipy.stats._distn_infrastructure.rv_frozen object at 0x7f4b951f2990>,
                                        'min_samples_split': <scipy.stats._distn_infrastructure.rv_frozen object at 0x7f4b951f2d10>,
                                        'random_state': <scipy.stats._distn_infrastructure.rv_frozen object at 0x7f4b951f7050>},
                   pre_dispatch='2*n_jobs', random_state=None, refit=True,
                   return_train_score=False, scoring=None, verbose=0)
```

In [22]:

```
print(random_search.best_score_)
```

```
print(random_search.best_score_)
print(random_search.best_params_)
```

```
0.7239337578447496
{'criterion': 'gini', 'max_depth': 10, 'max_features': 10, 'min_samples_split': 64, 'random_state': 5}
```

In [24]:

```
from sklearn.model_selection import GridSearchCV

param_grid = [
    {'max_depth': list(range(9, 11)), 'max_features': list(range(9,11)), 'min_samples_split': list(range(60, 67)), 'random_state': list(range(2, 20))}
]
grid = GridSearchCV(tree_clf, param_grid, cv=5, scoring='accuracy')

grid.fit(x_train, y_train)
```

```
/home/korni/anaconda3/lib/python3.7/site-packages/sklearn/model_selection/_split.py:667: UserWarning: The least populated class in y has only 1 members, which is less than n_splits=5.
  % (min_groups, self.n_splits)), UserWarning)
```

Out[24]:

```
GridSearchCV(cv=5, error_score=nan,
             estimator=DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None,
                                              criterion='gini', max_depth=None,
                                              max_features=None,
                                              max_leaf_nodes=None,
                                              min_impurity_decrease=0.0,
                                              min_impurity_split=None,
                                              min_samples_leaf=1,
                                              min_samples_split=2,
                                              min_weight_fraction_leaf=0.0,
                                              presort='deprecated',
                                              random_state=None,
                                              splitter='best'),
             iid='deprecated', n_jobs=None,
             param_grid=[{'max_depth': [9, 10], 'max_features': [9, 10],
                          'min_samples_split': [60, 61, 62, 63, 64, 65, 66],
                          'random_state': [2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12,
                                           13, 14, 15, 16, 17, 18, 19]}],
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring='accuracy', verbose=0)
```

In [25]:

```
print(grid.best_score_)
print(grid.best_params_)
print(grid.best_estimator_)
```

```
0.7254115670428991
{'max_depth': 9, 'max_features': 10, 'min_samples_split': 65, 'random_state': 13}
DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='gini',
                       max_depth=9, max_features=10, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=65,
                       min_weight_fraction_leaf=0.0, presort='deprecated',
                       random_state=13, splitter='best')
```

FINAL MODEL

In [9]:

```
tree_clf = DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='gini',
```

```
tree_clf = DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='gini',
                       max_depth=9, max_features=10, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=65,
                       min_weight_fraction_leaf=0.0, presort='deprecated',
                       random_state=13, splitter='best')


tree_clf.fit(x_train,y_train)
y_pred = tree_clf.predict(x_test)
```

Cross-Validation

In order to avoid overfitting of the model we used a technique called cross validation. This approach doesn't require usage of the validation set. In the technique called k-fold CV (short for cross validation) the training set is splitted into k smaller sets and for each of the k "folds" the model is trained using k-1 of the folds as a training set and the remaining fold is used as a validation set. After that the steps are repeated for some other "validation-fold" and a training set composed by other folds until every single fold have been used as a validation set. We tried it using 3 or 5 folds. That's why in our project everytime the cross validation method is called the results of it are stored in a three/five element table consisting of accuracy scores or mean squared errors.

In [10]:

```
cross_val_score(tree_clf, x_train, y_train)
```

```
/home/korni/anaconda3/lib/python3.7/site-packages/sklearn/model_selection/_split.py:667: UserWarning: T
he least populated class in y has only 1 members, which is less than n_splits=5.
  % (min_groups, self.n_splits)), UserWarning)
```

Out[10]:

```
array([0.72698496, 0.72381957, 0.72875989, 0.72216359, 0.72532982])
```

In [11]:

```
print(accuracy_score(y_test, y_pred))
```

```
0.7056506216914933
```

In [12]:

```
df = pd.DataFrame({'Actual': y_test, 'Predicted': y_pred})
df.head(25)
```

Out[12]:

	Actual	Predicted
10506	12	12
26313	12	12
2622	9	11
1489	11	11
19949	12	12
24157	12	12
14372	12	12
4424	12	12
8295	6	12
21560	12	12

	Actual	Predicted
8139	12	12
18659	12	12
21942	12	12
24272	12	12
21390	12	12
10607	12	12
13887	9	6
25695	12	12
14771	12	12
9889	12	12
23086	12	12
15533	12	12
20486	12	12
16293	6	12
16847	12	12

RandomForestClassifier

Random forests are an example of **ensemble learning**. Ensemble learning is a type of supervised learning and it involves taking multiple trained models, usually from the same base learner, and combining them to improve the prediction rate. The disadvantage is that this method needs significantly more computation than simple **decision trees**. Random forest can intuitively be thought of, as a collection of independent decision trees working together to produce a more accurate prediction.

In our, program the **sklearn.ensemble.RandomForestClassifier** is used.

In [13]:

```
from sklearn.ensemble import RandomForestClassifier

rf = RandomForestClassifier()
```

In [16]:

```
from sklearn.model_selection import RandomizedSearchCV
from scipy.stats import randint as sp_randint
param_dist = {"max_leaf_nodes": sp_randint(2,100),
              "min_samples_split": sp_randint(2, 100),
              "random_state": sp_randint(2, 100),

             }
# run randomized search
n_iter_search = 10
random_search = RandomizedSearchCV(rf, param_distributions=param_dist,
                                   n_iter=n_iter_search, cv=5, iid=False)
random_search.fit(x_train, y_train)
```

```
/home/korni/anaconda3/lib/python3.7/site-packages/sklearn/model_selection/_split.py:667: UserWarning: T
he least populated class in y has only 1 members, which is less than n_splits=5.
  % (min_groups, self.n_splits)), UserWarning)
/home/korni/anaconda3/lib/python3.7/site-packages/sklearn/model_selection/_search.py:823: FutureWarning
: The parameter 'iid' is deprecated in 0.22 and will be removed in 0.24.
  "removed in 0.24.", FutureWarning)
```

Out[16]:

```
RandomizedSearchCV(cv=5, error_score=nan,
```

```
                   estimator=RandomForestClassifier(bootstrap=True,
                                                    ccp_alpha=0.0,
                                                    class_weight=None,
                                                    criterion='gini',
                                                    max_depth=None,
                                                    max_features='auto',
                                                    max_leaf_nodes=None,
                                                    max_samples=None,
                                                    min_impurity_decrease=0.0,
                                                    min_impurity_split=None,
                                                    min_samples_leaf=1,
                                                    min_samples_split=2,
                                                    min_weight_fraction_leaf=0.0,
                                                    n_estimators=100,
                                                    n_jobs...
                   param_distributions={'max_leaf_nodes': <scipy.stats._distn_infrastructure.rv_frozen
object at 0x7fa2567fba90>,
                                        'min_samples_split': <scipy.stats._distn_infrastructure.rv_froz
en object at 0x7fa256811890>,
                                        'random_state': <scipy.stats._distn_infrastructure.rv_frozen ob
ject at 0x7fa25680c5d0>},
                   pre_dispatch='2*n_jobs', random_state=None, refit=True,
                   return_train_score=False, scoring=None, verbose=0)
```

In [17]:

```
print(random_search.best_score_)
print(random_search.best_params_)
```

```
0.7278913466069131
{'max_leaf_nodes': 90, 'min_samples_split': 26, 'random_state': 94}
```

In [18]:

```
param_dist = {"max_leaf_nodes": sp_randint(70,200),
              "min_samples_split": sp_randint(20, 35),
              "random_state": sp_randint(50, 100)
             }
# run randomized search
n_iter_search = 10
random_search = RandomizedSearchCV(rf, param_distributions=param_dist,
                                   n_iter=n_iter_search, cv=5, iid=False)
random_search.fit(x_train, y_train)
```

```
/home/korni/anaconda3/lib/python3.7/site-packages/sklearn/model_selection/_split.py:667: UserWarning: T
he least populated class in y has only 1 members, which is less than n_splits=5.
  % (min_groups, self.n_splits)), UserWarning)
/home/korni/anaconda3/lib/python3.7/site-packages/sklearn/model_selection/_search.py:823: FutureWarning
: The parameter 'iid' is deprecated in 0.22 and will be removed in 0.24.
  "removed in 0.24.", FutureWarning
```

Out[18]:

```
RandomizedSearchCV(cv=5, error_score=nan,
                   estimator=RandomForestClassifier(bootstrap=True,
                                                    ccp_alpha=0.0,
                                                    class_weight=None,
                                                    criterion='gini',
                                                    max_depth=None,
                                                    max_features='auto',
                                                    max_leaf_nodes=None,
                                                    max_samples=None,
                                                    min_impurity_decrease=0.0,
                                                    min_impurity_split=None,
                                                    min_samples_leaf=1,
                                                    min_samples_split=2,
                                                    min_weight_fraction_leaf=0.0,
                                                    n_estimators=100,
```

```
                                              n_estimators=100,
                                              n_jobs...
                   param_distributions={'max_leaf_nodes': <scipy.stats._distn_infrastructure.rv_frozen
object at 0x7fa2567d6790>,
                                        'min_samples_split': <scipy.stats._distn_infrastructure.rv_froz
en object at 0x7fa2567d6d50>,
                                        'random_state': <scipy.stats._distn_infrastructure.rv_frozen ob
ject at 0x7fa2567d6b10>},
                   pre_dispatch='2*n_jobs', random_state=None, refit=True,
                   return_train_score=False, scoring=None, verbose=0)
```

In [19]:

```python
print(random_search.best_score_)
print(random_search.best_params_)
```

```
0.7351726105920912
{'max_leaf_nodes': 184, 'min_samples_split': 25, 'random_state': 71}
```

In [13]:

```python
from sklearn.model_selection import RandomizedSearchCV
from scipy.stats import randint as sp_randint

param_dist = {"max_leaf_nodes": sp_randint(170,200),
              "min_samples_split": sp_randint(20, 30),
              "random_state": sp_randint(70, 80)
             }
# run randomized search
n_iter_search = 10
random_search = RandomizedSearchCV(rf, param_distributions=param_dist,
                                   n_iter=n_iter_search, cv=5, iid=False)
random_search.fit(x_train, y_train)
```

```
/home/korni/anaconda3/lib/python3.7/site-packages/sklearn/model_selection/_split.py:667: UserWarning: T
he least populated class in y has only 1 members, which is less than n_splits=5.
  % (min_groups, self.n_splits)), UserWarning)
/home/korni/anaconda3/lib/python3.7/site-packages/sklearn/model_selection/_search.py:823: FutureWarning
: The parameter 'iid' is deprecated in 0.22 and will be removed in 0.24.
  "removed in 0.24.", FutureWarning
```

Out[13]:

```
RandomizedSearchCV(cv=5, error_score=nan,
                   estimator=RandomForestClassifier(bootstrap=True,
                                              ccp_alpha=0.0,
                                              class_weight=None,
                                              criterion='gini',
                                              max_depth=None,
                                              max_features='auto',
                                              max_leaf_nodes=None,
                                              max_samples=None,
                                              min_impurity_decrease=0.0,
                                              min_impurity_split=None,
                                              min_samples_leaf=1,
                                              min_samples_split=2,
                                              min_weight_fraction_leaf=0.0,
                                              n_estimators=100,
                                              n_jobs...
                   param_distributions={'max_leaf_nodes': <scipy.stats._distn_infrastructure.rv_frozen
object at 0x7f85dcc20090>,
                                        'min_samples_split': <scipy.stats._distn_infrastructure.rv_froz
en object at 0x7f85dcc20110>,
                                        'random_state': <scipy.stats._distn_infrastructure.rv_frozen ob
ject at 0x7f85dcc20290>},
                   pre_dispatch='2*n_jobs', random_state=None, refit=True,
                   return_train_score=False, scoring=None, verbose=0)
```

In [14]:

```
print(random_search.best_score_)
print(random_search.best_params_)
```

```
0.7358587238627244
{'max_leaf_nodes': 180, 'min_samples_split': 27, 'random_state': 74}
```

In [15]:

```
from sklearn.model_selection import GridSearchCV

param_grid = [
    {'max_leaf_nodes': list(range(180, 183)), 'min_samples_split': list(range(25,28)), 'random_state':
list(range(72, 74))}
]
grid = GridSearchCV(tree_clf, param_grid, cv=5, scoring='accuracy')

grid.fit(x_train, y_train)
```

```
/home/korni/anaconda3/lib/python3.7/site-packages/sklearn/model_selection/_split.py:667: UserWarning: T
he least populated class in y has only 1 members, which is less than n_splits=5.
  % (min_groups, self.n_splits)), UserWarning)
```

Out[15]:

```
GridSearchCV(cv=5, error_score=nan,
             estimator=DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None,
                                              criterion='gini', max_depth=9,
                                              max_features=10,
                                              max_leaf_nodes=None,
                                              min_impurity_decrease=0.0,
                                              min_impurity_split=None,
                                              min_samples_leaf=1,
                                              min_samples_split=65,
                                              min_weight_fraction_leaf=0.0,
                                              presort='deprecated',
                                              random_state=13,
                                              splitter='best'),
             iid='deprecated', n_jobs=None,
             param_grid=[{'max_leaf_nodes': [180, 181, 182],
                          'min_samples_split': [25, 26, 27],
                          'random_state': [72, 73]}],
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring='accuracy', verbose=0)
```

In [16]:

```
print(grid.best_score_)
print(grid.best_params_)
print(grid.best_estimator_)
```

```
0.7216121365071697
{'max_leaf_nodes': 180, 'min_samples_split': 25, 'random_state': 72}
DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='gini',
                       max_depth=9, max_features=10, max_leaf_nodes=180,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=25,
                       min_weight_fraction_leaf=0.0, presort='deprecated',
                       random_state=72, splitter='best')
```

FINAL MODEL

In [14]:

```
rf = RandomForestClassifier(ccp_alpha=0.0, class_weight=None, criterion='gini',
                       max_depth=9, max_features=10, max_leaf_nodes=180,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=25,
                       min_weight_fraction_leaf=0.0,
                       random_state=72)

rf.fit(x_train, y_train)

y_pred = rf.predict(x_test)
```

In [15]:

```
cross_val_score(rf, x_train, y_train, cv=3, scoring="accuracy")
```

```
/home/korni/anaconda3/lib/python3.7/site-packages/sklearn/model_selection/_split.py:667: UserWarning: T
he least populated class in y has only 1 members, which is less than n_splits=3.
  % (min_groups, self.n_splits)), UserWarning)
```

Out[15]:

```
array([0.73425135, 0.72866867, 0.73595061])
```

In [16]:

```
print(accuracy_score(y_test, y_pred))
```

```
0.7206697033115844
```

In [17]:

```
df = pd.DataFrame({'Actual': y_test, 'Predicted': y_pred})
df.head(25)
```

Out[17]:

	Actual	Predicted
10506	12	12
26313	12	12
2622	9	11
1489	11	11
19949	12	12
24157	12	12
14372	12	12
4424	12	12
8295	6	12
21560	12	12
8139	8	12
18659	12	12
21942	12	12
24272	12	12
21390	12	12
10607	12	12

	Actual	Predicted
13887	9	9
25695	12	12
14771	12	12
9889	12	12
23086	12	12
15533	12	12
20486	12	12
16293	6	12
16847	12	12

KNeighborsClassifier

A K nearest neighbor algorithm is a data classifier, which estimates probability that a data point is a member of one group or the other depending on the group, in which the nearest data points are located. KNN has been used in statistical estimation and pattern recognition already in the beginning of 1970's as a non-parametric technique.

To use this classifier we used the library sklearn.neighbors.KNeighborsClassifier. We started by looking for the best parameters for our classifier.

In [ ]:

```
from sklearn.model_selection import GridSearchCV
from sklearn.neighbors import KNeighborsClassifier

knn = KNeighborsClassifier()

k_range = list(range(15, 31))
param_grid = dict(n_neighbors=k_range)
grid = GridSearchCV(knn, param_grid, cv=5, scoring='accuracy')

grid.fit(x_train, y_train)
```

In [ ]:

```
print(grid.best_score_)
print(grid.best_params_)
print(grid.best_estimator_)
```

FINAL MODEL

In [ ]:

```
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                           metric_params=None, n_jobs=None, n_neighbors=19, p=2, weights='uniform')
knn.fit(x_train, y_train)
y_pred = knn.predict(x_test)
```

After finding the best parameters we used them in the classifier. Used parameters:

- **algorithm** – Algorithm used to compute the nearest neighbors.
- **leaf_size -** Leaf size passed to BallTree or KDTree. This can affect the speed of the construction and query, as well as the memory required to store the tree.
- **metric -** The distance metric to use for the tree. The default metric is minkowski, and with p=2 is equivalent to the standard Euclidean metric.
- **metric_params -** Additional keyword arguments for the metric function

- **n_jobs** - The number of parallel jobs to run for neighbors search
- **n_neighbors** - Number of neighbors to use by default for kneighbors queries
- **p** - Power parameter for the Minkowski metric
- **weights** - Weight function used in prediction. We used 'uniform' which means that All points in each neighborhood are weighted equally

In [19]:

```
cross_val_score(knn, x_train, y_train, cv=3, scoring="accuracy")
```

```
/home/korni/anaconda3/lib/python3.7/site-packages/sklearn/model_selection/_split.py:667: UserWarning: T
he least populated class in y has only 1 members, which is less than n_splits=3.
  % (min_groups, self.n_splits)), UserWarning)
```

Out[19]:

```
array([0.71066793, 0.70824759, 0.71141365])
```

In [20]:

```
print(accuracy_score(y_test, y_pred))
```

```
0.7008494398621199
```

We should be satisfied with chosen parameters. The percentage of correct prediction is 70%, which is a good result.

In [21]:

```
df = pd.DataFrame({'Actual': y_test, 'Predicted': y_pred})
df.head(25)
```

Out[21]:

	Actual	Predicted
10506	12	12
26313	12	12
2622	9	11
1489	11	11
19949	12	12
24157	12	12
14372	12	12
4424	12	12
8295	6	12
21560	12	12
8139	8	12
18659	12	12
21942	12	12
24272	12	12
21390	12	12
10607	12	12
13887	9	12
25695	12	12
14771	12	12

	Actual	Predicted
9889	12	12
23086	12	12
15533	12	12
20486	12	12
16293	6	12
16847	12	12

REGRESSION PART

Linear Regression is a machine learning algorithm based on supervised learning. Linear regression performs the task to predict a dependent variable value (y) based on a given independent variable (x). So, this regression technique finds out a linear relationship between input (x) and output (y). The regression line is the best-fit line for given data. In linear regression, the relationships are modeled using linear predictor functions which estimate model parameters based on data.

In this case, we took the ratings column for our output, which is responsible for the overall rating of the game given by players. To use this regressor we used the library sklearn.linear_model.LinearRegression.

BASIC LINEAR REGRESSOR

In [22]:

```
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score

x,y = steam_lin.loc[:,steam_lin.columns != 'Rating'], steam_lin.loc[:,'Rating']
x_train,x_test,y_train,y_test = train_test_split(x,y,test_size = 0.25,random_state = 42)
```

In [23]:

```
from sklearn.linear_model import LinearRegression

regr = LinearRegression()
regr.fit(x_train, y_train)
y_pred = regr.predict(x_test)
```

In [28]:

```
from sklearn.model_selection import cross_val_score

scores = cross_val_score(regr, x_train, y_train,
                         scoring="neg_mean_squared_error", cv=3)
regr_rmse_scores = np.sqrt(-scores)
```

In [29]:

```
def display_scores(scores):
    print("Scores:", scores)
    print("Mean:", scores.mean())
    print("Standard deviation:", scores.std())
```

In [30]:

```
display_scores(regr_rmse_scores)
```

```
Scores: [22.19912006 22.58280641 22.4197434 ]
Mean: 22.400556620133045
```

```
Standard deviation: 0.15722574789939386
```

The most interesting result parameter is the standard deviation. This value is very close to zero, which is a correct result.

In [31]:

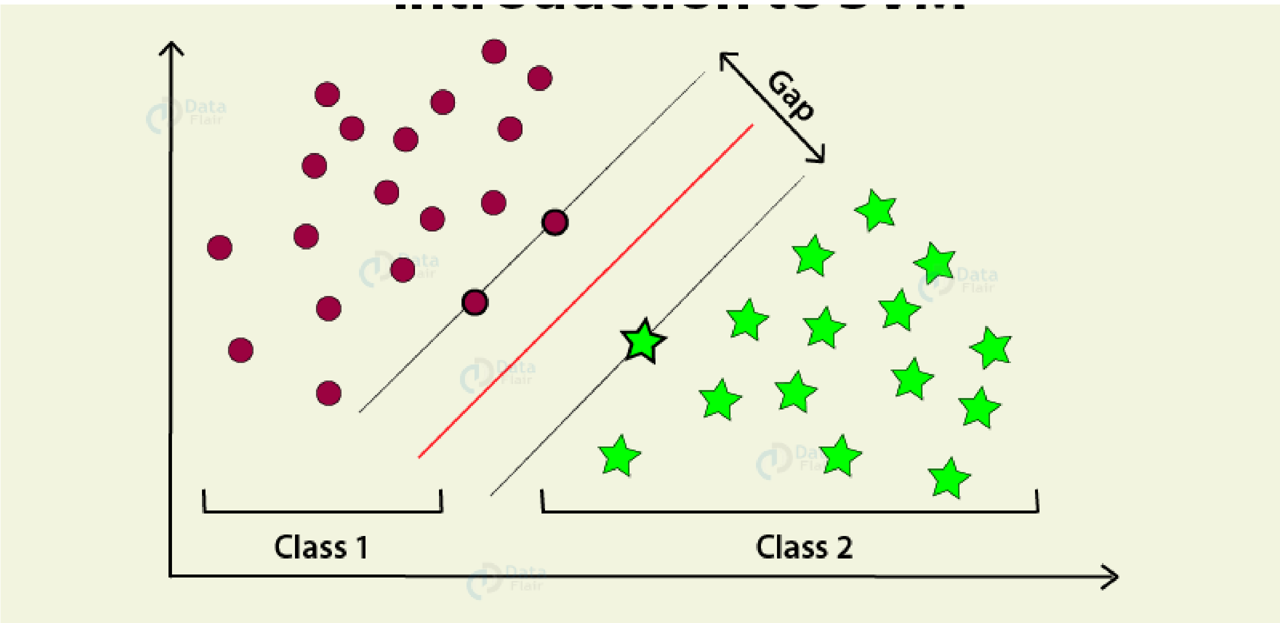
```
df = pd.DataFrame({'Actual': y_test, 'Predicted': y_pred})
df.head(25)
```

Out[31]:

	Actual	Predicted
10506	67.0	67.136819
26313	0.0	71.340231
2622	73.0	75.344646
1489	87.0	71.052604
19949	35.0	62.095020
24157	100.0	67.289610
14372	94.0	71.441188
4424	92.0	86.893058
8295	59.0	61.726586
21560	97.0	72.769288
8139	63.0	68.278870
18659	83.0	73.644100
21942	75.0	66.260007
24272	85.0	69.437414
21390	79.0	65.891525
10607	92.0	77.929307
13887	93.0	70.397588
25695	82.0	71.222868
14771	100.0	84.837322
9889	61.0	66.008482
23086	50.0	73.104049
15533	60.0	72.812892
20486	75.0	78.375268
16293	85.0	68.199341
16847	67.0	64.960454

SVM REGRESSOR

Support Vector Machines are a very powerful and versatile machine learning algorithms. It can be used in a linear or nonlinear classification tasks, regression tasks or to detect the outliers. It is especially useful in classification of the complex but not very big datasets. It operates on the principle of finding the widest gap between the separate categories, addition of antoher samples does not affect the margin because it is supported by the samples at the extremities, usually called supporting vectors. SVMs can also be used in regression tasks, this method is called support-vector regression (SVR). In this approach model depends only on subset of the training data, because cost function ignores any training data close to the model prediction.



In [20]:

```
from sklearn.svm import SVR

svr_rbf = SVR(kernel='rbf', C=100, gamma=0.1, epsilon=.1)
svr_rbf.fit(x_train, y_train)
y_pred = svr_rbf.predict(x_test)
```

In [22]:

```
from sklearn.model_selection import cross_val_score

scores = cross_val_score(svr_rbf, x_train, y_train,
                         scoring="neg_mean_squared_error", cv=3)
svr_rbf_rmse_scores = np.sqrt(-scores)
```

In [23]:

```
display_scores(svr_rbf_rmse_scores)
```

```
Scores: [23.14735249 23.56241822 23.50704331]
Mean: 23.40560467315412
Standard deviation: 0.18400586700463856
```

In [24]:

```
df = pd.DataFrame({'Actual': y_test, 'Predicted': y_pred})
df.head(25)
```

Out[24]:

	Actual	Predicted
10506	67.0	71.603198
26313	0.0	71.603198
2622	73.0	71.603198
1489	87.0	71.603198
19949	35.0	68.060669
24157	100.0	71.603198

	Actual	Predicted
14372	94.0	75.238684
4424	92.0	68.592391
8295	59.0	71.603198
21560	97.0	71.603198
8139	63.0	71.603198
18659	83.0	71.595086
21942	75.0	68.018753
24272	85.0	71.603198
21390	79.0	71.989553
10607	92.0	71.603198
13887	93.0	71.603198
25695	82.0	87.148876
14771	100.0	71.603198
9889	61.0	56.456677
23086	50.0	52.557790
15533	60.0	73.274185
20486	75.0	71.603198
16293	85.0	71.603198
16847	67.0	71.809911

RandomForestRegressor

In [34]:

```
from sklearn.ensemble import RandomForestRegressor

rf_regr = RandomForestRegressor(ccp_alpha=0.0, max_depth=9, max_features=10, max_leaf_nodes=180,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=25,
                       min_weight_fraction_leaf=0.0,
                       random_state=72)

rf_regr.fit(x_train, y_train)

y_pred = rf_regr.predict(x_test)
```

In [42]:

```
from sklearn.model_selection import cross_val_score

scores = cross_val_score(rf_regr, x_train, y_train,
                         scoring="neg_mean_squared_error", cv=5)
rf_regr_rmse_scores = np.sqrt(-scores)
```

In [43]:

```
display_scores(rf_regr_rmse_scores)
```

```
Scores: [22.07626998 21.74552655 22.01286883 22.12682454 22.2349739 ]
Mean: 22.03929276015576
Standard deviation: 0.1639006086142712
```

In [44]:

In [44]:

```
df = pd.DataFrame({'Actual': y_test, 'Predicted': y_pred})
df.head(50)
```

Out[44]:

	Actual	Predicted
10506	67.0	70.382869
26313	0.0	75.926196
2622	73.0	76.951582
1489	87.0	76.166092
19949	35.0	60.895129
24157	100.0	66.120683
14372	94.0	70.709694
4424	92.0	78.745918
8295	59.0	68.782455
21560	97.0	76.366781
8139	63.0	66.282876
18659	83.0	74.424134
21942	75.0	70.175803
24272	85.0	69.470579
21390	79.0	65.934889
10607	92.0	81.070525
13887	93.0	70.528285
25695	82.0	71.443682
14771	100.0	81.249155
9889	61.0	66.134821
23086	50.0	72.029327
15533	60.0	69.761921
20486	75.0	75.168967
16293	85.0	62.959633
16847	67.0	64.087615
15236	97.0	77.771159
12024	51.0	62.519747
24383	89.0	67.698012
14564	88.0	65.488940
15811	93.0	66.357916
8999	94.0	77.563190
26865	100.0	77.122857
4479	93.0	82.191612
7049	100.0	69.111027
21985	74.0	63.815314
2541	87.0	78.902818
12475	92.0	71.101601
21713	39.0	59.008709
2714	75.0	71.536950
20940	76.0	63.563596

	Actual	Predicted
20949	76.0	62.562596
2901	72.0	69.528673
5815	71.0	72.737155
13040	88.0	74.606165
22321	81.0	66.407811
14634	89.0	81.833847
11521	50.0	68.449273
10898	100.0	69.589566
20571	50.0	73.659137
18638	0.0	60.256725
9242	91.0	82.258054