STEAMULATOR

Predicting video game preferences from steam reviews

Hao-Yuan Chen (hyc404)

I-Wu Lu (iwl210)

Shashank K Pawar (skp363)

9th May 2017

Introduction to Data Science

with

Prof. Arthur Spirling

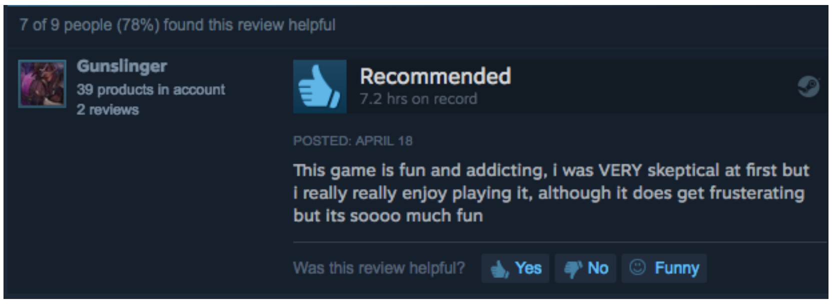
Spring 2017

Contents

1 Introduction

Steam is a digital game distribution platform offering over 11,800 video games for downloads. In 2015, the platform generated a revenue of 3.5 billion from its 125 million users. Despite its overwhelming financial success, Steam faces a similar problem as Netflix, of curating content for their users. With around 12000 games to choose from, recommending games to users becomes an important feature to guide them in their decision making process.

Currently, Steam uses a recommendation system based on tags associated with game genres. It recommends new games based on the games that already exist in a user's library. Our project takes a different approach for recommending games to the users. It analyzes data from steam reviews and user profiles to predict a user's preferences for recommending video games. This approach utilizes multiple data sources rather than limiting itself to the current Steam approach of using a user's game library. Our goal with this project is to include more data points to enhance the accuracy of predicting user's recommendations and thus improve the efficiency of recommending games through Steam.


7 of 9 people (78%) found this review helpful

Gunslinger
39 products in account
2 reviews

Recommended
7.2 hrs on record

POSTED: APRIL 18

This game is fun and addicting, i was VERY skeptical at first but i really really enjoy playing it, although it does get frusterating but its soooo much fun

Figure 1: Example of a Steam review

The data science approach to this problem is to build more coherent user models for making predictions. Utilizing user and game data and analyzing patterns of recommending games can be used to train a model to predict user recommendations and hence identify features that contribute to these recommendations. The factors that determine recommendations differ from user to user. However, it is possible to find patterns across users and manipulate the recommendation system for each user based on their history of recommendations. Although this is a more costly process, there is a huge business value in doing so. To put things in perspective, even a small improvement of 2-5% in prediction accuracy can lead to an additional revenue worth millions of dollars. In addition, the cost of false positives and true negatives is very low for our model. In other words, recommending a game that a user doesn't like or not recommending a game a user likes, will only have minor consequences (if any) to the company. As a result, exploring different approaches for recommending games to Steam users is a worthwhile effort and is exactly what this project tries to achieve.

2 Data Source

For this project, we had two main data sources. Our first source of data was the Steam website. As Steam does not provide a web API for loading review data from the database, we used a parser to retrieve information on the Steam website. Steam uses AJAX(Asynchronous JavaScript and XML) to reload new source into web pages, so we used *selenium* module to simulate the "scroll down" action to reach the bottom of the webpage. *driver.execute_ script("window.scrollTo(0, document.body.scrollHeight);")* will execute the script to scroll web page to bottom. After digging into Steam's html source, we found it helpful to use *driver.find_ element_ by_ link_ text('text')* to navigate the *div* we need. Furthermore, we used *beautifulsoup* module to parse the content from different html tags. For example, *soup.findAll("div", "class":"hours")* will give us all the *divs* which contain playing hours. In this way, we can scrape all the information in the html source. Table (1) is an example instance in our review csv file.

The second data source was a Steam Web API for user profile data. Using

Table 1: Review Example

User name	**Pos/neg**	**hours**	**Number of helpful**	**content**
HikaruJoe	1	1.3	11 of 17 people (65%)	2015 not bad ∼

Table 2: User Instance Example

User name	Level	Action	Adv.	Racing	RPG	Sim.	Sports	Str.
ajpos	33	224	72	4	40	33	2	60

this API, we retrieved user information to complement our review and game data. We also used a third-party *SteamAPI* based on the original one which is easier to use. It can generate a *SteamUser* object by taking user profile's url as a parameter. The object has many attributes such as *SteamUser.level* returns user's level, *SteamUser.games* returns a list of games the user has. By going through the list and calculating all the games, we can get the complete collection of games in user. Table (2) is an example instance in our user csv file.

3 Data Processing

To make our model faster, we transformed our csv files into the following dictionaries.

1. **Classification**: describes genres (key) and the corresponding game_id (value)
2. **Reviews**: contains properties of each review, such as content, game_id, user_id, etc. We use review_id (key) and a dictionary (value) to maintain all information of this review.
3. **Games_user**: contains the relationship of a certain game-user pair. Again, we use a layered dictionary to include this information. For example, the field *total_playtime* hold the information of how long the user spends on this game.
4. **User**: contains information from user profiles, such as user level and genres preferred.

The target variable for our prediction model is **'recommend'**. The variable explains whether a user recommended a game or not. We extracted this variable from the review dictionary and transformed it into a dummy variable with values 0 or 1 for binary classification.

The features in our models are extracted from the dictionaries mentioned above. There are three basic categories of features in our model.

1. User specific: These features include data related to the user independent of the game under consideration. This first category includes features 'user_genre_collection', 'user_steam_level' and 'user_total_reviews'.
2. Game specific: These features are independent of particular users reviewing the game. This category includes features 'game_genre', 'total_game_reviews' and 'avg_game_time'.
3. The third category includes feature that are specific for a given user reviewing a given game. This category includes features 'user_game_time', 'user_genre_time', 'user_game_level' and 'user_comment_emotions'.

A small group of features in the third category are extracted via the content of the review using natural language processing (NLP). This group contains relative rating of the positive and subjective scores of the review content. We educe the scores using well-known library "TextBlob" and list all scores of users and return the normalized z-score of the certain user. The following is an example illustrates how these two features work.If we are considering game G and user U, the scores of reviews given by U are [A: 24, B: 26, C: 21, D: 22, G: 25], given by TextBlob. Then we will return the value of

$$\frac{25-\mu}{\sigma} = \frac{25-23.6}{3.44} = 0.7548 \tag{1}$$

The formula includes not only the information from words, but also the relative level where this review is among all the games played by U The basic concept can be applied into the 'helpful' field as well, but in different direction. In short, we now consider the relative level where this review is among all users who play game G. The reason of this difference implies some key domain knowledge in

Table 3: Performance of Baseline Algorithm

True value	**precision**	**recall**	**f1 score**	**AUC**
0	0.46	0.49	0.48	0.6322
1	0.79	0.77	0.78	

user-game ratings. When we evaluate how positive a review is, the score is highly related to the reviewer; when we evaluate how helpful a review is, the score is more likely to be game-dependent.

4 Model and Extraction

As shown Sec.(3), we used a more sophisticated method which includes features with information of specific game/user pairs, such as natural language processing on reviews. To shown how the improvement of our models, we first consider a simpler model without all these features as the baseline model.

4.1 Baseline Model

An intuitive model on this classification problem is decision tree. We have implemented the basic decision tree and the code is listed in the Appendix. The results are shown in Table (3).

One can easily tell from this table that decision tree tends to predict the results to be 1, which means 'recommend', in this imbalanced dataset (The ratio of true 1/ true 0 is over 7/3). Therefore, even if it has fair good accuracy (overall = 0.7), there seems to be a big room for improvement. This motivates us to try other models listed as follows.

4.2 Models Applied

We started from linear SVM, SVM model with rbf kernel and random forest. We used *GridSearch* in *sklearn* to help us find the best parameters in these models. The results are shown in Table (4) and Table (5).

Table 4: SVM Model Results

Model	Best train score	C	Gamma	Best test score	AUC
Linear	0.7885	1		0.6950	0.6322
RBF	0.79	1	0.003	0.7368	0.5725

Table 5: Random Forrest Result

AUC	Score	Criterion	max_features	n_est
0.6734	0.7504	entropy	sqrt	10
0.6704	0.7364	gini	None	10
0.6697	0.7611	gini	sqrt	20

4.3 Model Comparison

As shown in Table (4) and Table (5), the Random Forest has the best accuracy (0.7611). Also, it has better AUC around 0.67, whereas the SVM models and the baseline models are around 0.6. There are a lot of interesting things to comment on these models. We list some key observations as follows.

1. Linear SVM has the highest accuracy but doesn't perform so well in the test set. This seems to be a result of over-fitting.

2. *max_features* in Random Forest seems to be a critical parameter. As we have 10 features, "None" actually means 10 features, while "sqrt" means 3 features. In short, we should not consider the whole features at the same time for this classification problem. This observation matches the general empirical experience: Empirical good default values are $max\_features = n\_features$ for regression problems, and $max\_features = sqrt(n\_features)$ for classification tasks.

5 Deployment

The model developed as a result of this project can be incorporated into the Steam recommendation system. The model can enhance the accuracy of game recommendations for Steam users. Currently, Steam has a recommendation tab in their game library, and this model can replace their existing model if it outperforms its prediction accuracy. As an alternative, the model can also be introduced in parallel with the existing recommendation system with as a separate category. This can allow a direct comparison between the performance of the two models.

The criteria for assessment for this model will be the proportion of accepted recommendations. From a business standpoint, a new model is valuable if it generates recommendations that users trust more than the previous model. A straightforward metric for acceptance of a recommendation system is the number of recommendations that actually lead users to buy a recommended game. To evaluate the efficiency of the new model, we can compare the recommendation turnout (games bought after recommendation) of the previous model with the new one. As our model has more data points, it has the advantage of more information but is also slightly more susceptible to overfit. When deploying the new model, the organization should be aware of the limitations of recommendation models. Even after testing the performance of the model using metrics like accuracy and AUC, it is still difficult to interpret the value of a recommendation model. This is because taste in complex media like games and movies is a complicated entity and like otherartistic media, a user's preferences in games are susceptible to sudden changes and conceptual drifts. There are several ethical considerations that went into the design of our prediction model. Firstly, we did not include any demographic variables in our model to avoid training based on gender, race, age or other socio-economic features. This allowed us to train our model based on interest and activity, rather than socioeconomic factors. Secondly, we indexed our data using gamertags rather than real names, to protect the privacy of our users. Finally, we used a random stratified sampling approach for our data to train our model to all types of games as well as gamers. This step helped us limit discrimination against gaming minorities.

6 Conclusion

In this project, we tried to tackle a difficult data science problem of predicting user preferences in media and arts. We did this for Steam, a digital distribution platform for Video games. We parsed our data through multiple sources by scraping webpages from the Steam website and from the Steam database API. After collecting and processing our data, we iteratively trained and analyzed two binary classification models. We tweaked these models to optimize the results by comparing them to our baseline decision tree model. After several rounds of tweaking model parameters, we got the best results from two different models. We got our best accuracy score (0.7885) from a SVM model with RBF and received our best AUC score (0.6734) from a Random forest model with max_features = 'sqrt', which translates to 3 feature model in our case. From the analysis, it seems that the simplicity of the RF model (only 3 features) helped avoid overfitting and lead to a simple yet efficient model. In addition to the data science aspect of our project we also worked on business and ethical aspects of the project. We discussed the deployment of our model using a partial-parallel-introduction approach and discussed the organizational considerations and risks associated with the implementation of this model. Finally, we talked about the ethical considerations in our sampling, feature engineering and modeling to avoid bias and discrimination in our prediction model.

7 Appendix

7.1 Contribution

- **Hao-Yuan Chen (hyc404)**:
  1. Build a parser to scrape raw data from Steam website using *selenium* and *beautifulsoup* modules
  2. Tune SVM models using *GridSearch* in *sklearn*
- **I-Wu Lu (iwl210)**:

   1. Include NLP concept to the feature generation
   2. Build most of the dictionaries and the feature generation function
   3. Design the training models and the tuning range of parameters

- **Shashank K Pawar (skp363)**:
   1. Feature design
   2. Ethics Design
   3. Business Planning
   4. Pitch Design and Presentation
   5. Domain insights research
   6. Deployment design

7.2 Code

1. **Parse Game Reviews**

```python
import csv
import time
import re
import os
import sys
from bs4 import BeautifulSoup
from selenium import webdriver
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.common.keys import Keys
from selenium.common.exceptions import NoSuchElementException

#python3 parse_General.py genre
def parse(id, newFileName):
    driver = webdriver.PhantomJS("../../phantomjs-2.1.1-linux-x86_64/bin/phantomjs")
    #scrolling
```

```
reviewUrl = "http://steamcommunity.com/app/" + id + \
"/reviews/?browsefilter=toprated&snr=1_5_reviews_"
driver.get(reviewUrl)
time.sleep(1)
curParse = 0
count = 0

originSoup = BeautifulSoup(driver.page_source,'html.parser')

curParse = len(originSoup.findAll("div", { "class" : "date_posted" }))
click = False
end = False
startTime = time.clock()
while True:
    if driver.current_url != reviewUrl:
        driver.close()
        return 0
        break
    oldSource = driver.page_source
    driver.execute_script("window.scrollTo(0, document.body.scrollHeight);")
    #after scroll
    newSource = driver.page_source

    if len(newSource) != len(oldSource):
        startTime = time.clock()

    looptime = time.clock() - startTime
    if looptime > 15 and click == False:
        print("finding click")
        try:
            driver.find_element_by_link_text('See More Content').click()
            print("click!!")
            startTime = time.clock()
            continue
        except:
            click = True
            continue

    if looptime > 20 and click == True:
        print("finding end")
        try:
            driver.find_element_by_link_text('share a screenshot, \
            make a video, or start a new discussion!')
            break
        except:
            click = False
            startTime = time.clock()
            end = True
            continue
    if end == True:
        break

page_source = newSource
driver.close()

#parsing
print("parse "+id)
soup = BeautifulSoup(page_source,'html.parser')
user = soup.findAll("div", {"class" : "apphub_CardContentAuthorBlock tall"})
date = soup.findAll("div", { "class" : "date_posted" })
```

```
hour = soup.findAll("div", { "class" : "hours" })
helpful = soup.findAll("div", { "class" : "found_helpful" })
positive = soup.findAll("div", { "class" : "title" })
content = soup.findAll("div", { "class" : "apphub_CardTextContent" })

userData = []
userName = []
dateData = []
contentData =[]
positiveData =[]
hourData = []
helpfulData = []
notHelpfulData = []
funnyData = []
for each in user:
    userData.append(each.find('a')['href'])

for each in date:
    dateData.append(each.get_text())

for each in content:
    contentData.append(each.get_text())

for each in positive:
    if(each.get_text()=="Recommended"):
        positiveData.append(1)
    else:
        positiveData.append(0)

for each in hour:
        hourData.append(each.get_text())

    for each in helpful:
        helpfulData.append(each.get_text())

    print("write: "+id)
    with open(newFileName, 'w', encoding='utf-8') as f:
            writer = csv.writer(f)
            writer.writerow(["user name","positive or negative",\
        "total playing time","number of helpful","content"])
            leng = len(userData)
            for i in range(leng):
            try:
                writer.writerow([userData[i],positiveData[i],\
                hourData[i],helpfulData[i],contentData[i]])
            except:
                print("out of range")
    f.close()
    print("end: "+id)

tag = sys.argv[1]
count = 0
appIdFile = "../appId/"+"appId_"+tag+".csv"

for id in open(appIdFile):
    for char in id:
        if char in "\n":
            id = id.replace(char,'')
    fileName = tag + "/"+id+".csv"
    newFileName =tag+"/"+"parseNew_"+id+".csv"
    if os.path.isfile(newFileName):
        print(newFileName+' exist')
        count = count+1
        continue
    else:
        with open(newFileName, 'w') as fout:
            print('{0} start'.format(newFileName))
            print(count)
        fout.close()
        parse(id, newFileName)
```

2. **Parse User Information**

```
import csv
import time
import os
import re
import sys
from bs4 import BeautifulSoup
from selenium import webdriver
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.common.keys import Keys
from selenium.common.exceptions import NoSuchElementException
import pandas as pd
import steamapi

#python3 parseUserGeneral genre
def parseUser(fileName, newFileName, csvtag, csvid):
    df = pd.read_csv(fileName)
    username = list()
```

```
recommend = list()
time = list()
helpful = list()
content = list()
for each in df[df.columns[0]]:
    if each != '':
        username.append(each)
for each in df[df.columns[1]]:
    if each != '':
        recommend.append(each)
for each in df[df.columns[2]]:
    if each != '':
        time.append(each)
for each in df[df.columns[3]]:
    if each != '':
        helpful.append(each)
for each in df[df.columns[4]]:
    if each != '':
        content.append(each)

print('UserNum: {0}'.format(len(username)))
genereList = []
userLevel = []
for each in username:
    print(each)
    if each == '':
        continue

    user = each
    generes = {'action':0 ,'adventure':0 ,'racing':0 ,\
    'rpg':0 ,'simulation':0 ,'sports':0 ,'strategy':0}
    appidList = []
    #Regular Expression
    idReg = re.compile('http://steamcommunity.com/id/(.*)/')
    profileReg = re.compile('http://steamcommunity.com/profiles/(.*)/')

    idMatch = re.match(idReg, user)
    profileMatch = re.match(profileReg, user)
    if idMatch:
        name = idMatch.group(1)
        try:
            me = steamapi.user.SteamUser(userurl=name)
            userLevel.append(me.level)
            for each in me.games:
                appidList.append(each.appid)
        except:
            genereList.append(generes)
            print('exception')
            continue
    elif profileMatch:
        name = profileMatch.group(1)
        try:
            me = steamapi.user.SteamUser(name)
            userLevel.append(me.level)
            for each in me.games:
                appidList.append(each.appid)
        except:
            genereList.append(generes)
            print('exception')
            continue
```

```python
    else:
        print('not found user')
        genereList.append(generes)
        continue

    for id in appidList:
        genereTag = ['action','adventure','racing',\
        'rpg','simulation','sports','strategy']
        found = 0
        for tag in genereTag:
            if found == 0:
                fileNameApp = "../../appId/appId_"+ tag  + ".csv"
                with open(fileNameApp, 'rt') as f:
                    reader = csv.reader(f, delimiter=',')
                    for row in reader:
                        if str(id) == row[0]:
                            generes[tag] = generes[tag] + 1
                            found = 1
                            break

    genereList.append(generes)
    print(generes)
print(len(genereList))
outFile = newFileName
with open(outFile, 'w') as fout:
    writer = csv.writer(fout)
    writer.writerow(["user name","user level",\
    "positive or negative","total playing time",\
    "number of helpful","content","Action","Adventure",\
    "Racing","RPG","Simulation","Sports","Strategy"])
```

```
        for i in range(len(username)):
            try:
                genere = genereList[i]
                writer.writerow([username[i],userLevel[i],\
                recommend[i],time[i],helpful[i],content[i],\
                genere['action'],genere['adventure'],genere['racing'],\
                genere['rpg'],genere['simulation'],genere['sports'],genere['strategy']])
            except:
                print('out of index')
    print('{0} succeed'.format(outFile))
    fout.close()

steamapi.core.APIConnection(api_key="56C90FBD8CC1C30B69E55D0194282197")

tag = sys.argv[1]
count = 0
appIdFile = "../../appId/"+"appId_"+tag+".csv"
for id in open(appIdFile):
    for char in id:
        if char in "\n":
            id = id.replace(char,'')
    fileName = "../"+tag+"/parseNew_"+id+".csv"
    newFileName = "../"+tag+"/finish/"+id+".csv"

    if os.path.isfile(fileName):
        if os.path.isfile(newFileName):
            print(newFileName+' exist')
            count = count+1
            continue
        else:
            with open(newFileName, 'w') as fout:
                print('{0} start'.format(newFileName))
            fout.close()

        print(id+" start!")
        userList = []
        df = pd.read_csv(fileName)

        for each in df[df.columns[0]]:
            userList.append(each)
        print(count)
        parseUser(fileName,newFileName, tag, id)
    else:
        print(id+" not exist")
```

3. **Feature generation functions**

```
from scipy.stats import zscore
import pip

from textblob import TextBlob

def get_review_helpful(g_id, u_id):
    list_of_review = []
    for r_id in review:
        if review[r_id]['game_id'] == g_id:
            list_of_review.append(r_id)
    sl = []
    target = []
    for i in range(len(list_of_review)):
        r_id = list_of_review[i]
        score = review[r_id]['helpful']
        sl.append(score)
        if u_id == review[r_id]['user_id']:
            target.append(i)
    #print('total review = ', len(list_of_review))
    #print('target len = ', len(target))
    if len(target) == 0:
        return 0
    if np.std(sl) == 0:
        return 0
    else:
        z = zscore(sl)
        val = [z[x] for x in target]
        return np.mean(val)

def get_review_normalized(g_id, u_id):
    list_of_review = u_review[u_id]
    p = []
    s = []
    target = []
    if len(list_of_review) <= 1:
        return (0,0)
    for i in range(len(list_of_review)):
        r_id = list_of_review[i]
        tup = get_review_pos_score(r_id)
        p.append(tup[0])
        s.append(tup[1])
        if g_id == review[r_id]['game_id']:
            target.append(i)
    #print('total review = ', len(list_of_review))
    #print('target len = ', len(target))
    if len(target) == 0:
        return (0,0)
    if np.std(p) == 0:
        a = 0
    else:
        #print(p)
        zp = zscore(p)
        val_p = [zp[x] for x in target]
        a = np.mean(val_p)

    if np.std(s) == 0:
        b = 0
    else:
        #print(s)
        zs = zscore(s)
        val_s = [zs[x] for x in target]
        b = np.mean(val_s)
    return (a, b)

def get_review_pos_score(r_id):
    pattern = TextBlob(review[r_id]['content'])
    pol = pattern.sentiment[0]
    sub = pattern.sentiment[1]
    return (pol, sub)

def get_num_games(g_id, u_id):
    target = ""
    for g in classification:
        if g_id in classification[g]:
```

```python
            target = g
            break
    return user[u_id][target]

def get_avg_user_level(g_id):
    level_total = 0
    cnt = 0
    for u in game_user[g_id]:
        if u in user:
            level_total = level_total + user[u]['userLevel']
            cnt = cnt + 1
    return level_total/cnt

def get_avg_user_playtime(g_id):
    playtime_total = 0
    cnt = 0
    for u in game_user[g_id]:
        playtime_total = playtime_total + game_user[g_id][u]['total_play_time']
        cnt = cnt + 1
    return playtime_total/cnt

def get_avg_user_playtime_in_G(g_id):
    playtime_total = 0
    num_user = 0
    target = ""
    for g in classification:
        if g_id in classification[g]:
            target = g
            break
    for all_other_g in classification[target]:
        if all_other_g not in result:
            continue
        for u in game_user[all_other_g]:
            playtime_total = playtime_total + \
            game_user[all_other_g][u]['total_play_time']
            num_user = num_user + len(game_user[all_other_g])
    return playtime_total/num_user

def get_user_playtime(g_id, u_id):
    return game_user[g_id][u_id]['total_play_time']

def get_user_level(u_id):
    return user[u_id]['userLevel']

#remember to load pickle file: review.p
def get_total_review_g(g_id):
    count = 0
    for each in review:
        if review[each]['game_id'] == g_id:
            count = count + 1
    #print('{0} : {1} '.format(g_id,count))
    return count



def get_total_review_u(u_id):
    count = 0
    for each in review:
        if review[each]['user_id'] == u_id:
            count = count + 1
    #print('{0} : {1} '.format(u_id,count))
    return count
```

4. **Baseline Model**

```
from sklearn import tree

from sklearn import metrics

baseline = tree.DecisionTreeClassifier(class_weight=None, \

 criterion='gini', max_depth=None, \

 max_features=None, max_leaf_nodes=None, min_samples_leaf=1, \

 min_samples_split=2, min_weight_fraction_leaf=0.0, \

 presort=False, random_state=None, splitter='best')

baseline.fit(X_train, Y_train.ravel())

expected = Y_test.ravel()

predicted = baseline.predict(X_test)

# summarize the fit of the model

print(metrics.classification_report(expected, predicted))

print(metrics.confusion_matrix(expected, predicted))
```

5. **GridSearch in SVM**

```
from sklearn.svm import SVC

from sklearn.model_selection import GridSearchCV

from sklearn.metrics import fbeta_score, make_scorer,\

average_precision_score, classification_report


parameter_candidates = [{'kernel': ['rbf'], 'gamma': [1e-3, 1e-4],

                     'C': [1, 10, 100, 1000]},

                    {'kernel': ['linear'], 'C': [1, 10, 100, 1000]}]


# Create a classifier object with the classifier and parameter candidates

print('start grid')

clf = GridSearchCV(estimator=SVC(), param_grid=parameter_candidates, n_jobs=-1)

clf.fit(X_train, np.ravel(Y_train))
```

```
print('finish')


print('Best score for data1:', clf.best_score_)

print('Best C:',clf.best_estimator_.C)

print('Best Kernel:',clf.best_estimator_.kernel)

print('Best Gamma:',clf.best_estimator_.gamma)
```