

Steam_Dataset

August 15, 2019

```
[1]: import os
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import sklearn
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split, cross_val_score, KFold
from xgboost import XGBClassifier
import pickle
import time
import wordcloud
import seaborn
%matplotlib inline
```

0.1 Goal

The goal of project is to develop a model to predict the number of owners for steam games, which should be useful for new game design.

0.2 Outline

This project includes three major parts: 1. EDA: 1) check dataset 2) target feature (owners) 3) consider categorical features (split and frequency) 4) consider numerical features (correlation) 5) generate new features 2. Model development: 1) train model (XGB classification with class weights) 2) performance check 3) error analysis 4) feature importance

0.3 1. EDA

```
[2]: df = pd.read_csv("../data/steam.csv")
print("Dataset includes " + str(df.shape[0]) + " different games")
```

Dateset includes 27075 different games

0.3.1 1) Check Basic Information

```
[3]: ### Check Columns ###
print("There are " + str(len(df.columns)) + " columns in data")
```

```

print("Among them, some features can be got before the game releasing, which can
    ↳be used as our input features")
print("Such as :" + ", ".join(df.columns[1:12]) + ", " + df.columns[-1])
print("Some of them can only be got after the game releasing, which are the
    ↳targets of our model")
print("Such as :" + ", ".join(df.columns[14:17]))
print("Some of them can be got at both time using pre-release testing, but we
    ↳need to do certain conversion")
print("Such as :" + ", ".join(df.columns[12:14]))

```

There are 18 columns in data

Among them, some features can be got before the game releasing, which can be used as our input features

Such as :name, release_date, english, developer, publisher, platforms, required_age, categories, genres, steamspy_tags, achievements, price

Some of them can only be got after the game releasing, which are the targets of our model

Such as :average_playtime, median_playtime, owners

Some of them can be got at both time using pre-release testing, but we need to do certain conversion

Such as :positive_ratings, negative_ratings

```

[4]: ##### Check General Information #####
df.info()
df.describe()

```

```

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 27075 entries, 0 to 27074
Data columns (total 18 columns):
appid                27075 non-null int64
name                 27075 non-null object
release_date         27075 non-null object
english              27075 non-null int64
developer            27075 non-null object
publisher            27075 non-null object
platforms            27075 non-null object
required_age         27075 non-null int64
categories           27075 non-null object
genres               27075 non-null object
steamspy_tags        27075 non-null object
achievements         27075 non-null int64
positive_ratings     27075 non-null int64
negative_ratings     27075 non-null int64
average_playtime     27075 non-null int64
median_playtime      27075 non-null int64
owners               27075 non-null object
price                27075 non-null float64

```

```
dtypes: float64(1), int64(8), object(9)
memory usage: 3.7+ MB
```

```
[4]:
```

	appid	english	required_age	achievements	\
count	2.707500e+04	27075.000000	27075.000000	27075.000000	
mean	5.962035e+05	0.981127	0.354903	45.248864	
std	2.508942e+05	0.136081	2.406044	352.670281	
min	1.000000e+01	0.000000	0.000000	0.000000	
25%	4.012300e+05	1.000000	0.000000	0.000000	
50%	5.990700e+05	1.000000	0.000000	7.000000	
75%	7.987600e+05	1.000000	0.000000	23.000000	
max	1.069460e+06	1.000000	18.000000	9821.000000	

	positive_ratings	negative_ratings	average_playtime	median_playtime	\
count	2.707500e+04	27075.000000	27075.000000	27075.000000	
mean	1.000559e+03	211.027147	149.804949	146.05603	
std	1.898872e+04	4284.938531	1827.038141	2353.88008	
min	0.000000e+00	0.000000	0.000000	0.000000	
25%	6.000000e+00	2.000000	0.000000	0.000000	
50%	2.400000e+01	9.000000	0.000000	0.000000	
75%	1.260000e+02	42.000000	0.000000	0.000000	
max	2.644404e+06	487076.000000	190625.000000	190625.000000	

	price
count	27075.000000
mean	6.078193
std	7.874922
min	0.000000
25%	1.690000
50%	3.990000
75%	7.190000
max	421.990000

```
[5]: ##### Check None #####
df.isnull().sum()
```

```
[5]: appid      0
     name      0
     release_date  0
     english    0
     developer  0
     publisher  0
     platforms  0
     required_age  0
     categories  0
     genres     0
     steamspy_tags  0
     achievements  0
```

```

positive_ratings    0
negative_ratings    0
average_playtime    0
median_playtime     0
owners              0
price               0
dtype: int64

```

```

[6]: ##### Check Repeate Game Names #####
name_list = {s:0 for s in set(df["name"])}
for i in df["name"]:
    name_list[i] += 1
name_repeate = []
for key,value in name_list.items():
    if value != 1:
        name_repeate.append(key)

print("There are " + str(len(name_repeate)) + " game names are repetitive")
print("They are: " + ", ".join(name_repeate))
### One example
df[df["name"] == "New York Bus Simulator"]

```

There are 41 game names are repetitive

They are: Luna, New York Bus Simulator, Zombie Apocalypse, RUSH, Escape, Scorch, Exodus, Alone, Bounce, Rumpus, Cortex, Escape Room, Hide and Seek, Beyond the Wall, Space Maze, Mars 2030, Slice&Dice, Experience, The Mine, Dark Matter, Alter Ego, 2048, Killing Time, Santa's Workshop, Fireflies, Dodge, Evolution, Taxi, Chaos Theory, The Great Escape, Colony, The Tower, Nightmare Simulator, Mystical, Ashes, City Builder, Surge, Ultimate Arena, Solitaire, Castles, Invasion

```

[6]:      appid      name release_date  english \
2729  283580  New York Bus Simulator   2014-08-06      1
8227  446480  New York Bus Simulator   2016-03-04      1

      developer      publisher platforms \
2729      TML-Studios      Aerosoft GmbH  windows
8227  Little Freedom Factory  United Independent Entertainment GmbH  windows

      required_age  categories  genres  steamspy_tags \
2729      0  Single-player  Simulation  Simulation;Masterpiece;Driving
8227      0  Single-player  Simulation      Simulation

      achievements  positive_ratings  negative_ratings  average_playtime \
2729      0      29      35      0
8227      0      7      42      0

      median_playtime  owners  price

```

2729	0	0-20000	8.99
8227	0	0-20000	3.99

0.3.2 2) Target feature: owners

Here, since owners is categorical feature , we tried to use the median(also average here) to replace it.

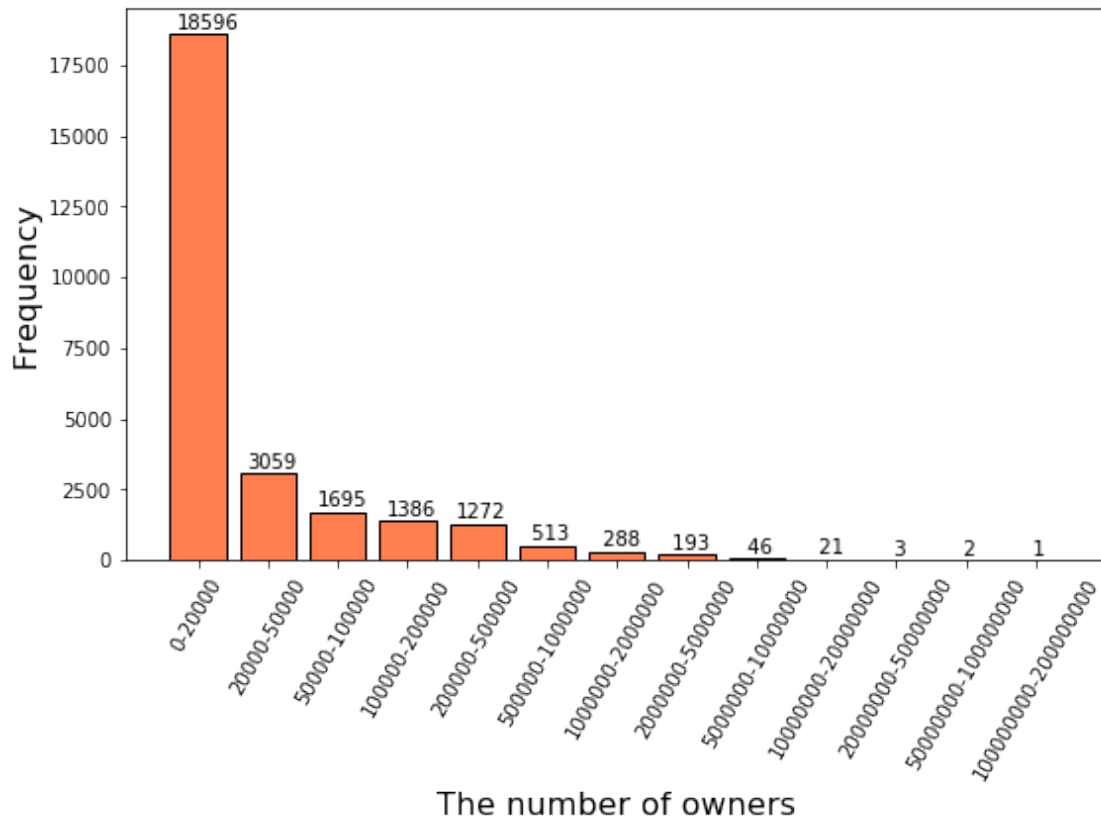
```
[7]: def getMedian(x):
      x_list = [ float(x) for x in x.split("-")]
      x_median = np.median(x_list)
      return x_median
```

```
[8]: df_prepare = df.copy()
      df_prepare["owners_median"] = df["owners"].apply(lambda x: getMedian(x))
```

Check the distribution of owners

```
[9]: df_owner_infor = { df_tmp[0]: len(df_tmp[1]) for df_tmp in df_prepare.
      ↳groupby("owners")}
      df_owner_infor = { x:y for x,y in sorted(df_owner_infor.items(), key = lambda x:
      ↳getMedian(x[0]))}
```

```
[10]: fig = plt.figure(figsize=[8,6])
      ax = plt.gca()
      plt.bar(df_owner_infor.keys(), df_owner_infor.values(), edgecolor="black",
      ↳color="coral")
      for idx, key in enumerate(df_owner_infor.keys()):
          ax.annotate("{:>4s}".format(str(df_owner_infor[key])), (idx-0.3,
          ↳df_owner_infor[key] + 200), fontsize=10)
      plt.xticks(rotation="60", fontsize=10)
      plt.tight_layout(pad=2)
      plt.xlabel("The number of owners", size=16)
      plt.ylabel("Frequency", size=16);
      #plt.savefig("Owners_distribution.jpg", dpi=300);
```



It is clearly that the owners of most games are in 0-20000 range, and some games have very high number owners, which are larger than 50 million. This results indicates that our dataset is unbalanced and should be processed carefully in the model development. Since there are only 27 games have owners larger than 10 million, we combine these games together and form the “larger than 10 million” class

```
[11]: ### Games with high onwers
df_prepare[df_prepare["owners"] == "100000000-200000000"]

[11]:   appid  name release_date  english developer publisher \
22    570  Dota 2   2013-07-09         1    Valve    Valve

      platforms  required_age \
22 windows;mac;linux         0

      categories \
22 Multi-player;Co-op;Steam Trading Cards;Steam W...

      genres  steamspy_tags  achievements \
22 Action;Free to Play;Strategy  Free to Play;MOBA;Strategy  0

      positive_ratings  negative_ratings  average_playtime  median_playtime \
22          863507          142079          23944          801
```

	owners	price	owners_median
22	100000000-200000000	0.0	150000000.0

0.3.3 3) Categorical features

There are many categorical features. Let's check if there is any significant trend in categorical features for successful games

```
[12]: def getTopN(infile, variable, sort_value="owners_median", TopN=5):
    """
    Get data grouped by certain features and return the TopN results ranking by
    ↪owners_median

    """
    infile_tmp = infile.groupby(variable).agg({"owners_median": np.mean,
    ↪"positive_ratings": np.mean, "average_playtime": np.mean, "median_playtime": np.
    ↪mean})

    return infile_tmp.sort_values(sort_value, ascending = False)[0:TopN]
```

```
[13]: def getDict(infile, variable):
    """
    Get data grouped by certain features and return the dictionary

    """
    infile_tmp = infile.groupby(variable).agg({"owners_median": np.mean,
    ↪"positive_ratings": np.mean, "average_playtime": np.mean, "median_playtime": np.
    ↪mean})

    return infile_tmp.to_dict()
```

```
[14]: def drawWordCloud(freq_dict):
    """
    Draw wordcloud for freq_dict
    """
    wc = wordcloud.WordCloud(background_color="white", max_font_size=100,
    ↪max_words=100, random_state=0).generate_from_frequencies(freq_dict)
    plt.figure(figsize=[8,6])
    plt.imshow(wc, interpolation="bilinear")
    plt.axis("off");
```

a. Developer

```
[15]: ##### Show Top 10 developer companies #####
getTopN(df_prepare, "developer")
```

```
[15]: owners_median positive_ratings \
developer
```

	average_playtime	median_playtime
developer		
PUBG Corporation	22938.000000	12434.000000
Valve;Hidden Path Entertainment	22494.000000	6502.000000
Smartly Dressed Games	3248.000000	413.000000
Valve	2663.538462	282.230769
Blue Mammoth Games	724.000000	146.000000

[illegible]**b. Publisher**

	owners_median	positive_ratings	average_playtime
publisher			
PUBG Corporation	75000000.0	496184.000000	22938.0
Digital Extremes	35000000.0	226541.000000	5845.0

Smartly Dressed Games	35000000.0	292574.000000	3248.0
Valve	17025000.0	175689.066667	3505.1
Grinding Gear Games	15000000.0	71593.000000	5263.0

	median_playtime
publisher	
PUBG Corporation	12434.0
Digital Extremes	394.0
Smartly Dressed Games	413.0
Valve	544.5
Grinding Gear Games	492.0

```
[18]: publisher_dict = getDict(df_prepare, "publisher")
drawWordCloud(publisher_dict["owners_median"])
```



c. Developer & Publisher

```
[19]: ##### Show Top 10 developer & publisher combinations #####
getTopN(df_prepare, ["developer", "publisher"])
```

		owners_median \
developer	publisher	
PUBG Corporation	PUBG Corporation	7.500000e+07
Valve;Hidden Path Entertainment	Valve	7.500000e+07
Smartly Dressed Games	Smartly Dressed Games	3.500000e+07
Digital Extremes	Digital Extremes	3.500000e+07
Valve	Valve	1.560577e+07

positive_ratings \

developer	publisher	
PUBG Corporation	PUBG Corporation	4.961840e+05
Valve;Hidden Path Entertainment	Valve	2.644404e+06
Smartly Dressed Games	Smartly Dressed Games	2.925740e+05
Digital Extremes	Digital Extremes	2.265410e+05
Valve	Valve	8.667212e+04

developer	publisher	average_playtime \
PUBG Corporation	PUBG Corporation	22938.000000
Valve;Hidden Path Entertainment	Valve	22494.000000
Smartly Dressed Games	Smartly Dressed Games	3248.000000
Digital Extremes	Digital Extremes	5845.000000
Valve	Valve	2663.538462

developer	publisher	median_playtime
PUBG Corporation	PUBG Corporation	12434.000000
Valve;Hidden Path Entertainment	Valve	6502.000000
Smartly Dressed Games	Smartly Dressed Games	413.000000
Digital Extremes	Digital Extremes	394.000000
Valve	Valve	282.230769

d. categories features with subclass For platform, categories, genres, steamspy_tags, each game has multiple types. To make the analysis easier, I first split them and get the vocabulary of them, then changed them into one-hot features, which can be used in future model development.

```
[20]: def getVocab(data, variable):
    """
    Get vocabulary for certain feature(variable)
    """
    vocab = {}
    for line in data[variable].tolist():
        if ";" in line:
            line = line.split(";")
        else:
            line = [line]
        for v in line:
            if v not in vocab:
                vocab[v] = 1
            else:
                vocab[v] += 1

    vocab = {x:y for x, y in sorted(vocab.items(), key = lambda x: x[1], reverse_
    => True)}

    return vocab
```

```
[21]: def splitCateg(data, variable, vocab):
    '''
    Split feature with its vocabulary and change it into onehot, and added split_
    →feature columns into initial data
    data: initial data
    variable: feature name
    vocab: its vocabulary
    '''
    v_split = {key:[] for key in vocab.keys()}
    for v in data[variable]:
        if ";" in v:
            v = v.split(";")
        else:
            v = [v]
        for key in v_split.keys():
            if key in v:
                v_split[key].append(1)
            else:
                v_split[key].append(0)

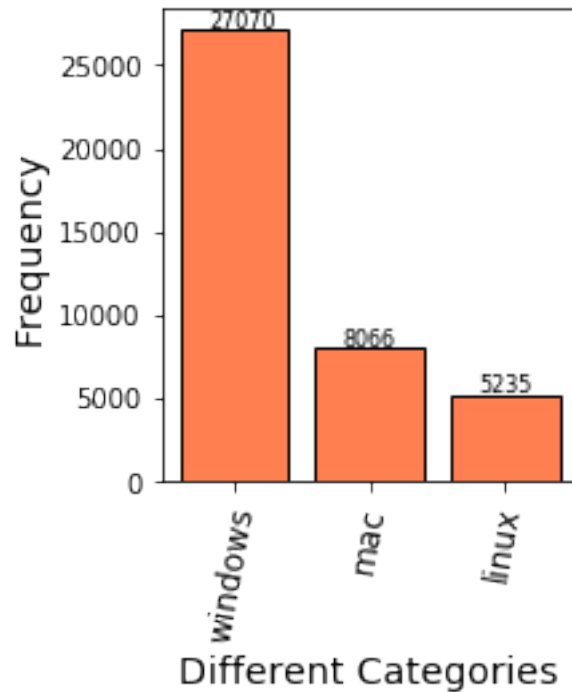
    v_data = pd.DataFrame(v_split)

    data_add = pd.merge(data, v_data , on = data.index)
    data_add.drop("key_0", axis = 1, inplace = True)

    return data_add
```

Platform

```
[22]: platform_vocab = getVocab(df_prepare, "platforms")
fig = plt.figure(figsize=[3,3])
ax = plt.gca()
plt.bar(platform_vocab.keys(),platform_vocab.values(), edgecolor="black",
→color="coral")
for idx, key in enumerate(platform_vocab.keys()):
    ax.annotate("{:>4s}".format(str(platform_vocab[key])), (idx-0.2,
→platform_vocab[key] + 100), fontsize=8)
plt.tight_layout()
plt.xticks(rotation="80", fontsize=12)
plt.xlabel("Different Categories", size=14)
plt.ylabel("Frequency", size=14);
```



```
[23]: ##### Add split onehot fetaures #####
df_prepare = splitCateg(df_prepare, "platforms", platform_vocab)
df_prepare.shape
```

[23]: (27075, 22)

categories

```
[24]: ### initial data, show the top 10 categories
getTopN(df_prepare, ["categories"])
```

```
[24]:
```

categories	owners_median \
Multi-player;Co-op;Steam Trading Cards;Steam Wo...	150000000.0
Multi-player;Steam Achievements;Full controller...	75000000.0
Multi-player;Online Multi-Player;Stats	37575000.0
Single-player;Online Multi-Player;Online Co-op;...	35000000.0
Multi-player;Cross-Platform Multiplayer;Steam A...	35000000.0

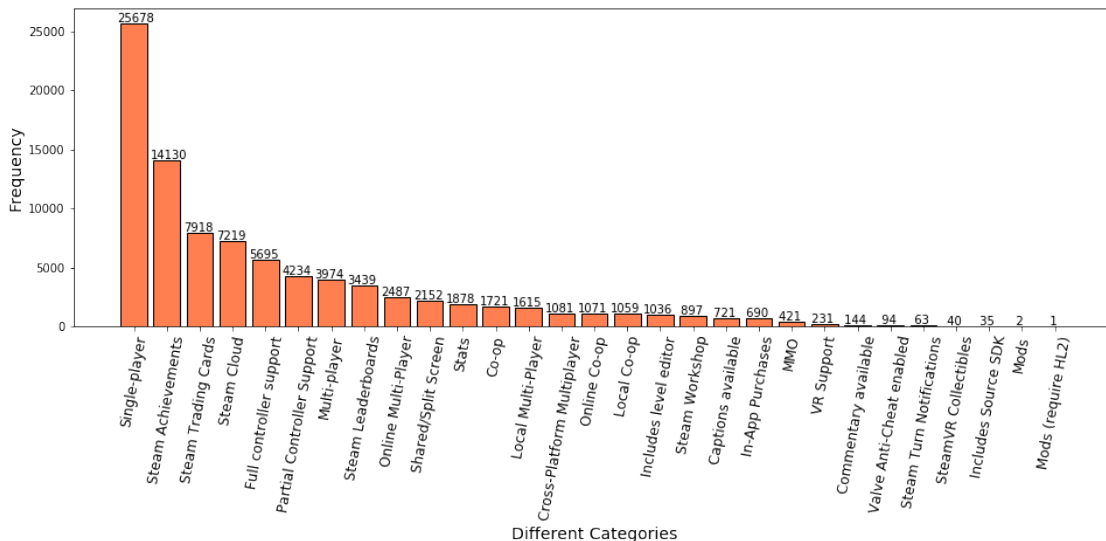
categories	positive_ratings \
Multi-player;Co-op;Steam Trading Cards;Steam Wo...	863507.0
Multi-player;Steam Achievements;Full controller...	2644404.0
Multi-player;Online Multi-Player;Stats	248303.5
Single-player;Online Multi-Player;Online Co-op;...	292574.0
Multi-player;Cross-Platform Multiplayer;Steam A...	515879.0

categories	average_playtime \
Multi-player;Co-op;Steam Trading Cards;Steam Wo...	23944.0
Multi-player;Steam Achievements;Full controller...	22494.0
Multi-player;Online Multi-Player;Stats	11502.5
Single-player;Online Multi-Player;Online Co-op;...	3248.0
Multi-player;Cross-Platform Multiplayer;Steam A...	8495.0

categories	median_playtime
Multi-player;Co-op;Steam Trading Cards;Steam Wo...	801.0
Multi-player;Steam Achievements;Full controller...	6502.0
Multi-player;Online Multi-Player;Stats	6250.5
Single-player;Online Multi-Player;Online Co-op;...	413.0
Multi-player;Cross-Platform Multiplayer;Steam A...	623.0

```
[25]: ##### Get categories vocabulary #####
cat_vocab = getVocab(df_prepare, "categories")
print("There are " + str(len(cat_vocab)) + " categories")
fig = plt.figure(figsize=[16,5])
ax = plt.gca()
plt.bar(cat_vocab.keys(),cat_vocab.values(), edgecolor="black", color="coral")
for idx, key in enumerate(cat_vocab.keys()):
    ax.annotate("{:>4s}".format(str(cat_vocab[key])), (idx-0.5, cat_vocab[key] +
    ↪100), fontsize=10)
plt.xticks(rotation="80", fontsize=12)
plt.xlabel("Different Categories", size=14)
plt.ylabel("Frequency", size=14);
```

There are 29 categories



```
[26]: ##### Add split categories features #####
df_prepare = splitCateg(df_prepare, "categories", cat_vocab)
```

Genres

```
[27]: ##### Initial genres #####
getTopN(df_prepare, ["genres"])
```

```
[27]:
```

	owners_median \
genres	
Action;Free to Play;Strategy	1.905688e+07
Action;Free to Play;Indie;Massively Multiplayer...	1.500000e+07
Action;Adventure;Massively Multiplayer	1.311167e+07
Action;Adventure;Free to Play;Massively Multipl...	7.675000e+06
Action;Adventure;Free to Play;Simulation;Sports	7.500000e+06

	positive_ratings \
genres	
Action;Free to Play;Strategy	108938.000000
Action;Free to Play;Indie;Massively Multiplayer...	80360.000000
Action;Adventure;Massively Multiplayer	102626.833333
Action;Adventure;Free to Play;Massively Multipl...	55847.500000
Action;Adventure;Free to Play;Simulation;Sports	11440.000000

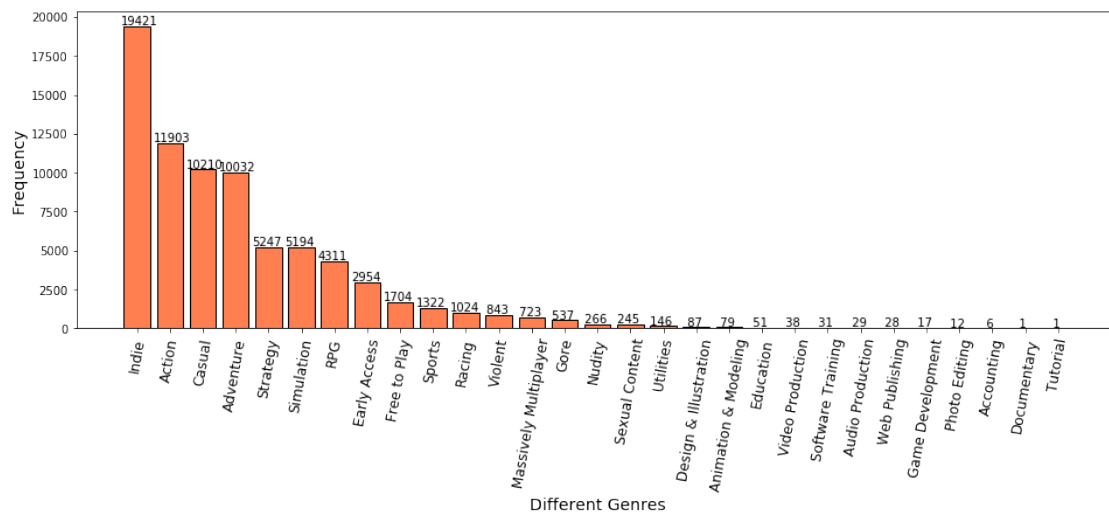
	average_playtime \
genres	
Action;Free to Play;Strategy	3156.875
Action;Free to Play;Indie;Massively Multiplayer...	1369.000
Action;Adventure;Massively Multiplayer	5141.000
Action;Adventure;Free to Play;Massively Multipl...	1589.000
Action;Adventure;Free to Play;Simulation;Sports	163.000

	median_playtime
genres	
Action;Free to Play;Strategy	325.25
Action;Free to Play;Indie;Massively Multiplayer...	211.00
Action;Adventure;Massively Multiplayer	2654.00
Action;Adventure;Free to Play;Massively Multipl...	99.50
Action;Adventure;Free to Play;Simulation;Sports	29.00

```
[28]: ##### Get genres vocabulary #####
genres_vocab = getVocab(df_prepare, "genres")
print("There are " + str(len(genres_vocab)) + " genres")
fig = plt.figure(figsize=[16,5])
ax = plt.gca()
```

```
plt.bar(genres_vocab.keys(),genres_vocab.values(), edgecolor="black",
        color="coral")
for idx, key in enumerate(genres_vocab.keys()):
    ax.annotate("{:>4s}".format(str(genres_vocab[key])), (idx-0.5,
        genres_vocab[key] + 100), fontsize=10)
plt.xticks(rotation="80", fontsize=12)
plt.xlabel("Different Genres", size=14)
plt.ylabel("Frequency", size=14);
```

There are 29 genres



```
[29]: ##### Add split genres features #####
df_prepare = splitCateg(df_prepare, "genres", genres_vocab)
```

Steam tags

```
[30]: ##### Initial top 10 steamspy_tags #####
getTopN(df_prepare, "steamspy_tags")
```

```
[30]:
```

steamspy_tags	owners_median	positive_ratings \
Free to Play;MOBA;Strategy	75375000.0	432730.5
Survival;Shooter;Multiplayer	75000000.0	496184.0
FPS;Multiplayer;Shooter	75000000.0	2644404.0
Free to Play;Survival;Zombies	35000000.0	292574.0
Free to Play;Action;Co-op	35000000.0	226541.0

steamspy_tags	average_playtime	median_playtime
Free to Play;MOBA;Strategy	11986.0	417.0

Survival;Shooter;Multiplayer	22938.0	12434.0
FPS;Multiplayer;Shooter	22494.0	6502.0
Free to Play;Survival;Zombies	3248.0	413.0
Free to Play;Action;Co-op	5845.0	394.0

```
[31]: ##### Get genres vocabulary #####
steamspy_vocab = getVocab(df_prepare, "steamspy_tags")
print("There are " + str(len(steamspy_vocab)) + " steamspy_tags")
```

There are 339 steamspy_tags

```
[32]: ##### Add split genres features #####
df_prepare = splitCateg(df_prepare, "steamspy_tags", steamspy_vocab)
df_prepare.shape
```

[32]: (27075, 419)

Check Game Type Effect on Successful Games

```
[33]: ##### For total games #####
fig = plt.figure(figsize=[40,10])
fig.suptitle("Total Games", fontsize=50)
list_vcab = [cat_vocab, genres_vocab, steamspy_vocab]
titles = ["categories", "genres", "steamspy_tags"]
for i in range(1,4):
    ax = fig.add_subplot(1,3,i)
    wc = wordcloud.WordCloud(background_color="white", max_font_size=50,
    ↪random_state=0).generate_from_frequencies(list_vcab[i-1])
    plt.imshow(wc,interpolation="bilinear")
    plt.axis("off")
    ax.set_title(titles[i-1], size=50)
plt.tight_layout()
plt.show();
```



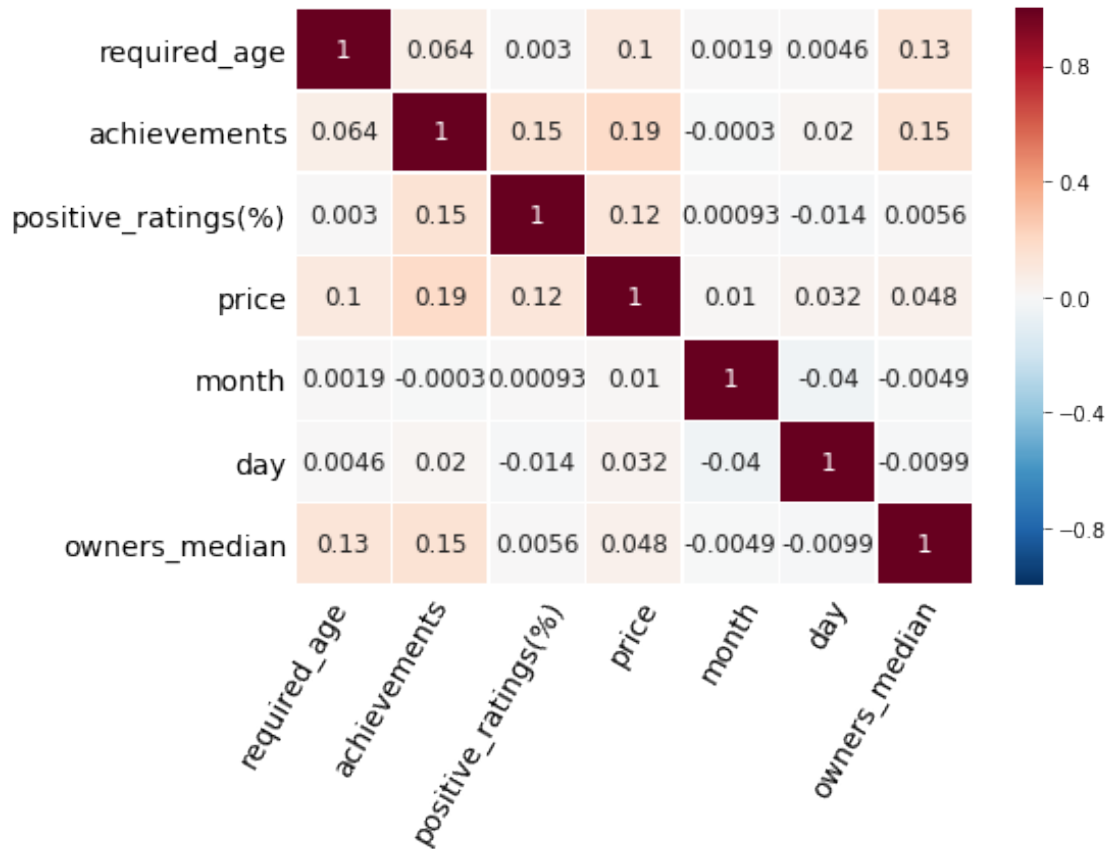
```
[34]: ##### For successful games (owners > 1M ) #####
df_prepare_success = df_prepare[df_prepare["owners_median"] >= 1500000]
success_cat_vocab = {key:sum(df_prepare_success[key + "_x"]) if key in
    ↪ steamspy_vocab.keys() else sum(df_prepare_success[key]) for key in cat_vocab}
```



```
return ref_year, ref_month, ref_day
```

```
[37]: year = []  
month = []  
day = []  
for i in range(df_prepare.shape[0]):  
    y,m,d = splitTime(df_prepare.loc[i,"release_date"])  
    year.append(y)  
    month.append(m)  
    day.append(d)  
df_prepare["year"] = year  
df_prepare["month"] = month  
df_prepare["day"] = day
```

```
[38]: ### Final correlation graph, should use data after adding release date  
→information  
figure = plt.figure(figsize=[8,6])  
ax = plt.gca()  
seaborn.heatmap(df_prepare[["required_age", "achievements",  
    →"positive_ratings_percentage", "price", "month", "day", "owners_median"]].  
    →corr(method="spearman"), cmap="RdBu_r", annot=True, annot_kws={"size": 12},  
    →vmin=-1, linewidths=0.5)  
plt.xticks([0,1,2,3.5,4.5,5.5,6],["required_age", "achievements",  
    →"positive_ratings(%)", "price", "month", "day", "owners_median"], size=14,  
    →rotation="60")  
plt.yticks([0.5,1.5,2.5,3.5,4.5,5.5,6.5],["required_age", "achievements",  
    →"positive_ratings(%)", "price", "month", "day", "owners_median"], size=14)  
ax.tick_params(axis='both', which='both', length=0)  
plt.tight_layout();  
#plt.savefig("correlation.png", dpi=300);
```



Currently, it seems no feature shows good correlation with owners_median, which indicates the challenge of this project.

0.3.5 5) New features: Developer_famous & Publisher_famous

Developer & Publisher For the developer and publisher, since there are two many different categorials, we converted them into two new features: famous and non-famous, based on previous owners. These two features indicate the developer/publisher reputation effect on game success. If a developer has average owners higher than 75% of developers before the target game release date, we assume it's a famous developer. If a publisher has average owners higher than 75% of publishers before the target game release date, we assume it's a famous publisher.

```
[39]: def time_filter(data, ref_date):
    """
    Remove games released after ref_date
    """
    ref_year, ref_month, ref_day = time.strptime(ref_date, '%Y-%m-%d').tm_year, \
    →time.strptime(ref_date, '%Y-%m-%d').tm_mon, time.strptime(ref_date, '%Y-%m-%d').
    →tm_mday
    data.release_date = data["release_date"].astype("datetime64")
    df_tmp = data[(data["release_date"].dt.year < ref_year) |
```

```

        ((data["release_date"].dt.year == ref_year) &
→(data["release_date"].dt.month < ref_month)) |
        ((data["release_date"].dt.year == ref_year) &
→(data["release_date"].dt.month == ref_month) & (data["release_date"].dt.day <
→ref_day))
    ]
    return df_tmp

```

```

[40]: def checkFamous(data, variable, ref_idx):
    """
    Check the publisher/developer of certain game is famous or not (based on
→ref_idx) before game release.
    The assumption here, is the owners of certain games always increase
→significantly in a short time after game release.
    Based on this assumption, we can determine whether a company is famous or
→not using this simple function.
    However, this assumption might be not True.
    We can also use other information, such as metacritic score to determine
→whether a company is famous or not
    """
    df_tmp = data[data.index != ref_idx]
    ref_date = data.iloc[ref_idx]["release_date"]
    ref_variable = data.iloc[ref_idx][variable]
    df_tmp = time_filter(df_tmp, ref_date)
    df_tmp_group = df_tmp.groupby(variable).agg({"owners_median": np.mean})
    df_tmp_group.reset_index(inplace=True)
    value = df_tmp_group.describe().loc["75%"].values[0]
    if df_tmp_group[df_tmp_group[variable] == ref_variable]["owners_median"].
→values >= value:
        return True
    else:
        return False

```

```

[41]: def addFamous(data, variable):
    """
    Get famous data and add them into initial data
    """
    out = open("../data/famous_" + variable + ".csv", "w")
    famous_list = []
    for i in range(data.shape[0]):
        famous = checkFamous(data, variable, i)
        if famous:
            famous_list.append(1)
        else:
            famous_list.append(0)
    print(str(i) + "," + str(famous))
    out.write(str(famous) + "\n")

```

```

out.close()
#data[variable + "_famous"] = famous_list

famous_percentage = len([i for i in famous_list if i == 1]) /
→len(famous_list)
print("Famous percentage is " + str(famous_percentage))
return data

```

```

[ ]: pd.options.mode.chained_assignment = None
df_prepare = addFamous(df_prepare, "developer")

```

```

[ ]: df_prepare = addFamous(df_prepare, "publisher")

```

```

[42]: df_famous_developer = pd.read_csv("../data/famous_developer.csv", header=None)
developer_famous = [1 if i == True else 0 for i in df_famous_developer[0].
→tolist()]
df_famous_publisher = pd.read_csv("../data/famous_publisher.csv", header=None)
publisher_famous = [1 if i == True else 0 for i in df_famous_publisher[0].
→tolist()]

```

```

[43]: df_prepare["developer_famous"] = developer_famous
df_prepare["publisher_famous"] = publisher_famous

```

0.4 3. Build Model

Since there are too less games with owners larger than 10 million, we combined the games with owners larger than 10 million and use the average number to replace the initial owners_media.

```

[44]: df_use = df_prepare.copy()
[45]: for idx in df_use.loc[df_use["owners"].isin(["100000000-200000000",
→"500000000-1000000000", "200000000-500000000", "100000000-200000000"])] .index:
df_use.iloc[idx, 18] = 26666667

```

26666667 is the average of owners_median of these games

```

[46]: ##### Change number into classes #####
Y = df_use[["owners_median"]]
ec = LabelEncoder()
df_use["target"] = ec.fit_transform(Y)

```

```

/Users/jianinglu1/anaconda3/lib/python3.7/site-
packages/sklearn/preprocessing/label.py:235: DataConversionWarning: A column-
vector y was passed when a 1d array was expected. Please change the shape of y
to (n_samples, ), for example using ravel().
y = column_or_1d(y, warn=True)

```

```

[47]: ##### Split dataset using 8:1:1 #####
train_idx, test_idx = train_test_split(df_use["appid"].values, test_size=0.10,
→stratify=df_use["target"], random_state=0)

```

```
train_idx, val_idx = train_test_split(train_idx, test_size=0.11,
    ↳stratify=df_use[df_use["appid"].isin(train_idx)]["target"], random_state=0)
```

```
[48]: train_idx.shape
      test_idx.shape
      val_idx.shape
```

```
[48]: (21686,)
```

```
[48]: (2708,)
```

```
[48]: (2681,)
```

We first trained a baseline model with all features(no time features) and simple XGB classifier, and found there are some features with 0 feature importance. To accelerate the model training process, we retrained the model without features, and includes time features(month and day). We also added class weight for each class.

```
[49]: ##### Remove non-important features #####
features = ['english', 'required_age', 'achievements', 'price',
            'positive_ratings_percentage', 'windows', 'mac', 'linux',
    ↳'developer_famous', 'publisher_famous'] + \
            list([key if key not in steamspy_vocab.keys() else key + "_x" for
    ↳key in cat_vocab.keys()]) + \
            list([key if key not in steamspy_vocab.keys() else key + "_x" for
    ↳key in genres_vocab.keys()]) + \
            list([key + "_y" if (key in cat_vocab.keys()) or (key in
    ↳genres_vocab.keys()) else key for key in steamspy_vocab.keys()])

features_imp = pd.read_csv("../models/Base_line/feature_imp.csv", header = None,
    ↳index_col=0)

features_imp.columns = features
mean_features_imp = features_imp.describe()[1:2].transpose()
mean_features_imp.sort_values("mean", inplace = True)
features_remove = [i for i in features if i not in
    ↳mean_features_imp[mean_features_imp["mean"] == 0].index.tolist()]
features_remove = features_remove + ["month", "day"]
print(str(len(features_remove)) + " features")
```

217 features

```
[50]: X_train = df_use[df_use["appid"].isin(train_idx)][features_remove].values
      Y_train = df_use[df_use["appid"].isin(train_idx)]["target"].values
      ##### sample weight is based on its target class
      Y_train_weight = (Y_train + 1)/np.max(Y_train + 1)
      X_val = df_use[df_use["appid"].isin(val_idx)][features_remove].values
      Y_val = df_use[df_use["appid"].isin(val_idx)]["target"].values
      Y_val_weight = (Y_val + 1)/np.max(Y_val + 1)
      X_test = df_use[df_use["appid"].isin(test_idx)][features_remove].values
      Y_test = df_use[df_use["appid"].isin(test_idx)]["target"].values
```

We use hold out validation set, which is split based on target classes. We also applied ensemble model to do the prediction.

0.4.1 1) Train Model

```
[51]: trnp_df = pd.DataFrame({"index": [i for i in range(X_train.shape[0])]})
tesp_df = pd.DataFrame({"index": [i for i in range(X_test.shape[0])]})
valp_df = pd.DataFrame({"index": [i for i in range(X_val.shape[0])]})
out = open("../models/Final/feature_imp.csv", "w")
for i in range(10):
    if "pima.pickle_" + str(i) + ".dat" in os.listdir("../models/Final/"):
        xgb = pickle.load(open("../models/Final/pima.pickle_" + str(i) + ".dat",
                                "rb"))
    else:
        xgb = XGBClassifier(n_estimators=1000, random_state=(i-1) * 10,
                             colsample_bytree=0.8, learning_rate=0.2, objective="multi:softmax",
                             num_class=10)
        xgb.fit(X_train, Y_train, sample_weight=Y_train_weight,
                eval_set=[(X_val, Y_val)], eval_metric="mlogloss", early_stopping_rounds=50,
                sample_weight_eval_set=[Y_val_weight])
        pickle.dump(xgb, open("../models/Final/pima.pickle_" + str(i) + ".dat",
                                "wb"))
        f_importance = xgb.feature_importances_
        out.write(str(i) + "," + ",".join([str(f) for f in f_importance]) + '\n')
        trnp = xgb.predict(X_train, ntree_limit=xgb.best_ntree_limit)
        tesp = xgb.predict(X_test, ntree_limit=xgb.best_ntree_limit)
        valp = xgb.predict(X_val, ntree_limit=xgb.best_ntree_limit)
        if i == 1:
            trnp_df = pd.DataFrame({"data" + str(i): trnp})
            tesp_df = pd.DataFrame({"data" + str(i): tesp})
            valp_df = pd.DataFrame({"data" + str(i): valp})
        else:
            trnp_df["data" + str(i)] = trnp
            tesp_df["data" + str(i)] = tesp
            valp_df["data" + str(i)] = valp
out.close()
```

[51]: 2632

[51]: 2655

[51]: 2655

[51]: 2601

[51]: 2601

[51]: 2663

[51]: 2652

[51]: 2655

[51]: 2637

[51]: 2664

```
[52]: trnp_df_final = trnp_df.mode(axis = 1)[0]
      tesp_df_final = tesp_df.mode(axis = 1)[0]
      valp_df_final = valp_df.mode(axis = 1)[0]
```

0.4.2 2) Check Performance

```
[53]: def getAccuracy(Y_true, Y_pred):
      '''
      Get prediction accuracy
      '''
      acc = (Y_pred == Y_true).sum().astype(float) / len(Y_true)*100
      return str(round(acc,2)) + "%"

      def calTPR(predict, target, threshold):
          '''
          Calculate true positive rate
          '''
          tesFP_class = (lambda x: [1 if i >= threshold else 0 for i in x])(predict)
          tesFY_class = (lambda x: [1 if i >= threshold else 0 for i in x])(target)
          TP = [ idx for idx, i in enumerate(tesFY_class) if (i == 1) and
          →(tesFP_class[idx] == 1) ]
          TotalP = [ idx for idx, i in enumerate(tesFY_class) if i == 1]
          TPR = len(TP) / len(TotalP)
          return TPR

          def calFPR(predict, target, threshold):
              '''
              Calculate false positive rate
              '''
              tesFP_class = (lambda x: [1 if i >= threshold else 0 for i in x])(predict)
              tesFY_class = (lambda x: [1 if i >= threshold else 0 for i in x])(target)
              FP = [ idx for idx, i in enumerate(tesFY_class) if (i == 0) and
              →(tesFP_class[idx] == 1) ]
              TotalN = [ idx for idx, i in enumerate(tesFY_class) if i == 0]
              FPR = len(FP) / len(TotalN)
              return FPR

      acc_train = getAccuracy(Y_train, trnp_df_final)
      acc_val = getAccuracy(Y_val, valp_df_final)
      acc_test = getAccuracy(Y_test, tesp_df_final)
      print("Prediction accuracy:\n" + "Train:" + acc_train + "," + "Validation:" +
      →acc_val + "," + "Test:" + acc_test )
```


Prediction accuracy:
Train:80.51%,Validation:69.56%,Test:68.5%

```
[55]: results_train = calTPR(trnp_df_final, Y_train, 1), calFPR(trnp_df_final,
    ↳Y_train, 1)
results_val = calTPR(valp_df_final, Y_val, 1), calFPR(valp_df_final, Y_val, 1)
results_test = calTPR(tesp_df_final, Y_test, 1), calFPR(tesp_df_final, Y_test, 1)
print("TPR:\n" + "Train:" + str(results_train[0]) + ",Validation:" +
    ↳str(results_val[0]) + ",Test:" + str(results_test[0]))
print("FPR:\n" + "Train:" + str(results_train[1]) + ",Validation:" +
    ↳str(results_val[1]) + ",Test:" + str(results_test[1]))
```

TPR:
Train:0.748377581120944,Validation:0.7027027027027027,Test:0.6851415094339622
FPR:
Train:0.0779551858312089,Validation:0.08469945355191257,Test:0.1

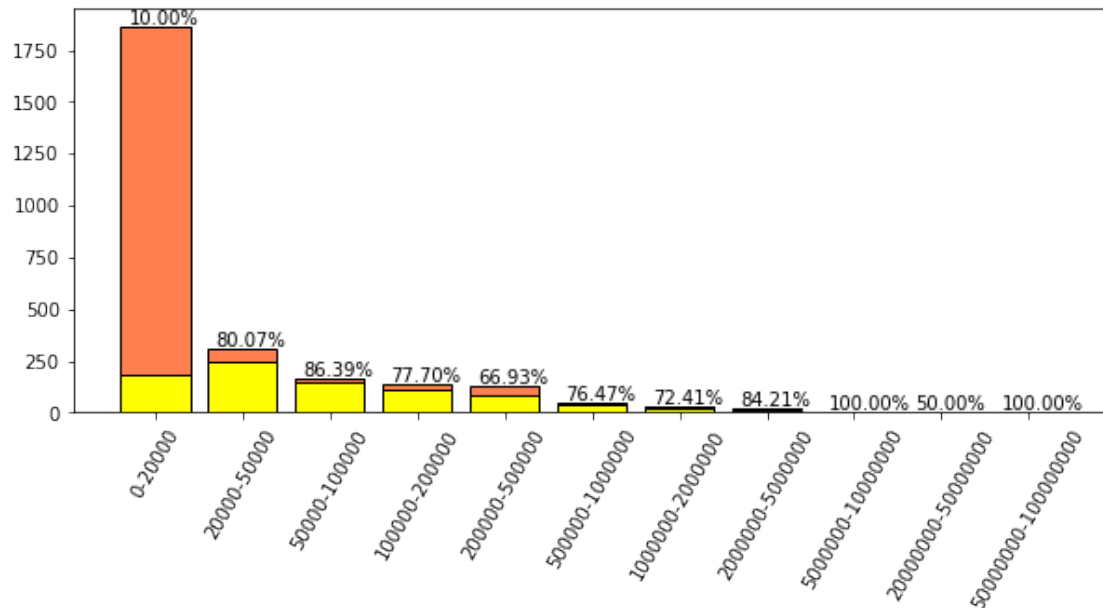
0.4.3 3) Error Analysis

```
[56]: df_test = df_use[df_use["appid"].isin(test_idx)]
    ### Initial target class distribution ###
df_test_owner_infor = { df_tmp[0]: len(df_tmp[1]) for df_tmp in df_test.
    ↳groupby("owners")}
df_test_owner_infor = { x:y for x,y in sorted(df_test_owner_infor.items(), key =
    ↳lambda x: getMedian(x[0]))}
df_test_owner_infor
df_test["predict"] = tesp_df_final.values
df_test_error = df_test[df_test["predict"] != df_test["target"]]
df_test_error_owner_infor = { df_tmp[0]: len(df_tmp[1]) for df_tmp in
    ↳df_test_error.groupby("owners")}
df_test_error_owner_infor = { x:y for x,y in sorted(df_test_error_owner_infor.
    ↳items(), key = lambda x: getMedian(x[0]))}
fig = plt.figure(figsize=[10,4])
ax = plt.gca()
plt.bar(df_test_owner_infor.keys(), df_test_owner_infor.values(),
    ↳edgecolor="black", color="coral")
plt.bar(df_test_error_owner_infor.keys(), df_test_error_owner_infor.values(),
    ↳edgecolor="black", color="yellow")

for idx, key in enumerate(df_test_owner_infor.keys()):
    ax.annotate("{:.2f}%".format(float(df_test_error_owner_infor[key]/
    ↳df_test_owner_infor[key] * 100)), (idx-0.3, df_test_owner_infor[key] + 10),
    ↳fontsize=10)
plt.xticks(rotation="60", fontsize=10);
#plt.xlabel("The number of owners (test set)", size=16)
#plt.ylabel("Frequency", size=16);
```

```
/Users/jianinglu1/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:6:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead
```

See the caveats in the documentation: <http://pandas.pydata.org/pandas-docs/stable/indexing.html#indexing-view-versus-copy>



```
[58]: def plot_confusion_matrix(y_true, y_pred, classes,
                                normalize=False,
                                title=None,
                                cmap=plt.cm.Blues):

    """
    This function prints and plots the confusion matrix.
    Normalization can be applied by setting `normalize=True`.
    """
    if not title:
        if normalize:
            title = 'Normalized confusion matrix'
        else:
            title = 'Confusion matrix, without normalization'

    # Compute confusion matrix
    cm = confusion_matrix(y_true, y_pred)
    # Only use the labels that appear in the data
    classes = classes[unique_labels(y_true, y_pred)]
```

```

if normalize:
    cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
    print("Normalized confusion matrix")
else:
    print('Confusion matrix, without normalization')
fig, ax = plt.subplots(figsize=[8,6])
im = ax.imshow(cm, interpolation='nearest', cmap=cmap)
ax.figure.colorbar(im, ax=ax)
# We want to show all ticks...
ax.set(xticks=np.arange(cm.shape[1]), yticks=np.arange(cm.shape[0]),
       # ... and label them with the respective list entries
       xticklabels=classes, yticklabels=classes,
       #title=title,
       )
ax.set_ylabel(ylabel='True label', fontsize=14)
ax.set_xlabel(xlabel='Predicted label', fontsize=14)
# Rotate the tick labels and set their alignment.
plt.setp(ax.get_xticklabels(), rotation=45, ha="right",
         rotation_mode="anchor", fontsize=12)
plt.setp(ax.get_yticklabels(), fontsize=12)

# Loop over data dimensions and create text annotations.
fmt = '.2f' if normalize else 'd'
thresh = cm.max() / 2.
for i in range(cm.shape[0]):
    for j in range(cm.shape[1]):
        ax.text(j, i, format(cm[i, j], fmt),
                ha="center", va="center", size=12,
                color="white" if cm[i, j] > thresh else "black")
fig.tight_layout(pad=0.0001);

return ax

```

```

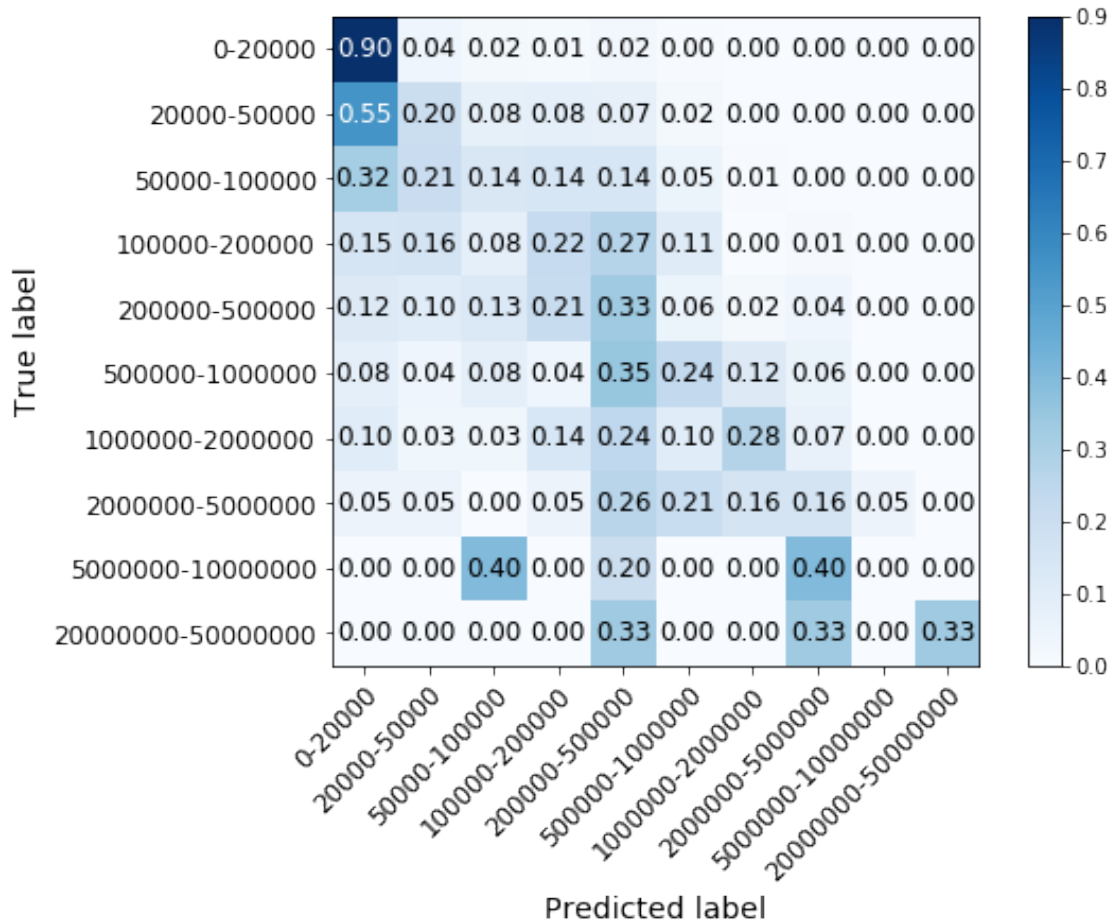
[59]: from sklearn.metrics import confusion_matrix
      from sklearn.utils.multiclass import unique_labels

      #confusion_matrix(Y_test, tesp_df_final)
      #np.set_printoptions(precision=3)
      # Plot normalized confusion matrix
      plot_confusion_matrix(Y_test, tesp_df_final, classes=np.array([i for i in_
      ↪df_test_owner_infor.keys()]), normalize=True, title='Normalized confusion_
      ↪matrix')
      #plt.savefig("Normalized_confusion_matrix.jpg", dpi=300);

```

Normalized confusion matrix

[59]: <matplotlib.axes._subplots.AxesSubplot at 0x122bdb908>



0.4.4 4) Feature Importance

```
[60]: features_imp = pd.read_csv("../models/Final/feature_imp.csv", header = None,
    ↪index_col=0)
features_imp.columns = features_remove
mean_features_imp = features_imp.describe()[1:2].transpose()
mean_features_imp.sort_values("mean", inplace = True)
mean_features_imp[mean_features_imp["mean"] == 0].index
```

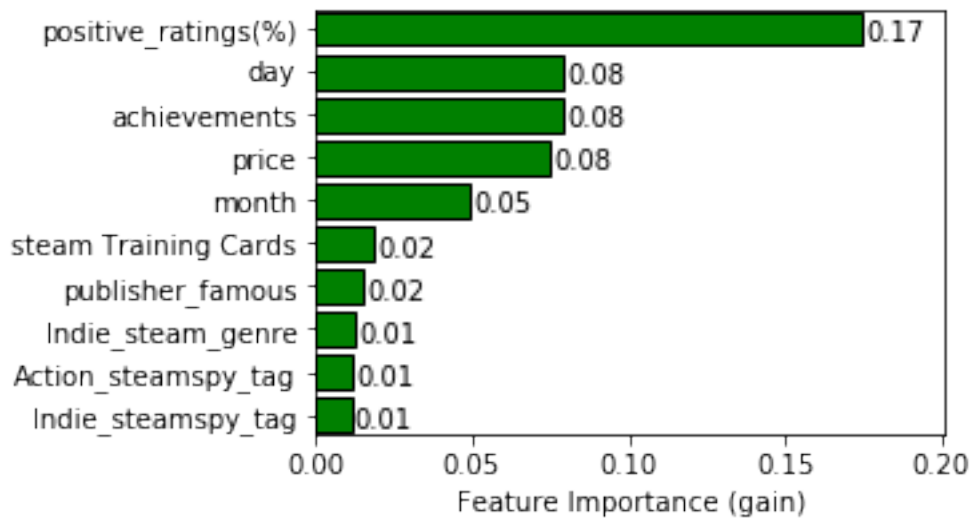
```
[60]: Index(['Rogue-lite', 'Wargame', 'Short', 'Chess', 'Cats', 'Underwater',
    'Turn-Based Tactics', 'Offroad', 'Cold War', 'Procedural Generation',
    'Video Production_y', 'Photo Editing_y', 'Cult Classic',
    'Real-Time with Pause', 'Magic', 'Photo Editing_x'],
    dtype='object')
```

```
[62]: fig = plt.figure(figsize=[6,3])
ax = plt.gca()
```

```

plt.barh(mean_features_imp.index[-10:], mean_features_imp["mean"][-10:],
        color="green", edgecolor="black")
for idx, x in enumerate(mean_features_imp["mean"][-10:]):
    ax.annotate("{:.2f}".format(x), (x + 0.001, idx-0.3))
plt.tight_layout(pad=2.0)
plt.yticks([9,8,7,6,5,4,3,2,1,0], ["positive_ratings(%)", "day", "achievements",
    "price", "month", "steam Training Cards", "publisher_famous",
    "Indie_steam_genre", "Action_steamspy_tag", "Indie_steamspy_tag"])
plt.margins(x=0.15, y = 0.005)
plt.xlabel("Feature Importance (gain)");
#plt.savefig("Feature_importance.png", dpi=300);

```



From the feature importance results, we found that the Top1 important features is positive_rating_percentage, which can be got from a pre-releasing game test.