100_playlist_baseline_model

December 12, 2018

```
In [1]: import pandas as pd
        import numpy as np
        from sklearn.model_selection import train_test_split
        from sklearn.neighbors import NearestNeighbors
        from scipy.sparse import csr_matrix

        subset100 = pd.read_csv("../raw_data/track_meta_100subset_new.csv")
```

0.0.1 Train-val-test split

```
In [5]: # Train-val-test split (20%)
        train, test = train_test_split(subset100, test_size=0.2, random_state=42, stratify = su
        train, val = train_test_split(train, test_size=0.2, random_state=42, stratify = train[
```

```
In [6]: test.head()
```

```
Out[6]:       Playlistid  Trackid      Artist_Name                      Track_Name  \
        557        38828       35         Bastille                         Pompeii
        556        38828       34   Britney Spears                       Womanizer
        2414      229646        7        Soft Cell                     Tainted Love
        1771      186672       28  Imagine Dragons                      Radioactive
        516        37634       17             LANY  WHERE THE HELL ARE MY FRIENDS

                              Album_Name  Track_Duration  \
        557                     Bad Blood          214147
        556         Circus (Deluxe Version)          224400
        2414        Non-Stop Erotic Cabaret          153762
        1771                  Night Visions          186813
        516   WHERE THE HELL ARE MY FRIENDS          216180

                                      Artist_uri  \
        557   spotify:artist:7EQ0qTo7fWT7DPxmxtSYEc
        556   spotify:artist:26dSoYclwsYLMAKD3tpOr4
        2414  spotify:artist:6aq8T2RcspxVOGgMrTzjWc
        1771  spotify:artist:53XhwfbYqKCa1cC15pYq2q
        516   spotify:artist:49tQo2QULno7gxHutgccqF

                                       Track_uri  \
```

```
557   spotify:track:3gbBpTdY8lnQwqxNCcf795
556   spotify:track:4fixebDZAVToLbUCuEloa2
2414  spotify:track:0cGG2EouYCEEC3xfa0tDFV
1771  spotify:track:6Ep6BzIOB9tz3P4sWqiiAB
516   spotify:track:4TA2nSix6i8K2VV9wt6rUn

                                Album_uri  acousticness   ...   loudness  \
557   spotify:album:64fQ94AVziavTPdnkCS6Nj        0.0755   ...     -6.383
556   spotify:album:2tve5DGwub1TtbX1khPX5j        0.0730   ...     -5.226
2414  spotify:album:3KFWViJ1wIHAdOVLFTVzjD        0.4620   ...     -8.284
1771  spotify:album:1vAEF8FOHoRFGiYOEeJXHW        0.1190   ...     -3.698
516   spotify:album:34ySll9UQXpSngEIONJbFO        0.0652   ...     -3.811

      mode  speechiness    tempo  time_signature  valence   Playlist  Album  \
557      1       0.0407  127.435               4    0.571         tb     55
556      1       0.0622  139.000               4    0.235         tb     55
2414     0       0.0378  144.435               4    0.623  Throwback    121
1771     1       0.0590  136.249               4    0.210   campfire     30
516      1       0.0344  127.994               4    0.472   not sure     16

      Track  Artist
557      63      44
556      63      44
2414    135      91
1771     34      29
516      23      13

[5 rows x 28 columns]
```

0.0.2 kNN Collaborative Filtering

```
In [7]: # Create Binary Sparse Matrix
        co_mat = pd.crosstab(train.Playlistid, train.Track_uri)
        co_mat = co_mat.clip(upper=1)
        assert np.max(co_mat.describe().loc['max']) == 1

        co_mat_sparse = csr_matrix(co_mat)
```

```
In [8]: # Train kNN model
        col_filter = NearestNeighbors(metric='cosine', algorithm='brute')
        col_filter.fit(co_mat_sparse)
```

```
Out[8]: NearestNeighbors(algorithm='brute', leaf_size=30, metric='cosine',
                metric_params=None, n_jobs=None, n_neighbors=5, p=2, radius=1.0)
```

0.1 Making Predictions

```
In [9]: def nholdout(playlist_id, df):
            '''Pass in a playlist id to get number of songs held out in val/test set'''
```

```
            return len(df[df.Playlistid == playlist_id].Track_uri)

        def kpredict(knnmodel, playlist_id, df):
            '''for a playlist id, generate list of 15*k predictions where k is num holdouts'''

            k = nholdout(playlist_id, df)*15 # number of holdouts
            ref_songs = co_mat.columns.values[co_mat.loc[playlist_id] == 1] # songs already in
            dist, ind = knnmodel.kneighbors(np.array(co_mat.loc[playlist_id]).reshape(1, -1), n
            rec_ind = co_mat.index[ind[0]] # recommended playlists

            n_pred = 0
            pred = []
            for i in rec_ind:
                new_songs = co_mat.columns.values[co_mat.loc[i] == 1] # potential recommendati
                for song in new_songs:
                    if song not in ref_songs: # only getting songs not already in target playl
                        pred.append(song)
                        n_pred += 1
                        if n_pred == k:
                            break
                if n_pred == k:
                    break

            return pred
```

```
In [14]: ### Prediction Example
         pi = 430 # target playlist index
         kpreds = kpredict(col_filter, pi, val) # list of predictions
```

```
In [23]: val_set = val[val.Playlistid == pi]
         val_set = val_set['Track_uri'] # ground truth
```

0.2 Metrics

```
In [24]: def r_precision(prediction, val_set):
         # prediction should be a list of predictions
         # val_set should be pandas Series of ground truths
             score = np.sum(val_set.isin(prediction))/val_set.shape[0]
             return score
```

```
In [25]: ### Example Usage
         r_precision(kpreds, val_set)
```

```
Out[25]: 0.0
```

```
In [26]: ### NDCG Code Source: https://gist.github.com/bwhite/3726239
         def dcg_at_k(r, k, method=0):
             r = np.asfarray(r)[:k]
```

```
            if r.size:
                if method == 0:
                    return r[0] + np.sum(r[1:] / np.log2(np.arange(2, r.size + 1)))
                elif method == 1:
                    return np.sum(r / np.log2(np.arange(2, r.size + 2)))
                else:
                    raise ValueError('method must be 0 or 1.')
            return 0.


        def ndcg_at_k(r, k, method=0):
            dcg_max = dcg_at_k(sorted(r, reverse=True), k, method)
            if not dcg_max:
                return 0.
            return dcg_at_k(r, k, method) / dcg_max
```

In [28]:
```
### Example Usage
# Generate binary relevance array
r = np.zeros(len(kpreds))
for i, p in enumerate(kpreds):
    if p in val_set:
        r[i] = 1

ndcg_at_k(r, len(r))
```

Out[28]: 0.0

0.3 Baseline Model Performance

In [57]:
```
rps = []
ndcgs = []
for pid in co_mat.index:
    ps = kpredict(col_filter, pid, val) # predictions
    vs = val[val.Playlistid == pid].Track_uri # ground truth
    rps.append(r_precision(ps, vs))

    r = np.zeros(len(ps))
    for i, p in enumerate(ps):
        if np.any(vs.isin([p])):
            r[i] = 1
    ndcgs.append(ndcg_at_k(r, len(r)))
```

In [58]:
```
avg_rp = np.mean(rps)
avg_ndcg = np.mean(ndcgs)
print('Avg. R-Precision: ', avg_rp)
print('Avg. NDCG: ', avg_ndcg)
print('Total Sum: ', np.mean([avg_rp, avg_ndcg]))
```

```
Avg. R-Precision:  0.07702539127539126
Avg. NDCG:  0.08034624710411524
Total Sum:  0.07868581918975326
```