Basic Visualizations

February 23, 2018

1 4 Basic Visualizations

First, you will create some basic visualizations of the MovieLens dataset described above. Using a method (e.g. histograms) of your choice, visualize the following: 1. All ratings in the MovieLens Dataset. 2. All ratings of the ten most popular movies (movies which have received the most ratings). 3. All ratings of the ten best movies (movies with the highest average ratings). 4. All ratings of movies from three genres of your choice (create three separate visualizations).

The Python packages Matplotlib and Seaborn are good choices for these visualizations, but there are also many other good visualization packages.

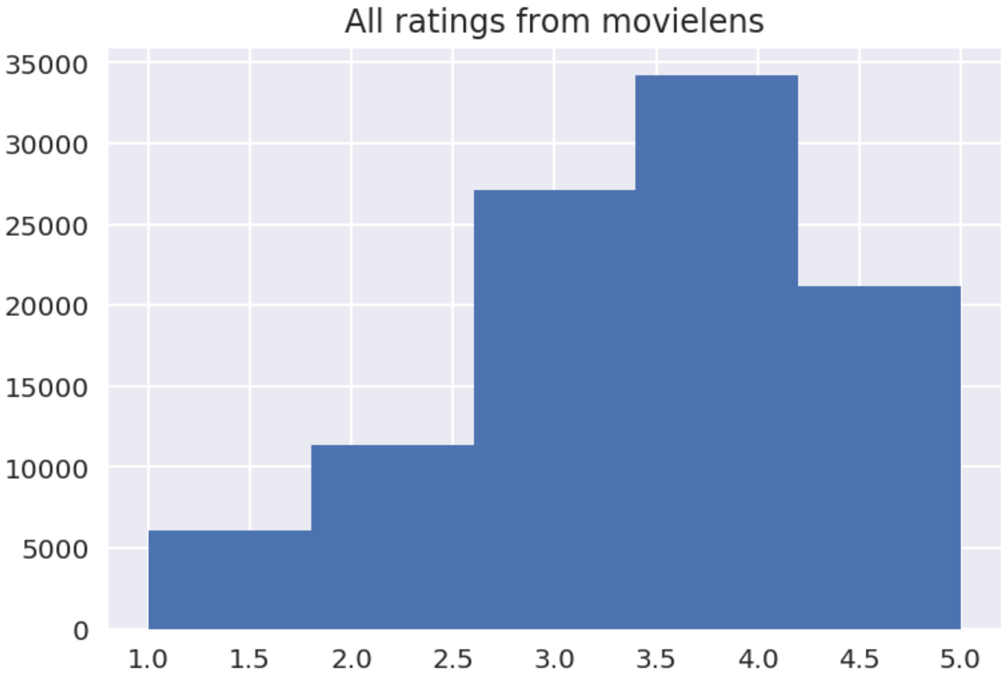
1.1 Report Deliverable

Your report should contain a section dedicated to basic visualizations. What, in general, did you observe? Did the results match what you would expect to see? How do the ratings from the most popular movies compare to the ratings of the best movies? How do the ratings of the three genres you chose compare to one another?

```
In [3]: import pandas as pd
        import seaborn as sns
        import matplotlib.pyplot as plt
        %config InlineBackend.figure_format='retina'

        sns.set()
        ratings = pd.read_table('data/data.txt', header = None, names = ['user', 'movie', 'rat
```

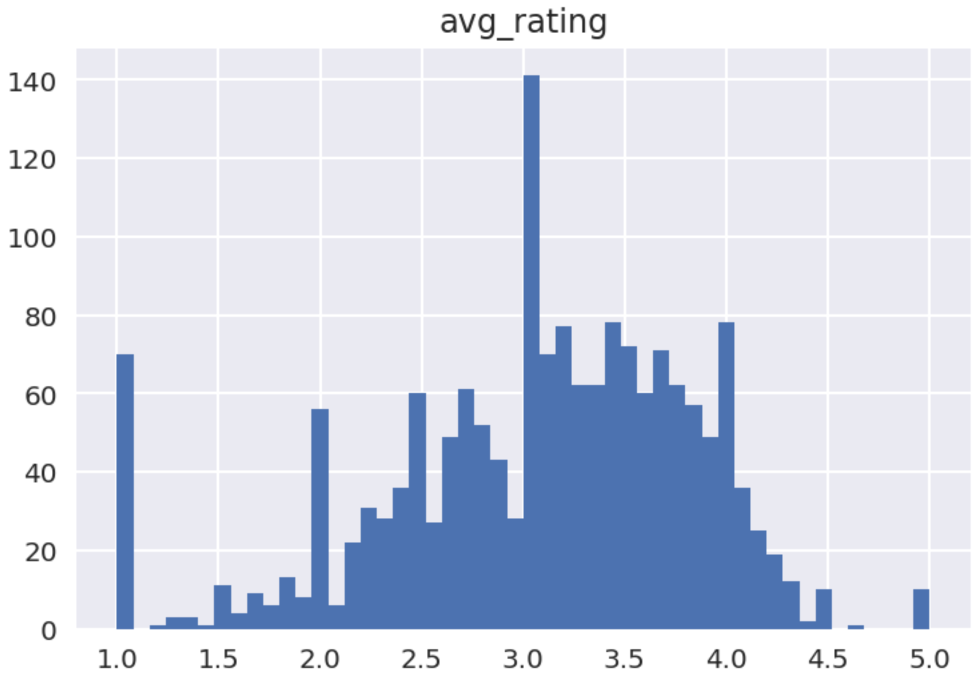
1.2 All ratings from MovieLens

```
In [29]: ratings.hist('rating', bins = 5)
         plt.title('All ratings from movielens')

Out[29]: Text(0.5,1,'All ratings from movielens')
```



1.3 Computing average ratings of all movies

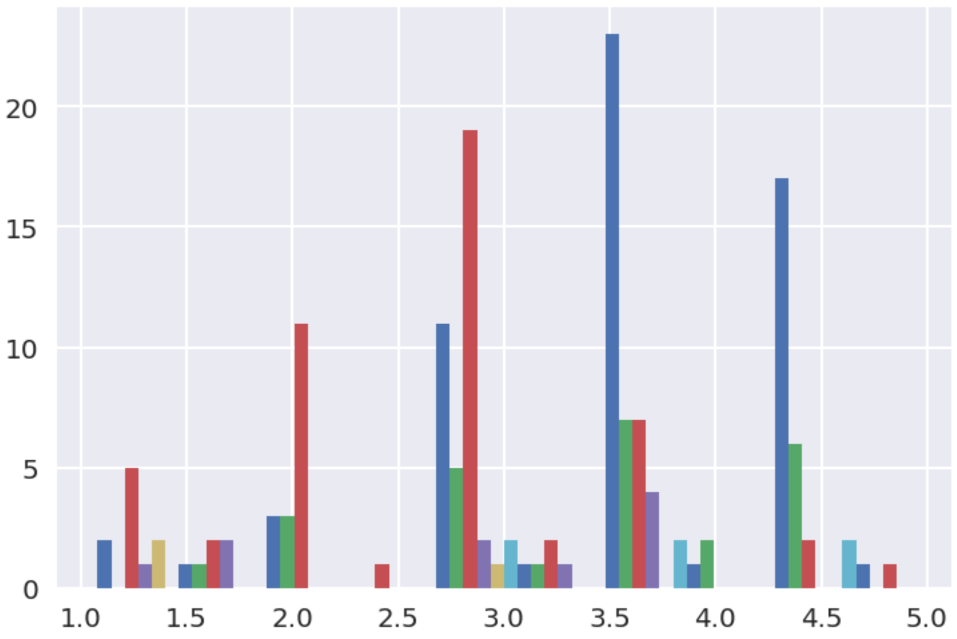
```
In [152]: avg_ratings = pd.DataFrame(columns =['movie','avg_rating'])

          for movie in range(1,1683):
              avg_ratings = avg_ratings.append({'movie':movie, 'avg_rating':ratings.loc[rating

          #avg_ratings.head()
          avg_ratings.hist('avg_rating', bins = 50)

Out[152]: array([[<matplotlib.axes._subplots.AxesSubplot object at 0x7f1d3cbaa438>]],
                dtype=object)
```



1.4 Comparing the top 10 best rated movies

```
In [153]: top10 = avg_ratings['avg_rating'].nlargest(n = 10)

          ratinglist=[]
          for movie in top10.index:
              ratinglist.append(ratings.loc[ratings['movie'] == movie]['rating'])
              df.head()

          plt.hist(ratinglist,bins=5)
          plt.show()
```

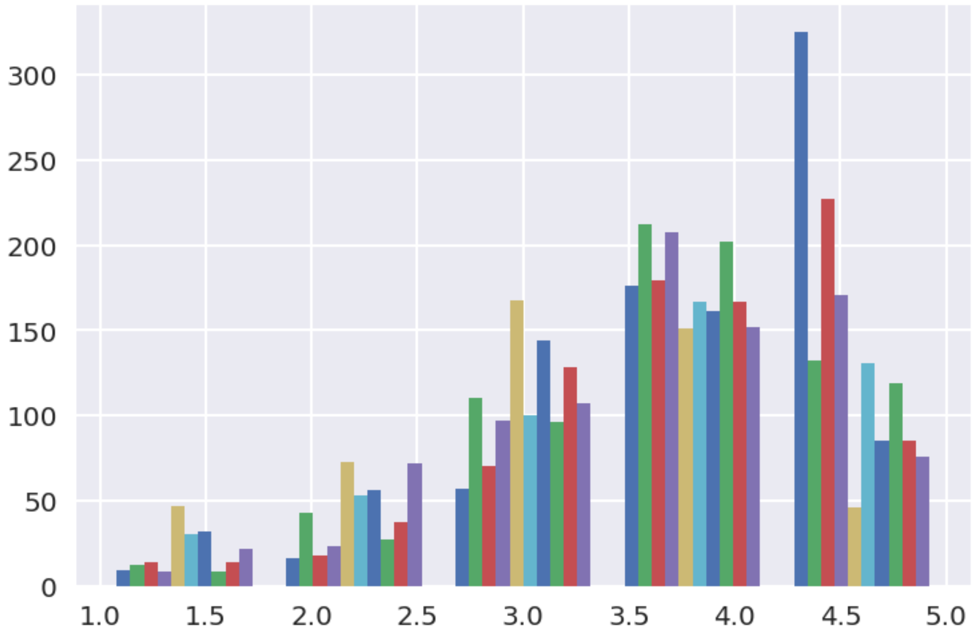
```
/usr/local/lib/python3.5/dist-packages/numpy/core/fromnumeric.py:52: FutureWarning: reshape is
  return getattr(obj, method)(*args, **kwds)
```



1.5 Comparing the top 10 most popular movies

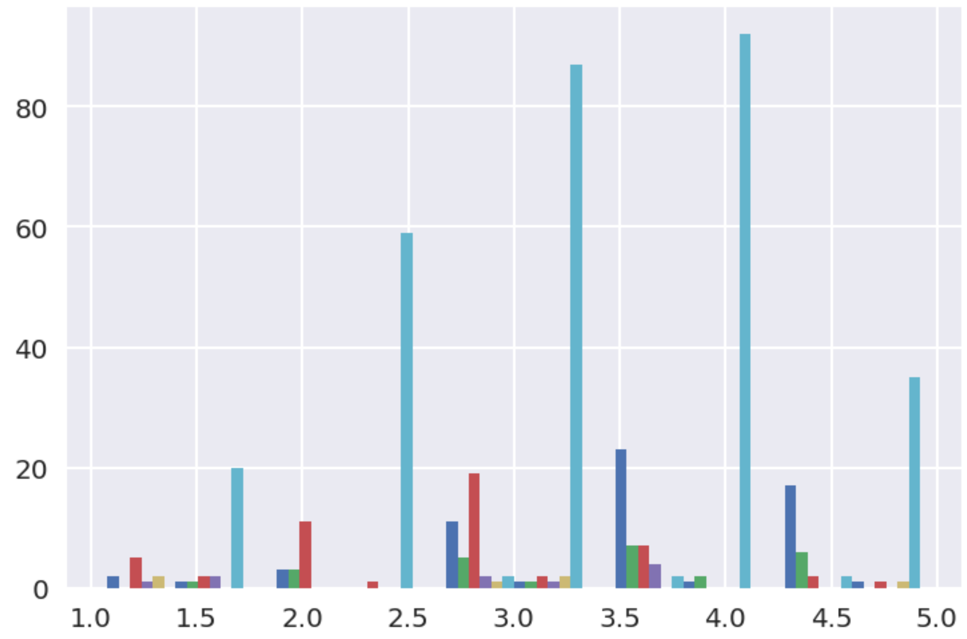
```
In [163]: nreviews = ratings['movie'].value_counts()
          top10popular = nreviews.head(10)

          ratinglist=[]
          for movie in top10popular.index:
              ratinglist.append(ratings.loc[ratings['movie'] == movie]['rating'])
              df.head()

          plt.hist(ratinglist,bins=5)
          plt.show()
```

```
/usr/local/lib/python3.5/dist-packages/numpy/core/fromnumeric.py:52: FutureWarning: reshape is
  return getattr(obj, method)(*args, **kwds)
```



```
In [149]: top10 = avg_ratings['avg_rating'].nlargest(n = 10)

          ratinglist=[]
          for movie in top10.index:
              ratinglist.append(ratings.loc[ratings['movie'] == movie]['rating'])
              df.head()

          plt.hist(ratinglist,bins=5)
          plt.show()
```

```
/usr/local/lib/python3.5/dist-packages/numpy/core/fromnumeric.py:52: FutureWarning: reshape is
  return getattr(obj, method)(*args, **kwds)
```



2 All ratings of movies from three genres of your choice

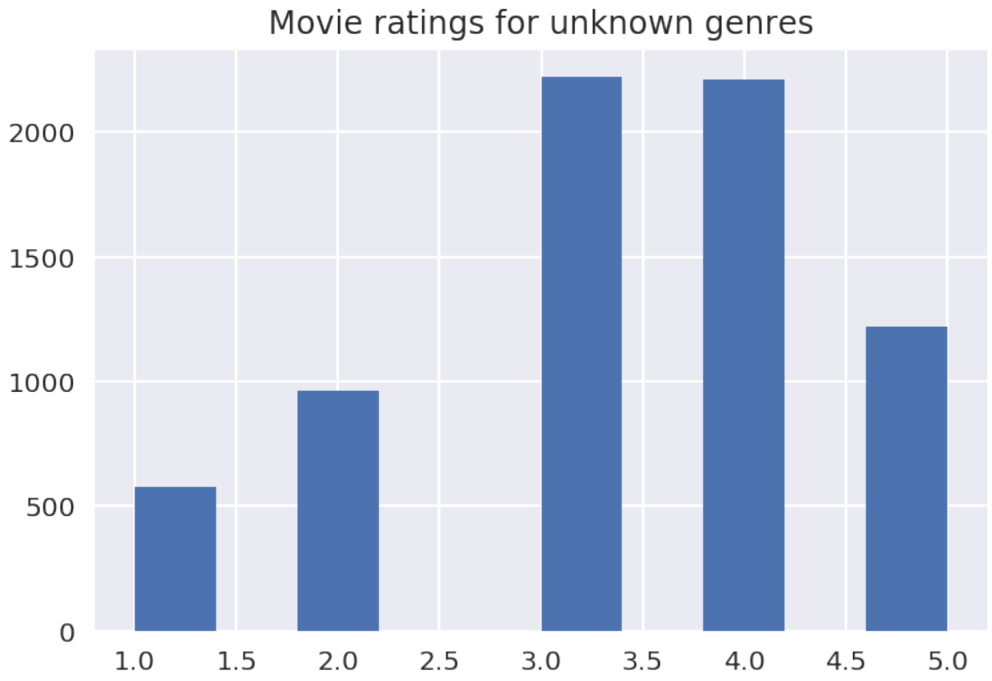
```
In [4]: headers = ['Id','Title', 'Unknown', 'Action', 'Adventure', 'Animation', 'Childrens', 'C
                   'Crime','Documentary', 'Drama', 'Fantasy', 'Noir', 'Horror', 'Musical', 'Mys
                   'Romance', 'Scifi', 'Thriller', 'War', 'Western']
        movies = pd.read_table('data/movies3.txt', header = None, names = headers)
        movies.index = movies.index + 1
        movies.head()
```

```
Out[4]:    Id              Title  Unknown  Action  Adventure  Animation  Childrens  \
        1   1   Toy Story (1995)        0       0          0          1          1
        2   2   GoldenEye (1995)        0       1          1          0          0
        3   3  Four Rooms (1995)        0       0          0          0          0
        4   4  Get Shorty (1995)        0       1          0          0          0
        5   5     Copycat (1995)        0       0          0          0          0

           Comedy  Crime  Documentary   ...     Fantasy  Noir  Horror  Musical  \
        1       1      0            0   ...           0     0       0        0
        2       0      0            0   ...           0     0       0        0
        3       0      0            0   ...           0     0       0        0
        4       1      0            0   ...           0     0       0        0
        5       0      1            0   ...           0     0       0        0

           Mystery  Romance  Scifi  Thriller  War  Western
        1        0        0      0         0    0        0
```

```
2         0          0        0          1     0          0
3         0          0        0          1     0          0
4         0          0        0          0     0          0
5         0          0        0          1     0          0

[5 rows x 21 columns]
```

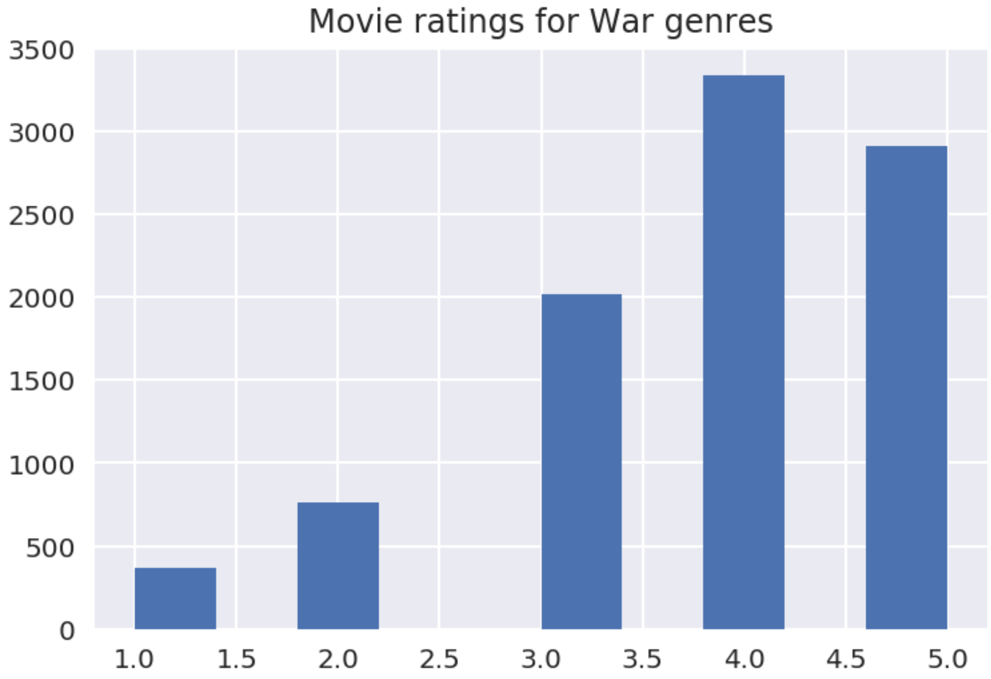
2.0.1 Plotting ratings for `Childrens` genre movies

```
In [9]: unknowns = movies.loc[movies['Childrens'] == 1]#['Title']
        unknown_ratings = ratings.loc[ratings['movie'].isin(unknowns.index)]
        unknown_ratings.hist('rating')
        plt.title('Movie ratings for unknown genres')

Out[9]: Text(0.5,1,'Movie ratings for unknown genres')
```



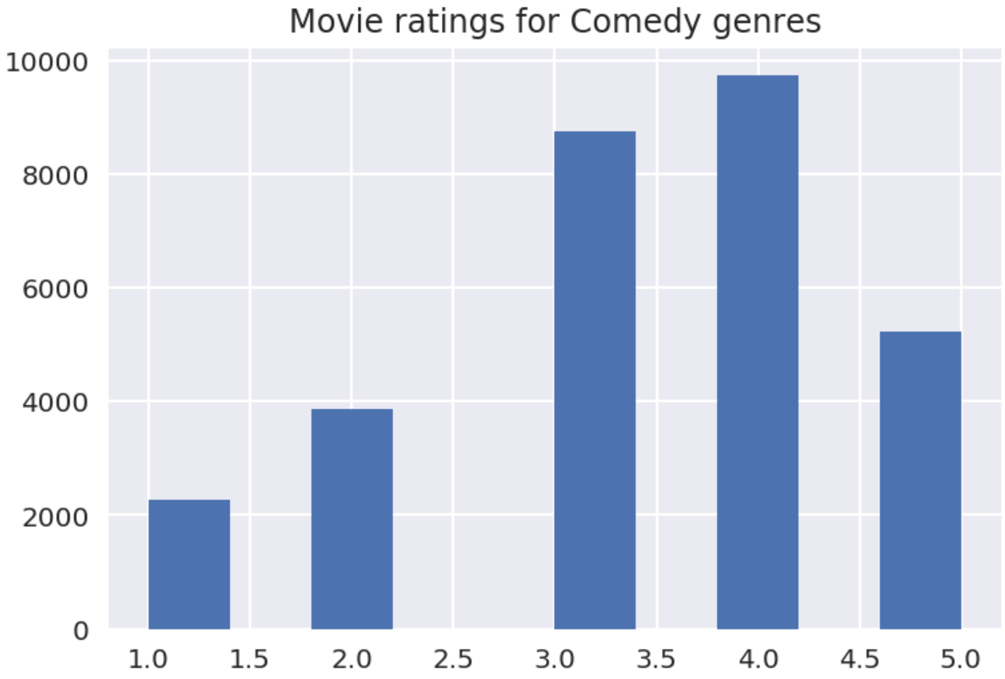
2.0.2 Plotting ratings for `War` genre movies

```
In [209]: wars = movies.loc[movies['War'] == 1]#['Title']
          war_ratings = ratings.loc[ratings['movie'].isin(wars.index)]
          war_ratings.hist('rating')
          plt.title('Movie ratings for War genres')
          #wars.head(100)

Out[209]: Text(0.5,1,'Movie ratings for War genres')
```



2.0.3 Plotting ratings for Comedy genre movies

```
In [212]: comedies = movies.loc[movies['Comedy'] == 1]#['Title']
          comedies_ratings = ratings.loc[ratings['movie'].isin(comedies.index)]
          comedies_ratings.hist('rating')
          plt.title('Movie ratings for Comedy genres')
          #comedies.head(100)

Out[212]: Text(0.5,1,'Movie ratings for Comedy genres')
```



```
In [193]: ratings.loc[ratings['movie'].isin([1373,265])]
```

```
Out[193]:        user  movie  rating
          2172    130    267       5
          3781      5    267       4
          7245    268    267       3
          8567    181   1373       1
          12475   297    267       3
          14756   319    267       4
          15292     1    267       4
          49295   532    267       3
          93523   833    267       1
          99723   422    267       4
```

5 Matrix Factorization Visualizations

In [2]:

```
# Setup
import utils
import matrix_factorization as mf
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
%config InlineBackend.figure_format='retina'
sns.set()
sns.set_style("white")
```

In [4]:

```
Y_train = utils.get_training_data()
Y_test = utils.get_test_data()
Y = utils.get_data()
movie_id, movie_title, movie_genre, genres = utils.get_movies()
genre_similarity = utils.genre_similarity(movie_genre)

M = 943 # users
N = 1682 # movies
K = 20
reg = 0.1
eta = 0.03
```

1. Simple SGD from Homework

In [5]:

```
U_simple, V_simple, err_simple = mf.train_model(Y_train, M, N, K, eta, reg)
err_test_simple = mf.get_err(U_simple, V_simple, Y_test)
err_test_simple /= Y_test.shape[0]
```

```
Epoch  0: current average training error 0.506
Epoch  1: current average training error 0.435
Epoch  2: current average training error 0.416
Epoch  3: current average training error 0.395
Epoch  4: current average training error 0.379
Epoch  5: current average training error 0.373
Epoch  6: current average training error 0.363
Epoch  7: current average training error 0.355
Epoch  8: current average training error 0.346
Epoch  9: current average training error 0.339
Epoch 10: current average training error 0.332
Epoch 11: current average training error 0.327
Epoch 12: current average training error 0.323
Epoch 13: current average training error 0.316
Epoch 14: current average training error 0.314
Epoch 15: current average training error 0.311
Epoch 16: current average training error 0.309
Epoch 17: current average training error 0.301
Epoch 18: current average training error 0.301
```

2. Incorporating a Bias Term

In [7]:

```
U_bias, V_bias, biases, err_bias = mf.train_model(Y_train, M, N, K, eta, reg, include_bias=True)
err_test_bias = mf.get_err(U_bias, V_bias, Y_test, biases=biases)
err_test_bias /= Y_test.shape[0]
```

```
Epoch  0: current average training error 0.441
Epoch  1: current average training error 0.411
Epoch  2: current average training error 0.398
Epoch  3: current average training error 0.387
Epoch  4: current average training error 0.375
Epoch  5: current average training error 0.363
Epoch  6: current average training error 0.354
Epoch  7: current average training error 0.348
Epoch  8: current average training error 0.337
Epoch  9: current average training error 0.329
Epoch 10: current average training error 0.322
Epoch 11: current average training error 0.317
Epoch 12: current average training error 0.310
Epoch 13: current average training error 0.305
Epoch 14: current average training error 0.302
Epoch 15: current average training error 0.299
Epoch 16: current average training error 0.295
Epoch 17: current average training error 0.292
Epoch 18: current average training error 0.289
Epoch 19: current average training error 0.286
Epoch 20: current average training error 0.284
Epoch 21: current average training error 0.281
Epoch 22: current average training error 0.281
```

Off-the-shelf solution

In [8]:

```
import surprise
from surprise import accuracy
from surprise import SVD
from surprise import Reader
from surprise import Dataset

pkf = surprise.model_selection.PredefinedKFold()
reader = Reader(rating_scale=(1, 5))

fulldata = Dataset.load_from_folds([("data/train.txt","data/test.txt")],reader)

surprise_SVD = SVD(n_factors = 20, n_epochs = 30, biased = True)

for trainset, testset in pkf.split(fulldata):

    # train and test algorithm.
    surprise_SVD.fit(trainset)
    predictions = surprise_SVD.test(testset)

    # Compute and print Root Mean Squared Error
    surprise_error = accuracy.rmse(predictions, verbose=True)
```

```
RMSE: 0.9286
```

In [13]:

```
accuracies = pd.DataFrame(columns=["Error"])
accuracies = accuracies.append(pd.Series([err_test_simple], name="Simple SVD", index=["Error"]))
accuracies = accuracies.append(pd.Series([err_test_bias], name="Biased SVD", index=["Error"]))
accuracies = accuracies.append(pd.Series([surprise_error], name="Surprise", index=["Error"]))
accuracies
```

Out[13]:

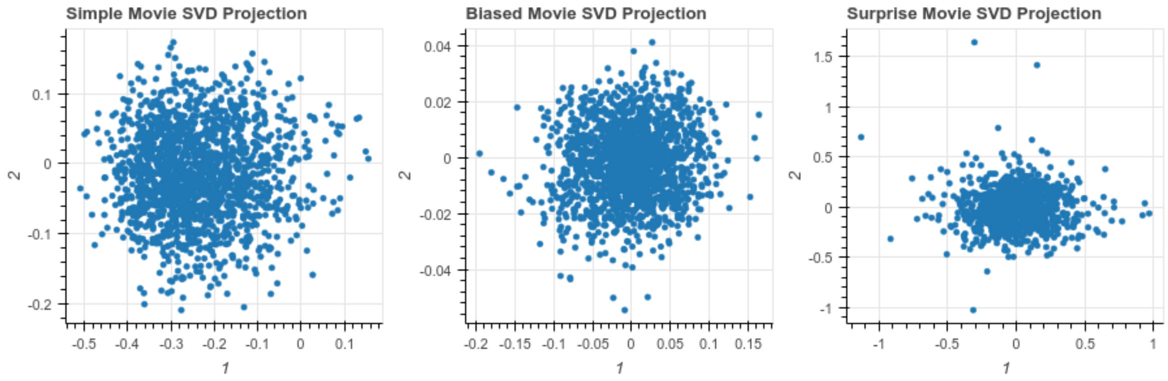
	Error
Simple SVD	0.452937
Biased SVD	0.429117
Surprise	0.926644

Visualization

Movie projections

In [42]:
```
from bokeh.plotting import figure, show
from bokeh.io import output_notebook
from bokeh.layouts import row

def get_SVD_movie_projection(V):

     # SVD for the latent factor of the movies
    A, _, _ = np.linalg.svd(V)

    V_transformed = np.multiply(A[0:1, :].T, V)

    return V_transformed

def get_movie_projection_plot(V, title, size=300):

    plot = figure(plot_width=size, plot_height=size, title=title)
    plot.circle(V[0, :], V[1, :])
    plot.xaxis.axis_label = "1"
    plot.yaxis.axis_label = "2"
    plot.toolbar_location = None

    return plot

V_simple_transformed = get_SVD_movie_projection(V_simple)
V_bias_transformed = get_SVD_movie_projection(V_bias)
V_surprise_transformed = surprise_SVD.pu[:, 0:2].T

simple_SVD_projection_plot = get_movie_projection_plot(V_simple_transformed, "Simple Movie SVD Projection")
bias_SVD_projection_plot = get_movie_projection_plot(V_bias_transformed, "Biased Movie SVD Projection")
off_the_shelf_projection_plot = get_movie_projection_plot(V_surprise_transformed, "Surprise Movie SVD Projection")
output_notebook()
show(row([simple_SVD_projection_plot, bias_SVD_projection_plot, off_the_shelf_projection_plot]))
```

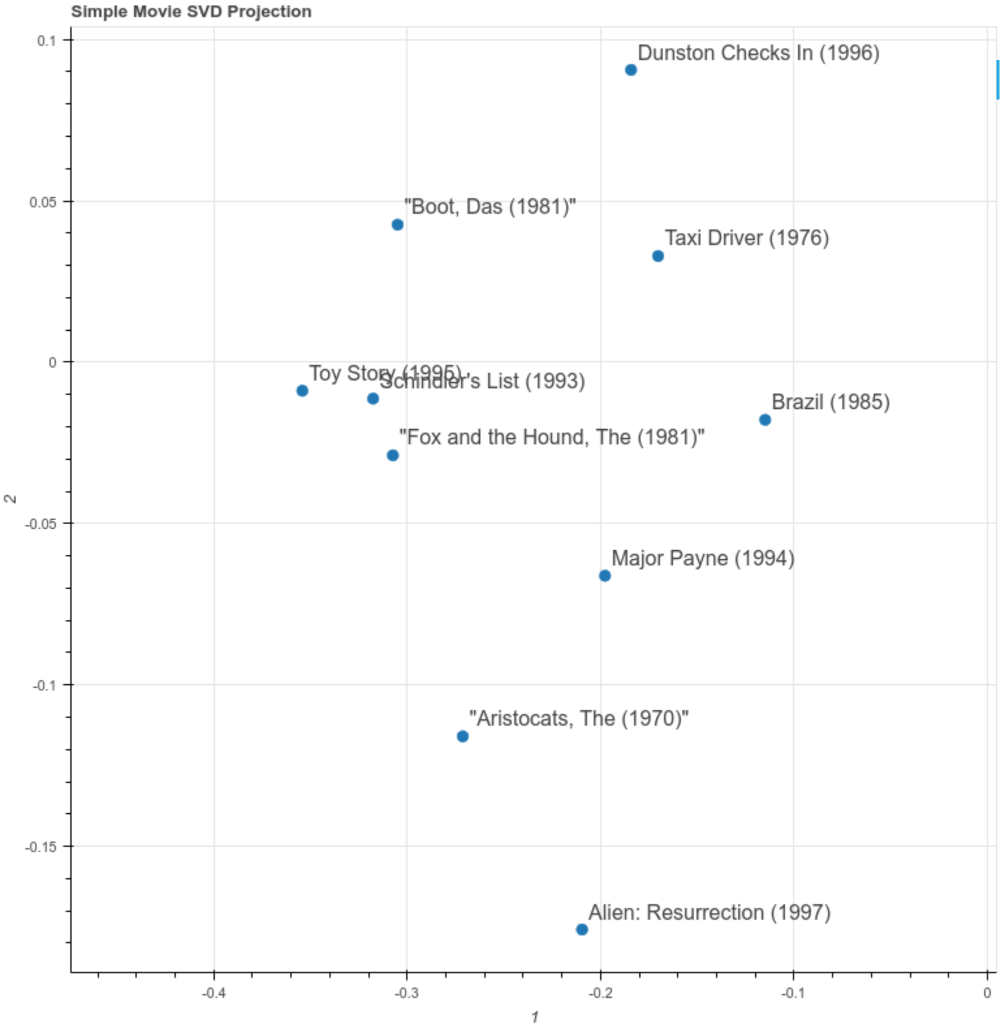
(https://bokeh.pydata.org) BokehJS 0.12.14 successfully loaded.



a) Visualizing 10 Movies

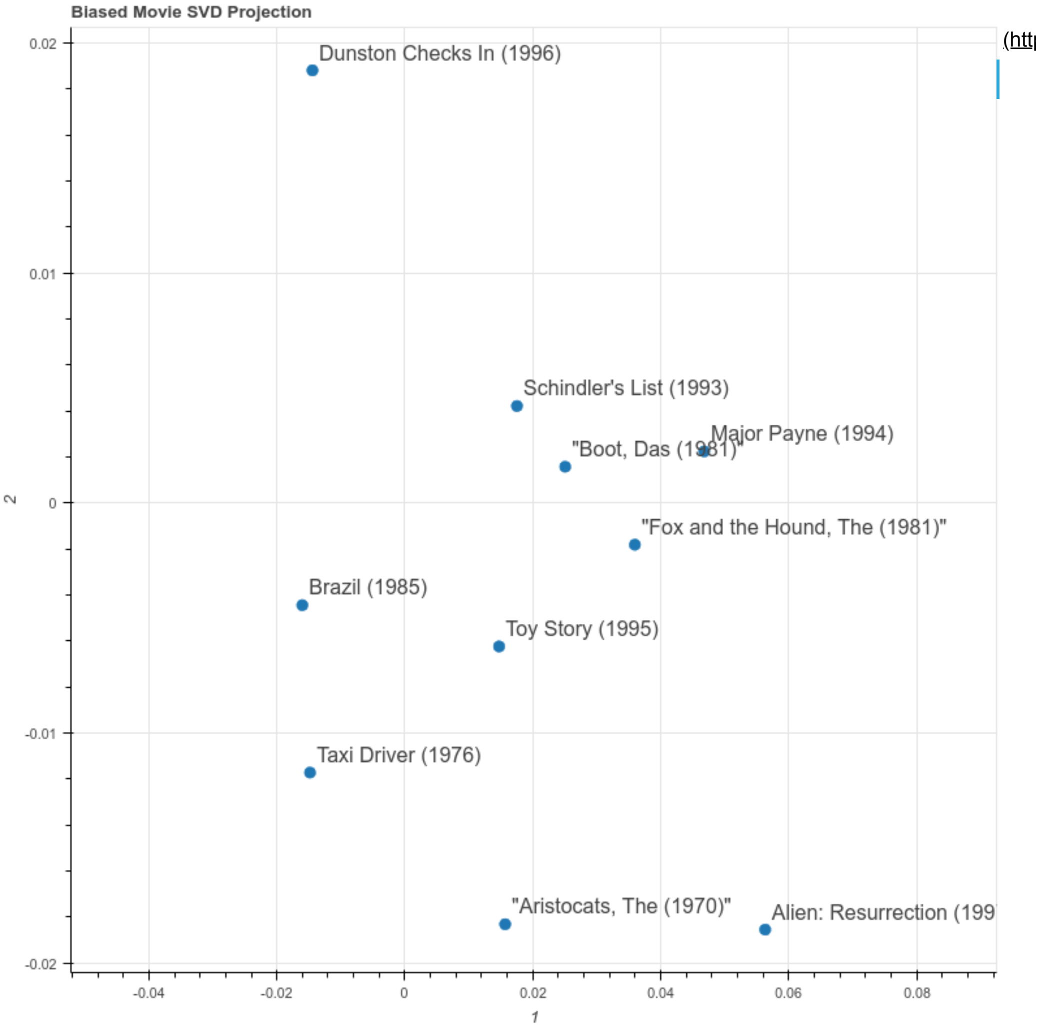
In [134]:

```
movies_of_interest = [
    "\"Aristocats, The (1970)\"", "\"Fox and the Hound, The (1981)\"", "Major Payne (1994)", "Toy Story (1995)", "Du
nston Checks In (1996)",
    "Taxi Driver (1976)", "\"Boot, Das (1981)\"", "Brazil (1985)", "Schindler's List (1993)", "Alien: Resurrection
 (1997)"
]

movies_of_interest_ids = []

for movie_of_interest in movies_of_interest:
    movie_index = list(movie_title).index(movie_of_interest)#np.where(movie_title == movie_of_interest)[0]
    movies_of_interest_ids.append(movie_index)

from bokeh.models import ColumnDataSource, Range1d, LabelSet, Label

def get_custom_movie_projection_plot(coordinates, movie_titles, title, size=450):

    x_range = max(coordinates[0, :]) - min(coordinates[0, :])
    x_min = min(coordinates[0, :]) - x_range * 0.5
    x_max = max(coordinates[0, :]) + x_range * 0.5

    plot = figure(plot_width=size, plot_height=size, title=title, x_range=(x_min, x_max))
    plot.circle(coordinates[0, :], coordinates[1, :])
    plot.xaxis.axis_label = "1"
    plot.yaxis.axis_label = "2"
    source = ColumnDataSource(data=dict(dim1=coordinates[0, :],
                                        dim2=coordinates[1, :],
                                        names=movie_titles))

    plot.scatter(x='dim1', y='dim2', size=8, source=source)

    labels = LabelSet(x='dim1', y='dim2', text='names', level='glyph',
                  x_offset=5, y_offset=5, source=source, render_mode='canvas')
    plot.add_layout(labels)

    return plot

simple_SVD_custom_movie_plot = get_custom_movie_projection_plot(V_simple_transformed[:, movies_of_interest_ids], mov
ies_of_interest, "Simple Movie SVD Projection", size=800)
show(simple_SVD_custom_movie_plot)
```


Simple Movie SVD Projection
Dunston Checks In (1996)
"Boot, Das (1981)"
Taxi Driver (1976)
Toy Story (1995)
Schindler's List (1993)
Brazil (1985)
"Fox and the Hound, The (1981)"
Major Payne (1994)
"Aristocats, The (1970)"
Alien: Resurrection (1997)
0.1
0.05
0
-0.05
-0.1
-0.15
2
-0.4
-0.3
-0.2
-0.1
0
1

(htt

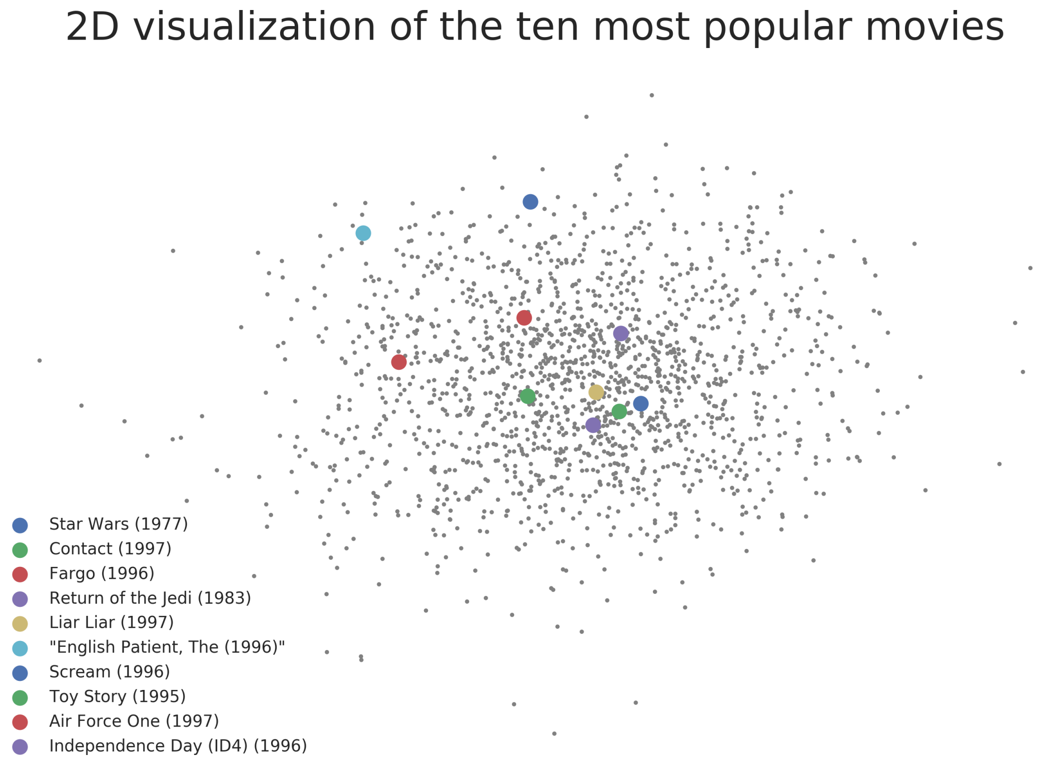
```
In [135]: bias_SVD_custom_movie_plot = get_custom_movie_projection_plot(V_bias_transformed[:, movies_of_interest_ids], movies_
          of_interest, "Biased Movie SVD Projection", size=800)
          show(bias_SVD_custom_movie_plot)
```



b) Most popular movies

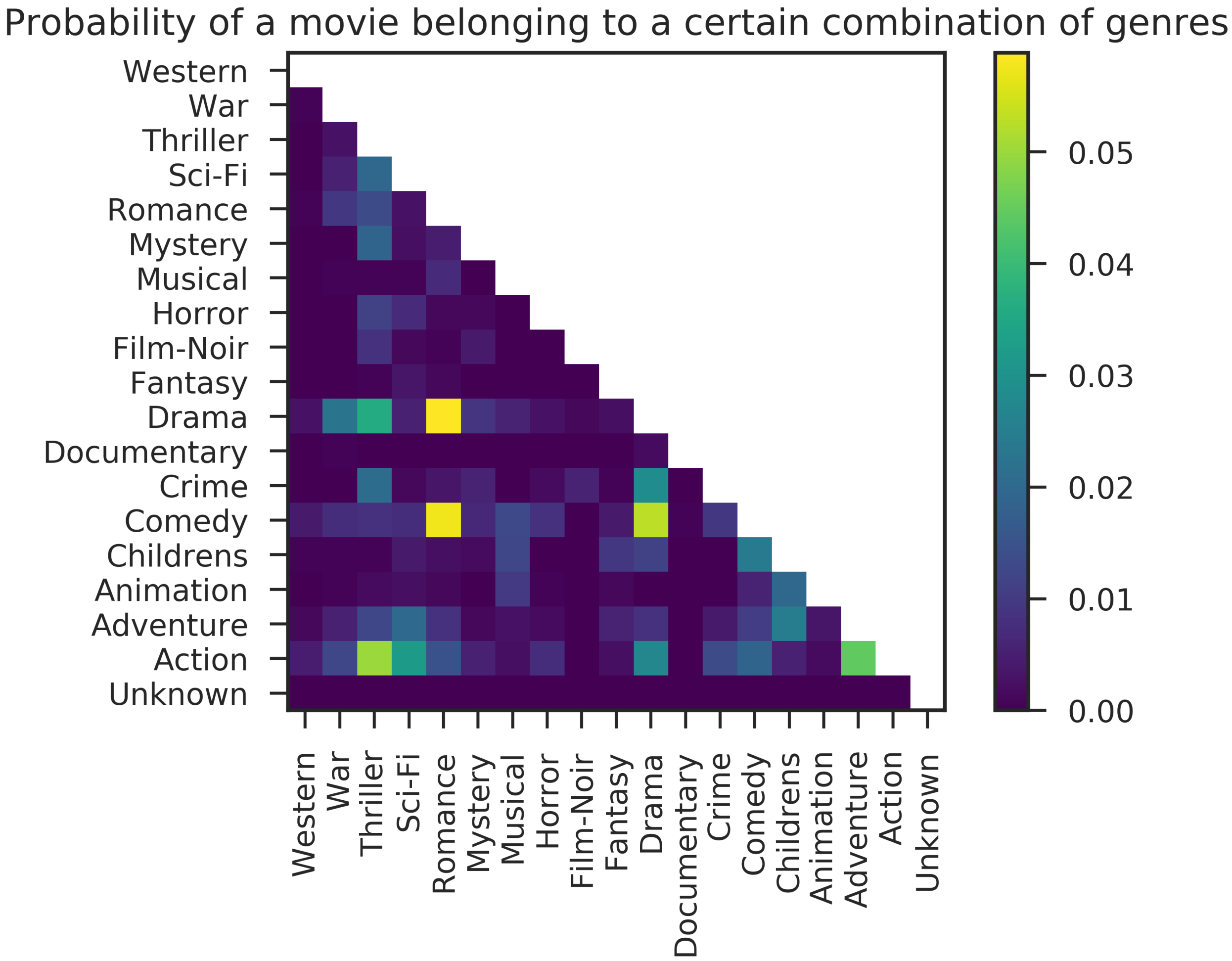
```
In [62]: YM = utils.list_to_matrix(Y, M, N)
         counts, ratings, ratings_bayesian = utils.bayesian_rating(Y, thr=4)
         rank_counts = np.argsort(-counts) # Indices of counts in descending order
         rank_ratings = np.argsort(-ratings) # Indices of ratings in descending order
         rank_ratings_bayesian = np.argsort(-ratings_bayesian) # Indices of ratings in descending order
```

In [67]:
```
plt.figure(dpi=300)
plt.scatter(V_bias_transformed[0, :], V_bias_transformed[1, :], 1, 'gray')
for idx in rank_counts[:10]:
    plt.scatter(V_bias_transformed[0, idx], V_bias_transformed[1, idx], 20, label=movie_title[idx])
plt.axis('off')
plt.legend(prop={'size': 5})
plt.title('2D visualization of the ten most popular movies')
plt.show()
```

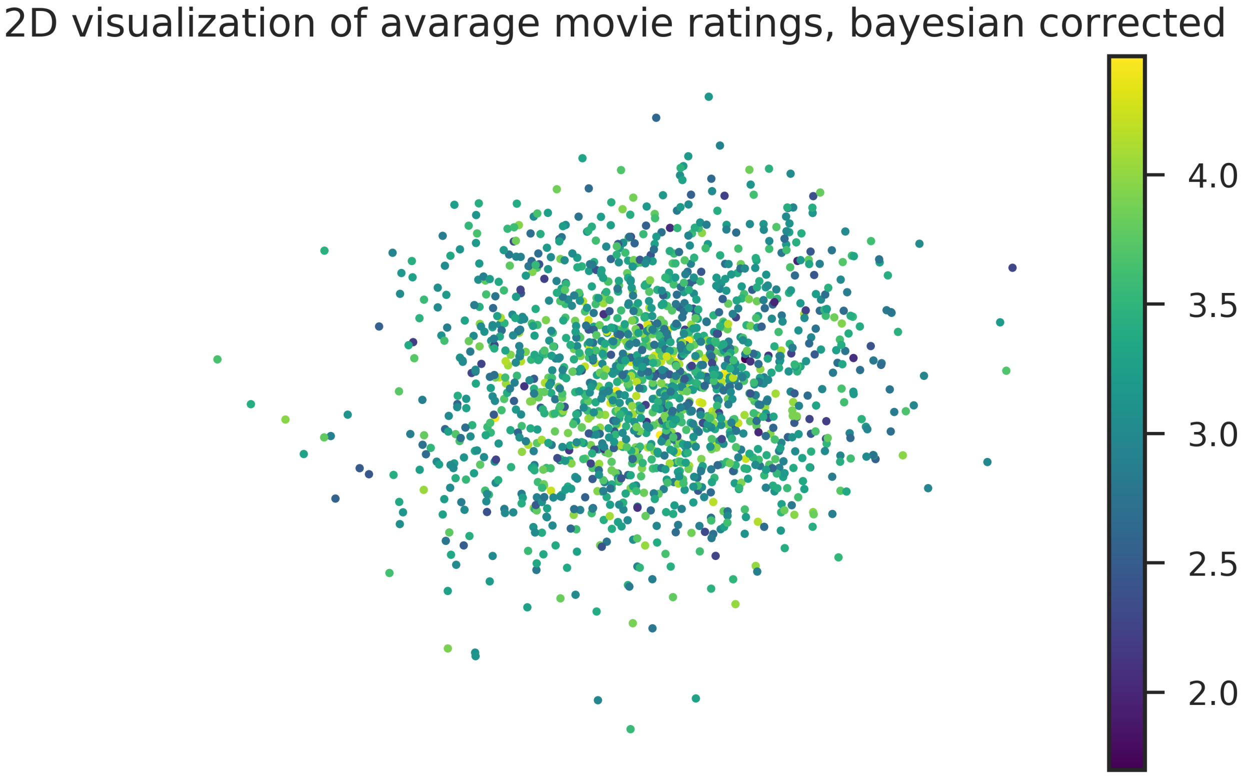


c) Ten Best Movies

In [68]:
```
sns.set_style("ticks")
n_genres = len(genres)
genre_similarity_nodiag = genre_similarity.copy()
for i in range(n_genres):
    genre_similarity_nodiag[i, i] = np.nan
plt.figure(dpi=300)
plt.imshow(np.rot90(genre_similarity_nodiag, 2), extent=[0.5, n_genres+0.5, 0.5, n_genres+0.5],
           cmap='viridis')
plt.xticks(np.arange(n_genres)+1, genres[::-1], rotation='vertical')
plt.yticks(np.arange(n_genres)+1, genres)
plt.colorbar()
plt.title('Probability of a movie belonging to a certain combination of genres')
plt.show()
```

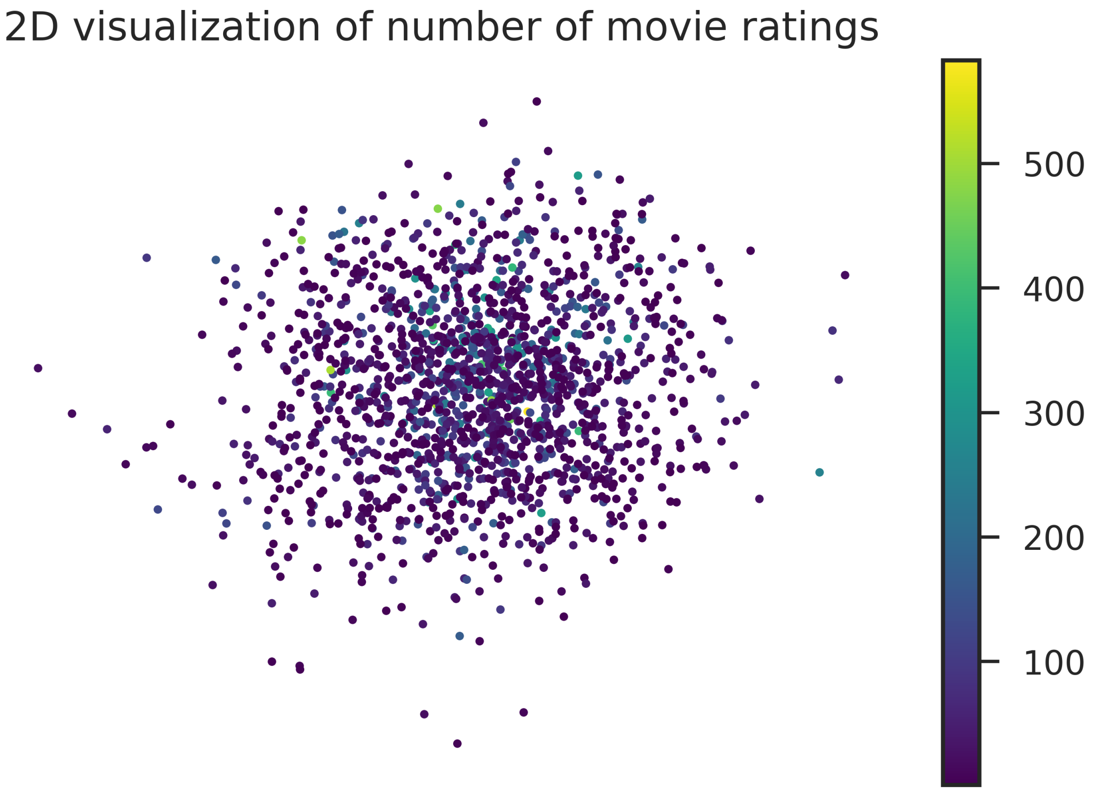


In [69]:
```
plt.figure(dpi=300)
plt.scatter(V_bias_transformed[0, :], V_bias_transformed[1, :], 5, ratings_bayesian, cmap='viridis')
plt.title('2D visualization of avarage movie ratings, bayesian corrected')
plt.colorbar()
plt.axis('off')
plt.show()
```



In [71]:

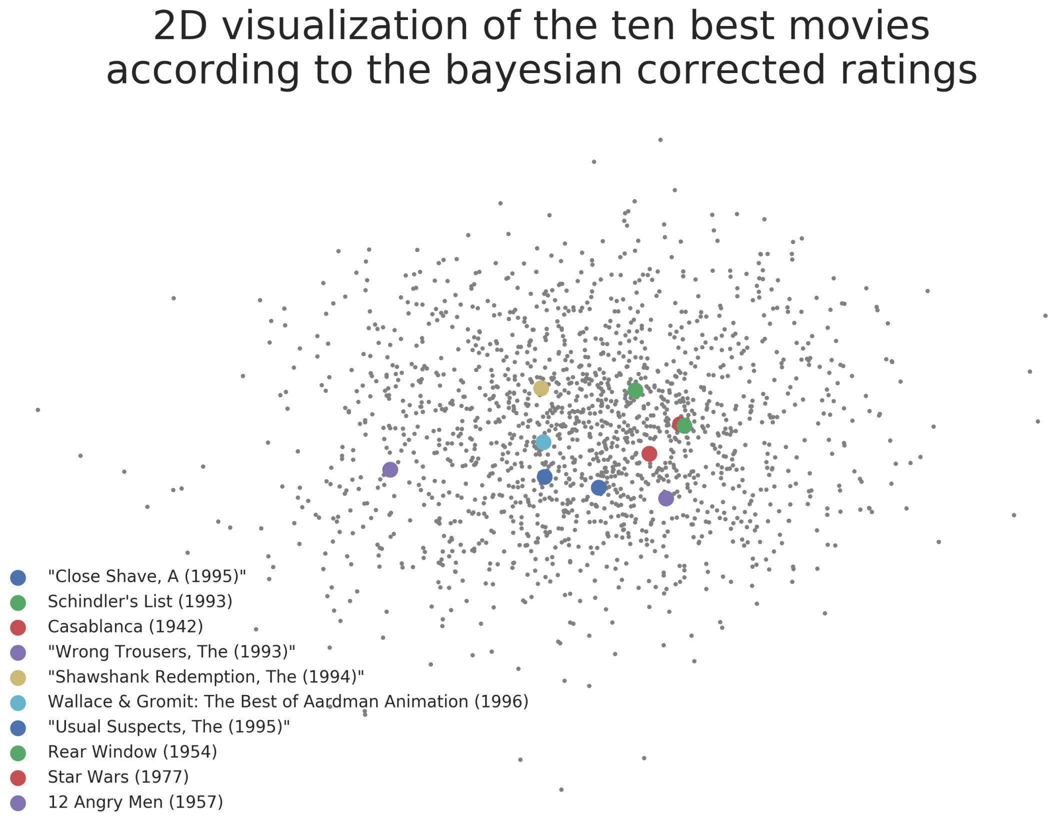
```
plt.figure(dpi=300)
plt.scatter(V_bias_transformed[0, :], V_bias_transformed[1, :], 5, counts, cmap='viridis')
plt.title('2D visualization of number of movie ratings')
plt.colorbar()
plt.axis('off')
plt.show()
```



2D visualization of number of movie ratings

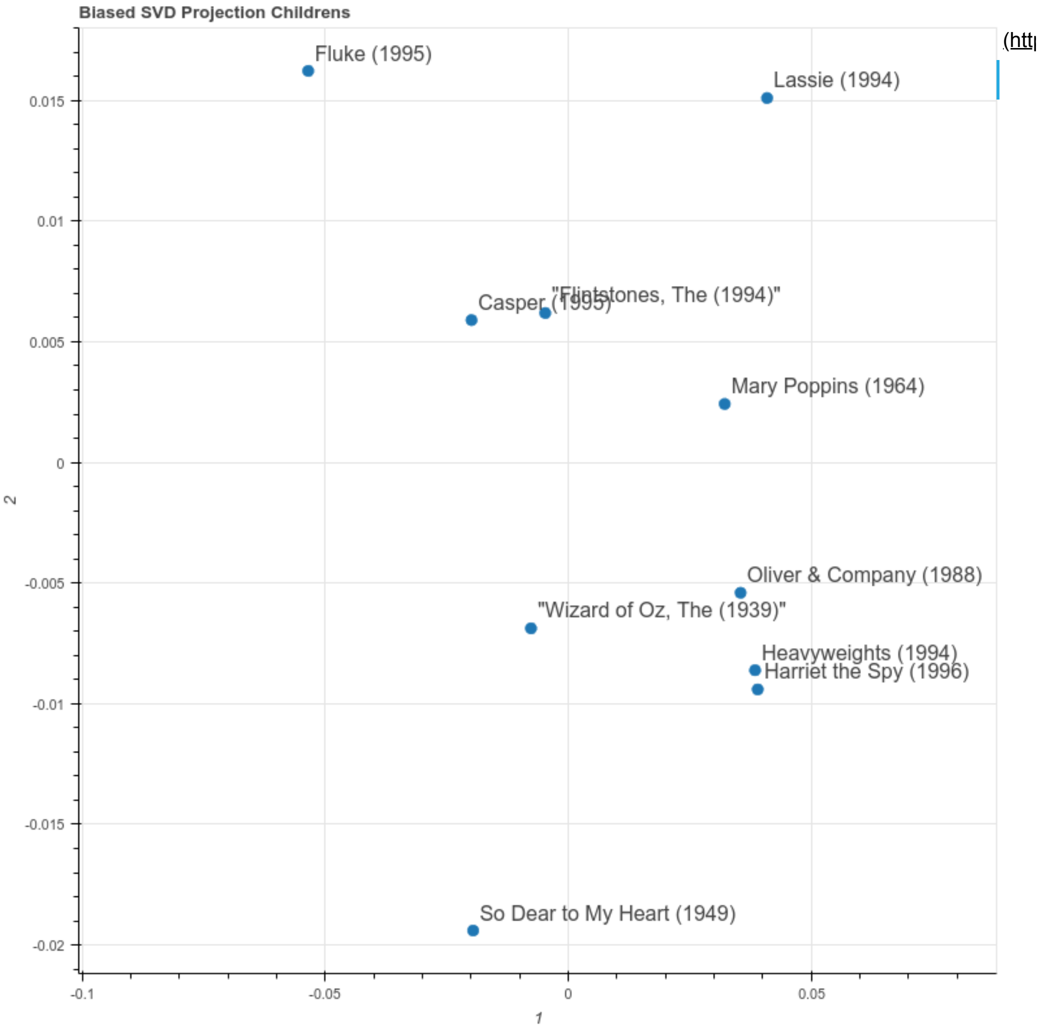
In [78]:

```
plt.figure(dpi=300)
plt.scatter(V_bias_transformed[0, :], V_bias_transformed[1, :], 1, 'gray')
for idx in rank_ratings_bayesian[:10]:
    plt.scatter(V_bias_transformed[0, idx], V_bias_transformed[1, idx], 20, label=movie_title[idx])
plt.axis('off')
plt.legend(prop={'size': 5})
plt.title('2D visualization of the ten best movies\naccording to the bayesian corrected ratings')
plt.show()
```

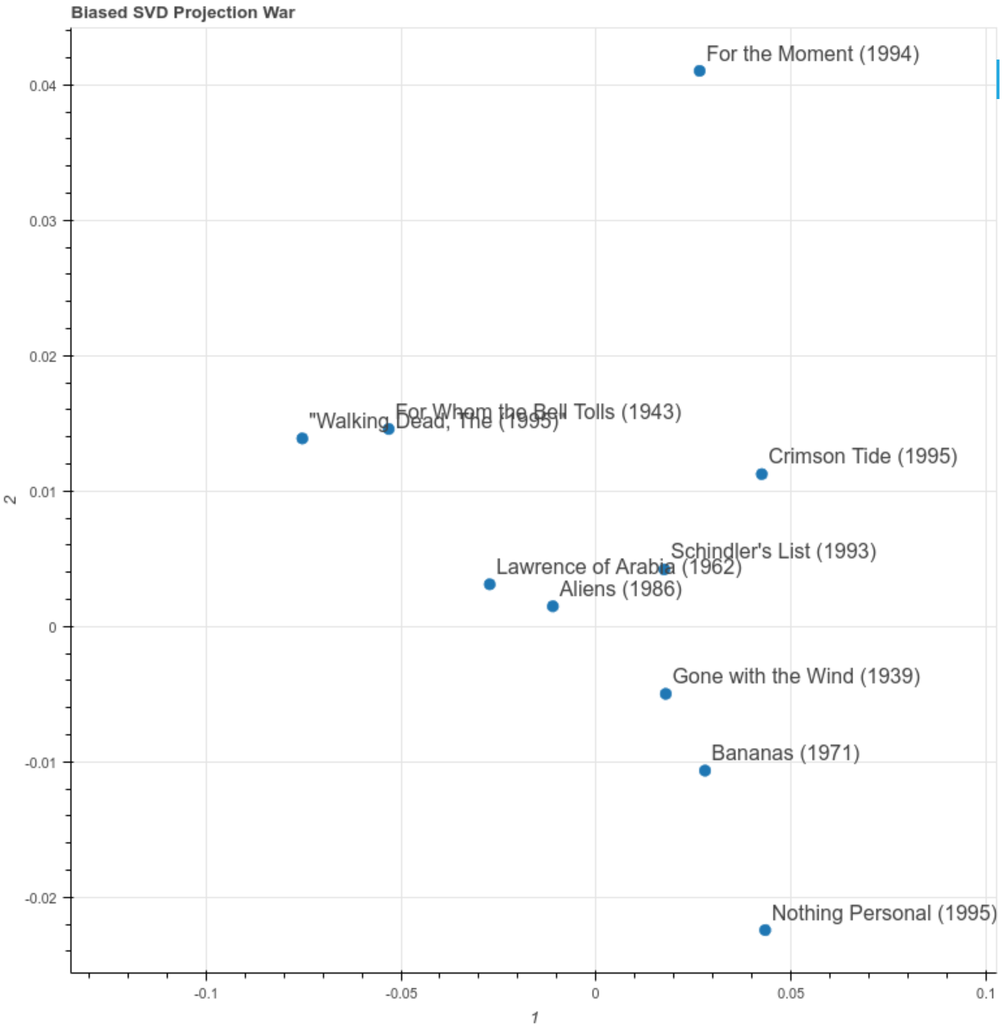


Movies from 3 genres

In [136]:
```
def get_random_genre_projection_plot(genre, title):

    genre_movie_ids = movie_id[np.where(movie_genre[:, np.where(genres == genre)[0][0]] == 1)] - 1
    genre_movie_ids = np.random.choice(genre_movie_ids, 10)
    genre_movie_titles = movie_title[genre_movie_ids]

    plot = get_custom_movie_projection_plot(V_bias_transformed[:, genre_movie_ids], genre_movie_titles, "%s %s" % (t
itle, genre), size=800)
    return plot

plot = get_random_genre_projection_plot("Childrens", "Biased SVD Projection")
show(plot)
```

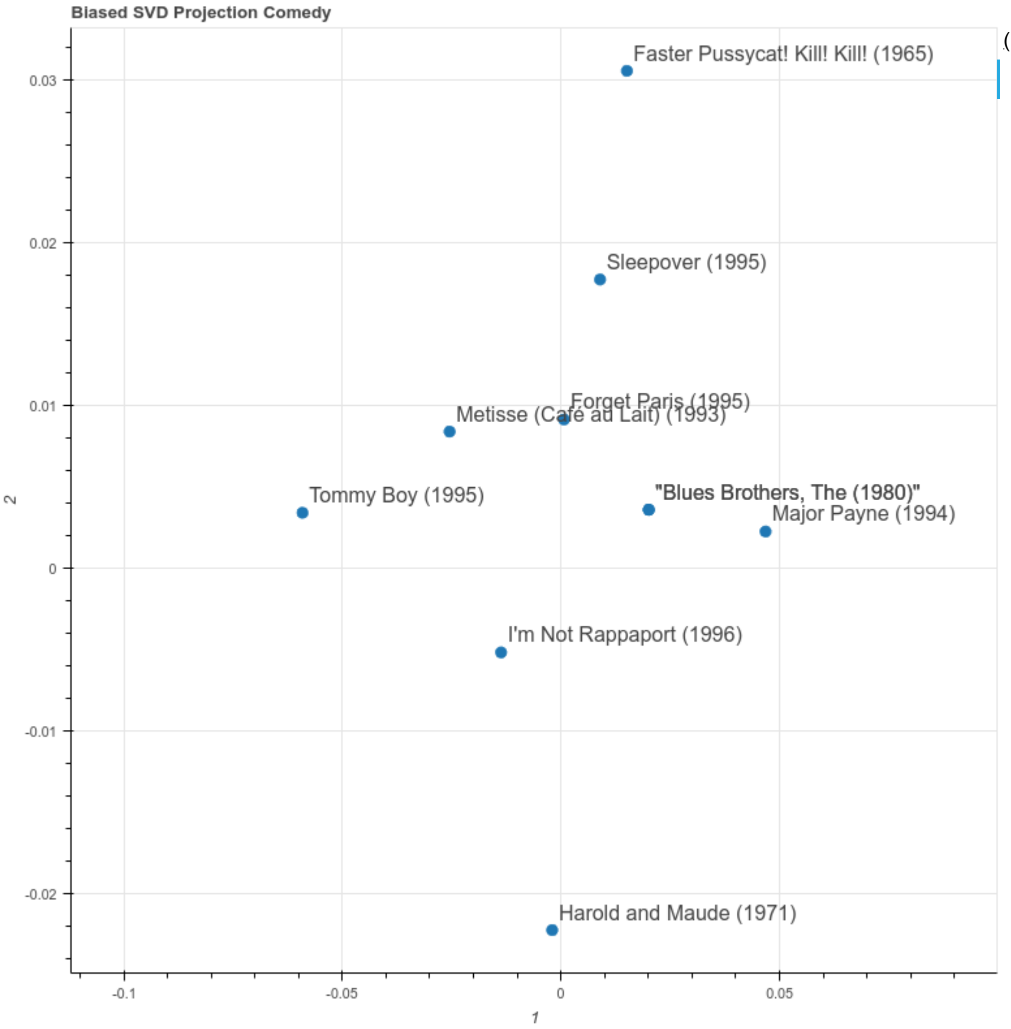


In [137]:
```
plot = get_random_genre_projection_plot("War", "Biased SVD Projection")
show(plot)
```



(htt

```
In [138]: plot = get_random_genre_projection_plot("Comedy", "Biased SVD Projection")
          show(plot)
```



(htt

Custom Visualizations

In [1]:

```
# Setup
import utils
import matrix_factorization as mf
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
%config InlineBackend.figure_format='retina'
sns.set()
sns.set_style("white")
```

In [20]:

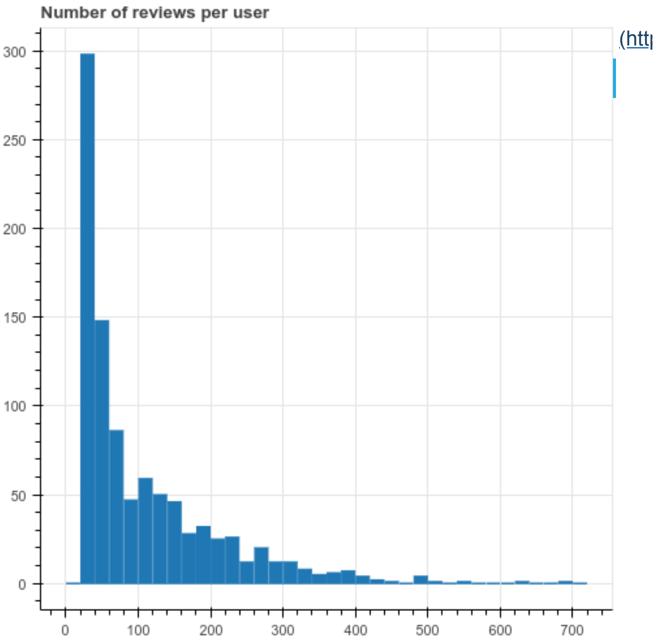
```
Y_train = utils.get_training_data()
Y_test = utils.get_test_data()
Y = utils.get_data()
movie_id, movie_title, movie_genre, genres = utils.get_movies()
genre_similarity = utils.genre_similarity(movie_genre)

M = 943 # users
N = 1682 # movies
K = 20
reg = 0.1
eta = 0.03
num_genres = len(genres)
```

Quantifying Critics

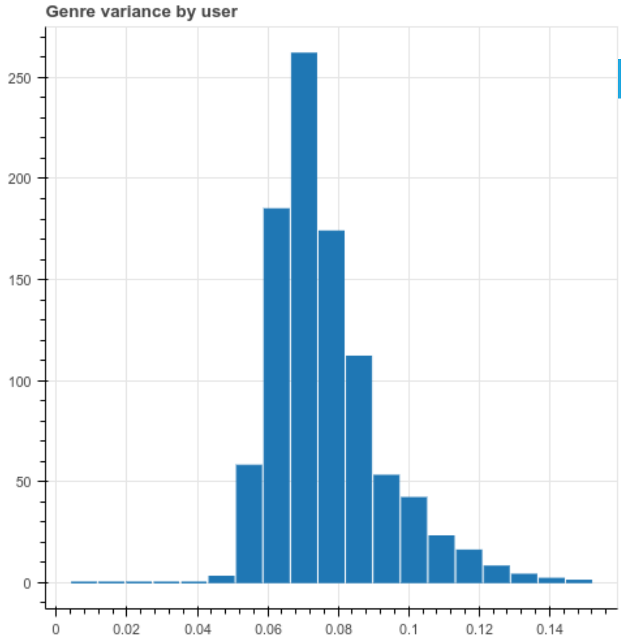
In [33]:

```
num_user_reviews = np.zeros((M, ))

for rating_index in range(Y.shape[0]):
    user_index = Y[rating_index][0] - 1
    num_user_reviews[user_index] += 1

from bokeh.plotting import figure, show
from bokeh.io import output_notebook

max_num_reviews = max(num_user_reviews)
bin_size = 20
bin_edges = list(range(0, int(max_num_reviews)+1, int(bin_size)))
bin_centers = np.array(bin_edges[1:]) - int(bin_size/2)
user_review_counts, _ = np.histogram(num_user_reviews, bins=bin_edges)
plot = figure(title="Number of reviews per user", plot_width=500, plot_height=500)
plot.vbar(x=bin_centers, top=user_review_counts, width=0.9*bin_size)

output_notebook()
show(plot)
```

(https://bokeh.pydata.org) BokehJS 0.12.14 successfully loaded.



In [95]:
```
num_user_reviews_by_genre = np.zeros((M, num_genres))

for rating_index in range(Y.shape[0]):
    user_index = Y[rating_index][0] - 1
    movie_index = Y[rating_index][1] - 1

    rating_genres = movie_genre[movie_index]

    # Normalize the rating genres
    rating_genres = rating_genres / sum(rating_genres)

    # Update the users genre rating row

    num_user_reviews_by_genre[user_index, :] = num_user_reviews_by_genre[user_index, :] + rating_genres

user_genre_variance = np.zeros((M, ))
for user_index in range(M):
    num_user_reviews_by_genre[user_index, :] = num_user_reviews_by_genre[user_index, :] / sum(num_user_reviews_by_genre
    user_genre_variance[user_index] = np.std(num_user_reviews_by_genre[user_index, :])


max_variance = max(user_genre_variance)
bin_edges = np.linspace(0, max_variance, 20)
bin_size = bin_edges[1] - bin_edges[0]
bin_centers = np.array(bin_edges[1:]) - int(bin_size/2)
counts, _ = np.histogram(user_genre_variance, bins=bin_edges)
plot = figure(title="Genre variance by user", plot_width=500, plot_height=500)
plot.vbar(x=bin_centers, top=counts, width=0.9*bin_size)

show(plot)
```



(htt

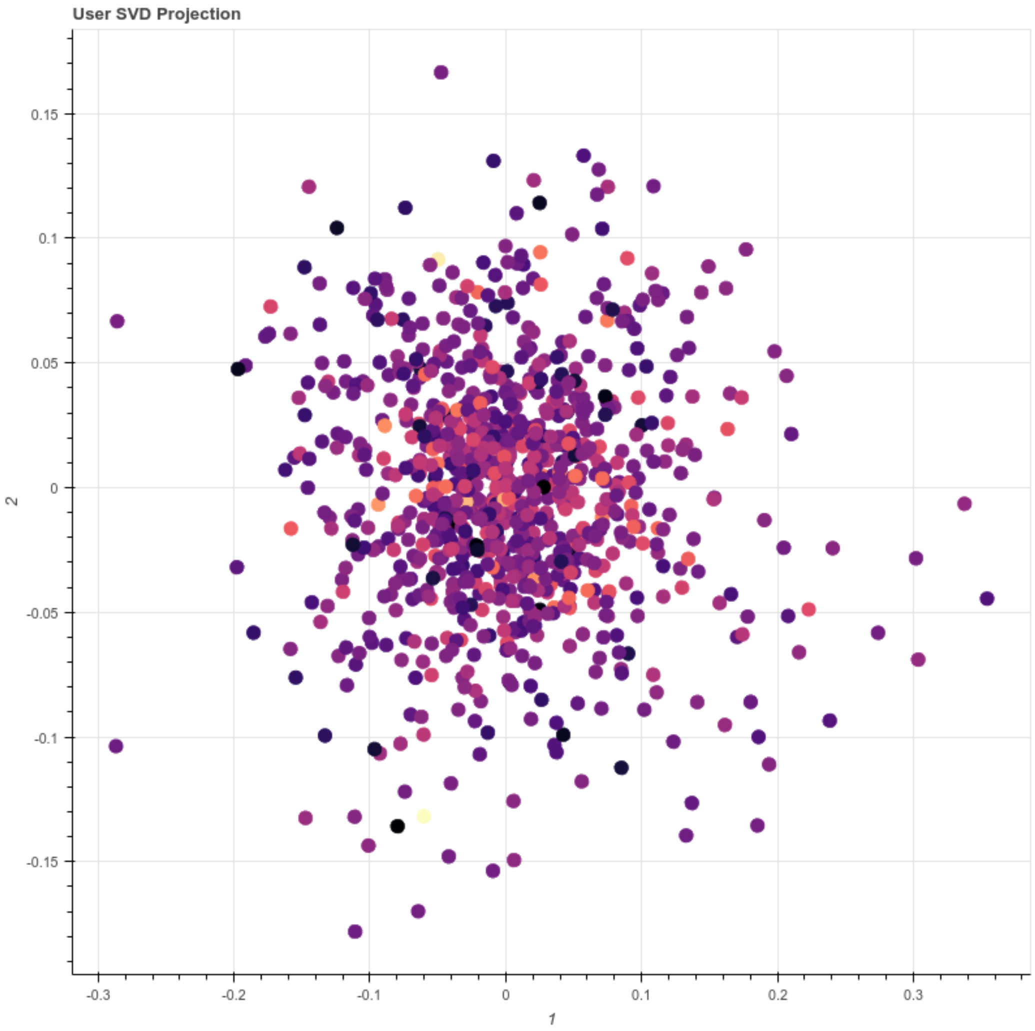
In [96]:

```
U_bias, V_bias, biases, err_bias = mf.train_model(Y, M, N, K, eta, reg, include_bias=True)
err_test_bias = mf.get_err(U_bias, V_bias, Y_test, biases=biases)
err_test_bias /= Y_test.shape[0]
```

```
Epoch  0: current average training error 0.443
Epoch  1: current average training error 0.415
Epoch  2: current average training error 0.401
Epoch  3: current average training error 0.391
Epoch  4: current average training error 0.380
Epoch  5: current average training error 0.372
Epoch  6: current average training error 0.363
Epoch  7: current average training error 0.353
Epoch  8: current average training error 0.347
Epoch  9: current average training error 0.341
Epoch 10: current average training error 0.333
Epoch 11: current average training error 0.329
Epoch 12: current average training error 0.321
Epoch 13: current average training error 0.318
Epoch 14: current average training error 0.315
Epoch 15: current average training error 0.311
Epoch 16: current average training error 0.308
Epoch 17: current average training error 0.304
Epoch 18: current average training error 0.301
Epoch 19: current average training error 0.298
Epoch 20: current average training error 0.295
Epoch 21: current average training error 0.297
```

In [110]:

```
from bokeh.models import ColumnDataSource, LinearColorMapper

def get_SVD_user_projection(U):

     # SVD for the latent factor of the movies
    A, _, _ = np.linalg.svd(U)

    U_transformed = np.multiply(A[0:1, :].T, U)

    return U_transformed

def get_user_projection_plot(U, title, values, size=300):

    plot = figure(plot_width=size, plot_height=size, title=title)
    plot.xaxis.axis_label = "1"
    plot.yaxis.axis_label = "2"
    plot.toolbar_location = None

    data_source = ColumnDataSource({"x": U[0, :], "y": U[1, :], "values": values})

    color_mapper = LinearColorMapper(palette='Magma256', low=min(values), high=max(values))
    plot.circle(x="x", y="y", color={'field': 'values', 'transform': color_mapper}, source=data_source, size=10)

    return plot

critic_rank =  (1 - user_genre_variance / max(user_genre_variance)) + (num_user_reviews / max(num_user_reviews))

U_bias_transformed = get_SVD_user_projection(U_bias)
plot = get_user_projection_plot(U_bias_transformed, "User SVD Projection", critic_rank, size=800)

show(plot)
```



In [ ]: