team-1_report

January 10, 2022

1 DS3000 Final Project: Board Game Reccomendation

1.1 Team -1 (example)

- Piotr Sapiezynski (p.sapiezynski@northeastern.edu)
- Matt Higger (m.higger@ccs.neu.edu)

2 Executive Summary

We build a board game reccomendation system by collecting a data from boardgamegeek.com's list of boardgames and collecting 13 users preferences about which games they do and don't enjoy. Our reccomender works by identifying the unrated game most similar to the user's top rated games. To validate our method, we estimate how closely the predicted user preferences match the observed user preferences under cross validation. The predicted user ratings do a poor job of matching actual user preferences. We suggest that the model struggles because it fails to find a meaningful way of measuring whether two games are similar or not.

3 Ethical Considerations

Like any tool which reccomends products on might buy, this tool may be subject to bias from board game companies who wish to drive consumers to their products. We suggest that any product derived from this work be open-source to allow for people to easily audit its use for commercial bias.

4 Introduction

Finding the right board game to play is difficult. The time and money required to play test a game is considerable and media which describes gameplay can fail to capture the experience accurately. As a result, many players learn about new games by word of mouth. This situation leaves many great games "undiscovered" and hinders player enjoyment of gaming by only playing popular games. **This project aims to reccomend new board games to a player who submits their preferences on other games**.

5 Data Description

5.1 Games

(Full details of game data can be found in `ex_game_clean.ipynb`, a summary of the relevant details is given here).

We scrape a list of boardgames ranked by popularity from BoardGameGeek.

[1]:
```
import pandas as pd

df_game = pd.read_csv('game_final.csv', index_col='game_id')
df_game.loc[:, ['description', 'title']].head()
```

[1]:
```
                                               description  \
game_id
174430   Vanquish monsters with strategic cardplay. Ful…
161936   Mutating diseases are spreading around the wor…
224517   Build networks, grow industries, and navigate …
167791   Compete with rival CEOs to make Mars habitable…
233078   Build an intergalactic empire through trade, r…

                                     title
game_id
174430                          Gloomhaven
161936           Pandemic Legacy: Season 1
224517                  Brass: Birmingham
167791                  Terraforming Mars
233078   Twilight Imperium: Fourth Edition
```

In particular, we collect the category tags associated with each individual game:

[2]:
```
def is_feat(col, feat_prefix=('cat: ',)):
    for prefix in feat_prefix:
        if col.startswith(prefix):
            return True
    return False

def strip_feat(col, feat_prefix=('cat: ',)):
    for prefix in feat_prefix:
        if col.startswith(prefix):
            return col[len(prefix):]
    raise Error('input column is not a feature')


# build x feature list (any category a game belongs to)
x_feat_list = list()
for col in df_game.columns:
    if is_feat(col):
```

```
        x_feat_list.append(col)

x_feat_list[:5]
```

[2]:
```
['cat: Adventure',
 'cat: Exploration',
 'cat: Fantasy',
 'cat: Fighting',
 'cat: Miniatures']
```

[3]:
```
df_game.loc[:, x_feat_list[:5]].head()
```

[3]:
```
         cat: Adventure  cat: Exploration  cat: Fantasy  cat: Fighting  \
game_id
174430             True              True          True           True
161936            False             False         False          False
224517            False             False         False          False
167791            False             False         False          False
233078            False              True         False          False

         cat: Miniatures
game_id
174430              True
161936             False
224517             False
167791             False
233078             False
```

5.2 User Preferences

User preferences were collected by soliciting student responses via a google form. Students of the spring 2020 DS3000 class were solicited:

Each user is represented by an integer `alias`. Each column represents a game and the values are the responses to the question above. Missing values indicate that a user did not give their preference on a particular game.

[4]:
```
df_pref = pd.read_csv('pref_final.csv', index_col='alias')
df_pref
```

[4]:
```
       174430  2398  171  178900  188834  105134  2453  12962  2181  278  …  \
alias                                                                     …
1         1.0   7.0  6.0     5.0     4.0     4.0   5.0    1.0   5.0  2.0  …
6         6.0   NaN  NaN     4.0     5.0     NaN   NaN    NaN   NaN  NaN  …
7         NaN   NaN  NaN     4.0     NaN     NaN   NaN    NaN   NaN  NaN  …
9         NaN   NaN  3.0     7.0     NaN     NaN   5.0    NaN   NaN  NaN  …
15        NaN   NaN  7.0     NaN     6.0     NaN   NaN    NaN   NaN  NaN  …
17        NaN   NaN  NaN     6.0     NaN     NaN   NaN    NaN   NaN  NaN  …
```

```
18          NaN   NaN  NaN      NaN      NaN      NaN   NaN     NaN   NaN  NaN  …
19          NaN   NaN  4.0      NaN      NaN      NaN   6.0     NaN   4.0  NaN  …
20          NaN   5.0  5.0      7.0      6.0      6.0   NaN     NaN   NaN  NaN  …
21          NaN   NaN  7.0      4.0      5.0      NaN   NaN     NaN   NaN  NaN  …
22          NaN   3.0  3.0      6.0      NaN      NaN   2.0     NaN   NaN  NaN  …
23          NaN   NaN  7.0      7.0      7.0      NaN   4.0     NaN   NaN  NaN  …
24          NaN   NaN  NaN      NaN      NaN      NaN   NaN     NaN   NaN  NaN  …

       195162  92415  275467  128882  204305  169786  120677  31627  174785  \
alias
1         NaN    NaN     NaN     NaN     NaN     NaN     NaN    NaN     NaN
6         NaN    NaN     NaN     NaN     NaN     NaN     NaN    NaN     NaN
7         NaN    NaN     NaN     NaN     NaN     NaN     NaN    NaN     NaN
9         NaN    NaN     NaN     NaN     NaN     NaN     NaN    NaN     NaN
15        NaN    NaN     NaN     NaN     NaN     NaN     NaN    NaN     NaN
17        NaN    NaN     NaN     NaN     NaN     NaN     NaN    NaN     NaN
18        NaN    NaN     NaN     NaN     NaN     NaN     NaN    NaN     NaN
19        NaN    NaN     NaN     NaN     NaN     NaN     NaN    NaN     NaN
20        4.0    NaN     NaN     NaN     NaN     NaN     NaN    NaN     NaN
21        NaN    NaN     NaN     NaN     NaN     NaN     NaN    NaN     NaN
22        NaN    5.0     NaN     NaN     NaN     NaN     NaN    NaN     NaN
23        NaN    NaN     2.0     NaN     NaN     NaN     NaN    NaN     NaN
24        NaN    NaN     NaN     NaN     NaN     3.0     4.0    6.0     5.0

       253284
alias
1         NaN
6         NaN
7         NaN
9         NaN
15        NaN
17        NaN
18        NaN
19        NaN
20        NaN
21        NaN
22        NaN
23        NaN
24        6.0

[13 rows x 78 columns]
```

Users were asked to rank at least 9 games, though we include all users with at least 8 to include a few more users:

[5]:
```
# games ranked per user
(df_pref >= 0).sum(axis=1)
```

```
[5]: alias
1     11
6     15
7     21
9      8
15     9
17     9
18    10
19     9
20    10
21    10
22    12
23     8
24     8
dtype: int64
```

6 Method

6.1 1 - NN Regressor

To reccomend games to users we use a 1-Nearest Neighbor Regressor. In essense, every game is given an estimated user preference as the preference score of the "most similar" game among all the games the user has rated.

This approach requires that we are able to identify the "most similar" game to any other. To do so we build a distance metric which measures game similarity. The distance between similar games should be small while the distance between different games should be large. We choose the metric as the traditional squared distance:

$$d_{i,j} = ||y_1 - y_0||_2^2 = \sum_i (y_{1,i} - y_{0,i})^2$$

where vectors x_i represent a board games tags:

```
[6]: # for example, for our first two games:
y = df_game.loc[:, x_feat_list].values.astype(int)
y0 = y[0, :]
y1 = y[1, :]
y0
```

```
[6]: array([1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0])
```

The vector x0 above indicates that the first game has the first 5 tags (i.e. Adventure, Exploration, Fantasy, Fighting, Miniatures) but none of the others.

[7]:
```python
import numpy as np

# compute distance between first and second games
d01 = np.linalg.norm(y1 - y0) ** 2
d01
```

[7]:
```
7.000000000000001
```

Notice that the distance is equivilent to a count of how many category tags in `x_feat_list` which are different between two games.

6.2 Principle Component Analysis

There are two problems with the distance metric above. 1. **The scale of each feature is different:**

[8]:
```python
df_game.loc[:, x_feat_list].var().sort_values()
```

[8]:
```
cat: Vietnam War                    0.001008
cat: Expansion for Base-game        0.001008
cat: Trivia                         0.002014
cat: American Revolutionary War     0.002014
cat: World War I                    0.002014
                                      …
cat: Science Fiction                0.110995
cat: Fighting                       0.131274
cat: Economic                       0.159312
cat: Fantasy                        0.164704
cat: Card Game                      0.183588
Length: 79, dtype: float64
```

Left uncorrected, all the difference among `Card Game` would dominate the differences scores and ignore features with lower variances (i.e. `Vietnam War`, `Expansion for Base-game`).

2. **Even if each features were given identical variance, some features effectively "double count" the importance of a feature by being correlated**

[9]:
```python
df_game.loc[:, x_feat_list[:5]].corr()
```

[9]:
```
                  cat: Adventure  cat: Exploration  cat: Fantasy  \
cat: Adventure          1.000000          0.426874      0.353859
cat: Exploration        0.426874          1.000000      0.177884
cat: Fantasy            0.353859          0.177884      1.000000
cat: Fighting           0.320280          0.154813      0.391303
cat: Miniatures         0.293766          0.155875      0.242340

                  cat: Fighting  cat: Miniatures
cat: Adventure         0.320280         0.293766
cat: Exploration       0.154813         0.155875
```

```
cat: Fantasy            0.391303          0.242340
cat: Fighting           1.000000          0.434097
cat: Miniatures         0.434097          1.000000
```

Notice that these first 5 tags are all positively correlated with each other (when one tag occurs any of the others is more likely to occur). In some sense, we can consider that each of these tags are redundant measurements of the same intrinsic game feature. We are effectively over-counting this feature by including it with each feature.

To resolve both of these issues, we use a pre-processing step before applying our 1-NN regressor: Principle Component Analysis (PCA). PCA will: - ensure output features each have equal variance - ensure output features are all uncorrelated with each other

7 Results

7.1 Estimation

[10]:
```
import pandas as pd

# reload data
df_game = pd.read_csv('game_final.csv', index_col='game_id')
df_pref = pd.read_csv('pref_final.csv', index_col='alias')

# ensure column names are integers
df_pref.rename(int, axis=1, inplace=True)

# build x feature list (any category a game belongs to)
x_feat_list = list()
for col in df_game.columns:
    if  is_feat(col):
        x_feat_list.append(col)

x_feat_list[:5]
```

[10]:
```
['cat: Adventure',
 'cat: Exploration',
 'cat: Fantasy',
 'cat: Fighting',
 'cat: Miniatures']
```

[11]:
```
from sklearn.model_selection import KFold
from sklearn.metrics import r2_score

def get_x_y(alias, df_pref, df_game, x_feat_list):
    """ gets the input / output features of regressor for one user

    The input features are the game categories (binary) and
    the output features are the user preferences
```

```
    Args:
        alias (int): alias given to a user (index of df_pref)
        df_pref (pd.DatFrame): user preferences
        df_game (pd.DataFrame): game stats

    Returns:
        x (np.array): (n_samples, n_feat) corresponds to the
            categories every game does / doesn't belong to
        y (np.array): (n_samples) user preferences of corresponding
            samples
        game_id_list (list): game ids with ratings
    """

    # get non null preferences for a given alias
    s_pref_alias = df_pref.loc[alias, :]
    s_pref_alias.dropna(inplace=True)

    # get list of game_id which user submitted preferences about
    game_id_list = list(s_pref_alias.index)

    # extract x, y
    x = df_game.loc[game_id_list, x_feat_list].values
    y = s_pref_alias.values

    return x, y, game_id_list
```

[12]:
```
import numpy as np
from sklearn.linear_model import LinearRegression
from sklearn.neighbors import KNeighborsRegressor

def cv_train(x, y_true):
    """ leave one out cross validation regression of x, y

    Args:
        x (np.array): (n_samples, n_feat) input features
        y_true (np.array): (n_samples) output feature

    Returns:
        regressor (LinearRegression): model which predicts y
            from x
        r2 (float): percentage of variance of y which is
            explained by the model under cross validation
            (r2=1 is strongest possible model, r2 = 0 is
            a non-helpful model)
    """
    # initialize kfold
```

```
    n_samples = x.shape[0]
    kfold = KFold(n_splits=n_samples)

    # initialize regressor
    reg = KNeighborsRegressor(n_neighbors=1)

    y_pred = np.empty_like(y_true)
    for train_idx, test_idx in kfold.split(x):
        # split data
        x_train = x[train_idx, :]
        y_train = y_true[train_idx]
        x_test = x[test_idx, :]

        # fit regressor
        reg.fit(x_train, y_train)

        # predict
        y_pred[test_idx] = reg.predict(x_test)

    # compute r2
    r2 = r2_score(y_true=y_true, y_pred=y_pred)

    # fit model on entire dataset (best for predicting new samples)
    reg.fit(x, y_true)

    return reg, r2
```

[13]:
```
def predict_score(alias, df_pref, df_game, x_feat_list):
    """ predicts scores on all games

    Args:
        alias (int): integer alias of user
        df_pref (pd.DataFrame): user preferences
        df_game (pd.DataFrame): games stats
        x_feat_list (list): features used to define distance
            between games

    Returns:
        df_predicted_pref (pd.DataFrame): estimated user preferences
            (includes preferences for all games, not just the ones
            the user has rated)
        reg (KNeighborsRegressor): regressor which predicts user preferences
        r2 (float): cross validated r2 value
    """

    # extract relevant data
```

```
    x, y, game_id_list = get_x_y(alias, df_pref, df_game, x_feat_list)

    # cross validate & train model
    reg, r2 = cv_train(x, y)

    # predict scores of all games (not just ones with observed preferrences)
    x_all = df_game.loc[:, x_feat_list].values
    y_predict = reg.predict(x_all)

    # collect / sort preferences in dataframe
    df_predicted_pref = pd.DataFrame({'title': df_game['title'],
                                      'pref': y_predict,
                                      'url': df_game['url']},
                                     index=df_game.index)

    # record whether preferences were observed (user supplied) or not
    df_predicted_pref.loc[:, 'observed'] = False
    df_predicted_pref.loc[game_id_list, 'observed'] = True

    # store x_feat in df_predcticted_pref (redundant but helpful to know
    # which were used across multiple runs)
    for x_feat_idx, x_feat in enumerate(x_feat_list):
        df_predicted_pref.loc[:, x_feat] = x_all[:, x_feat_idx]


    # sort by estimated rating
    df_predicted_pref.sort_values('pref', inplace=True, ascending=False)

    return df_predicted_pref, reg, r2
```

7.2 Validation

To validate our model, we compute the cross-validated r^2 value among all the games a user has given ratings for.
- If this value is close to 1, then we can effectively predict user preferences - If this value is close to zero, then we are effectively guessing user preferences blindly - If this value is negative, then we are doing worse than guessing user preferences blindly

7.2.1 Without applying PCA:

[14]:
```
def validate_all(df_pref, df_game, x_feat_list):
    """ computes cross validated r2 for each alias

    Args:
        df_pref (pd.DataFrame): user preferences
        df_game (pd.DataFrame): games stats
        x_feat_list (list): features used to define distance
```

```
            between games

    Returns:
        df_validate (pd.DataFrame): index is alias, contains
            column `cv_r2` as well as `num_pref`, the number of
            preferences available for a given user
    """
    df_validate = pd.DataFrame()
    for alias in df_pref.index:
        # predict scores
        df_predicted_pref, reg, r2 = predict_score(alias, df_pref, df_game,␣
↪x_feat_list)

        # collect validation stats in one dataframe
        row = dict(alias=alias, cv_r2=r2,␣
↪num_pref=df_predicted_pref['observed'].sum())
        df_validate = df_validate.append(row, ignore_index=True)

    # prep and display df_validate
    df_validate.set_index('alias', inplace=True)
    df_validate.sort_values('cv_r2', inplace=True)

    return df_validate
```

[15]:

```
# validate model (without pca)
df_validate = validate_all(df_pref, df_game, x_feat_list)
df_validate
```

[15]:

alias	cv_r2	num_pref
9.0	-3.296296	8.0
23.0	-2.692308	8.0
1.0	-1.360236	11.0
18.0	-1.128514	10.0
21.0	-0.875000	10.0
17.0	-0.660428	9.0
7.0	-0.625000	21.0
20.0	-0.562500	10.0
15.0	-0.528302	9.0
22.0	-0.336709	12.0
19.0	-0.170000	9.0
6.0	-0.097561	15.0
24.0	0.125000	8.0

Only user `alias=24` achieved any improvement in preference estimation from our method.

7.2.2 Applying PCA:

[16]:
```
from sklearn.decomposition import PCA

# extract old x values
x = df_game.loc[:, x_feat_list].values

# transform to new x values
n_components = 2
pca = PCA(n_components=n_components, whiten=True)
x_new = pca.fit_transform(x)

# add pca features back into dataframe
x_feat_list_new = [f'pca{idx}' for idx in range(n_components)]
for idx, feat in enumerate(x_feat_list_new):
    df_game.loc[:, feat] = x_new[:, idx]
```

[17]:
```
# validate using only first n_pca features
df_validate_pca = validate_all(df_pref, df_game, x_feat_list_new)
```

[18]:
```
df_validate_pca
```

[18]:

alias	cv_r2	num_pref
17.0	-2.633690	9.0
9.0	-2.481481	8.0
1.0	-1.988189	11.0
23.0	-1.961538	8.0
21.0	-1.569444	10.0
18.0	-1.369478	10.0
15.0	-0.910377	9.0
7.0	-0.825000	21.0
20.0	-0.687500	10.0
22.0	-0.518987	12.0
6.0	-0.219512	15.0
19.0	0.010000	9.0
24.0	0.125000	8.0

PCA does improve results, though we are still not able to predict user preferences better than chance on the average user.

7.2.3 Visualization

[19]:
```
def get_text(game_row):
    # gets a string, in plotly format, of all tags a game contains
    title = game_row['title']
    tags = '<br>'.join([strip_feat(col) for col, val in game_row.items() if val␣
 ↪and is_feat(col)])
```

```
    return '<br>'.join([f'title: {title}',
                        f'{tags}'])
df_game['hovertext'] = df_game.apply(get_text, axis=1)
```

[20]:
```
import plotly.graph_objects as go
import plotly.express as px
from plotly.subplots import make_subplots

hovertemplate = '%{text}'

def print_plotly_scatter(alias, df_pref, df_game, x_feat_list, f_html=None,␣
↪x_feat_idx_horz=0, x_feat_idx_vert=1):

    if f_html is None:
        f_html = f'user{alias}.html'

    # compute predicted scores
    df_predicted_pref, reg, r2 = predict_score(alias, df_pref, df_game,␣
↪x_feat_list_new)

    x_feat0 = x_feat_list[x_feat_idx_horz]
    x_feat1 = x_feat_list[x_feat_idx_vert]

    # build scatter
    fig = make_subplots()
    for observed in [False, True]:
        #  select only relevant rows
        row_bool = df_predicted_pref['observed'] == observed
        df = df_predicted_pref.loc[row_bool, :]

        s_text = df_game.loc[df.index, 'hovertext']

        if observed:
            marker_dict = dict(size=12, line=dict(width=2, color='black'),␣
↪colorscale='viridis')
            name = 'user-given'
        else:
            marker_dict = dict(colorscale='viridis',␣
↪colorbar=dict(thickness=20, title='preference'))
            name = 'estimated'

        trace = go.Scatter(x=df[x_feat0],
                           y=df[x_feat1],
                           mode='markers',
                           marker=marker_dict,
                           marker_color=df['pref'],
                           hovertemplate=hovertemplate,
```

```
                            text=s_text,
                            name=name)

        fig.add_trace(trace)

    legend_dict = legend=dict(yanchor="top", y=0.99, xanchor="left", x=0.01)
    fig.update_layout(title=f'user {alias} preferences',
                      xaxis_title=x_feat0,
                      yaxis_title=x_feat1,
                      legend=legend_dict)

    fig.write_html(f_html)

    return f_html
```

[41]:
```
from IPython.display import IFrame

f_html = print_plotly_scatter(alias=7, df_pref=df_pref, df_game=df_game,␣
 ↪x_feat_list=x_feat_list_new)

# allows us to embed html in jupyter (helpful if error in creation of plot)
IFrame(src=f_html, width=900, height=600)
```

[41]:
```
<IPython.lib.display.IFrame at 0x7efbfdbe09a0>
```

8 Discussion

The project did not succeed in being able to predict a user's preference any better than chance for the average user. (Cross validated $r^2 < 0$ for all users). This can be due to three reasons: 1. Our user preference data was insufficient: - With only 8 to 20 games per user, we may not have enough data to accuractely characterize a single user's preferences among all the unique board games - The user rating scale is somewhat subjective and was often biased towards games users enjoyed. This makes intuitive sense as the majority of time one is interacting with a game they're interacting with a game they've selected because they enjoy it. - **We'd suggest future work collect only a list of games that a user enjoys** 1. Our distance metric, which defines which games are similar or different, was insufficient: - After much experimenting, we couldn't identify an `x_feat_list` which significantly improved the cross validated r^2 metric. - **We'd suggest future work do more feature engineering to identify which aspects of a game make is "similar" or "different".**

```
Alternatively, one could define a metric of game similarity based on the correlation of user ra
    - users typically rate both games high / low
        - games are similar
    - users typically rate one game high and the other low:
        - games are different
```

1. (Most significantly) Our 1-NN classifier was insufficient because:

- it gave identical scores to many games. This is not helpful in identifying a single best game to reccomend to a user
- it never synthesizes all the user preferences into its estimate. Instead, it relies exclusively on only the nearest neighbor.
- **We'd suggest future work discard the 1-NN classifier in favor of something which synthesizes all of a user's preferences (Regression, Density Estimation)**

Not all results were negative, while the distance between games was not sufficient to reccomend games, it did provide some intuitive meaning: - games in the lower left corner above are typically economic / negotiation games - games in the upper right corner above are typically strategy / fighting / minature games

8.1 Takeaway:

Taken together, we do not think this work should be used to reccomend board games.