Data Mining and Machine Learning - Exercise 5

Johan Rodhe

October 26, 2016

1

a) Clustering generated dataset

The generating of the data for this exercise is done in the same way as in previous exercises so I will not cover that more here.

- The first part of the algorithm is to initialize the prototypes. For this, two random points were sampled from the dataset. This can be seen on line 69-70 in Figure 1.
- The second part consists of assigning all the points in the dataset to clusters.
- E-step: Compute the responsibility matrix R as follows(Eq 9.2 in Bishop):

$$r_{nk} = \begin{cases} 1, & \text{if } k = argmin_j ||x_n - \mu_j||^2 \\ 0, & \text{otherwise} \end{cases} \tag{1}$$

- M-stem: Update the prototypes.
- Repeat the previous three steps Niter times.

```
67  def kMeansIter(X, K, Niter):
68      J = []
69      old_mu = random.sample(X, K)
70      mu = random.sample(X, K)
71      while(Niter >= 0):
72          old_mu = mu
73          clusters, R, j = cluster(X, mu, K)
74          J.append(j)
75          mu = update_prototypes(old_mu, clusters)
76          Niter -= 1
77      return mu, clusters, R, J
```

Figure 1: The initialisation can bee seen on line 69-70.

```
26  def cluster(X, mu, K):
27      clusters = {}
28      R = np.zeros(shape=(len(X), K))
29      j = 0
30      n = 0
31      for x in X:
32          best_mu = min([(i[0], la.norm(x - mu[i[0]])) for i in enumerate(mu)], key=lambda t:t[1])[0]
33          for k in range(K):
34              if best_mu == k:
35                  R[n][k] = 1
36              else:
37                  R[n][k] = 0
38          try:
39              clusters[best_mu].append(x)
40          except KeyError:
41              clusters[best_mu] = [x]
42          n += 1
43      for n in range(len(X)):
44          for i in range(K):
45              j += R[n][i] * la.norm(X[n] - mu[i])
46      return clusters, R, j
```

Figure 2: Function implemented in python featuring the assignment of points to cluster step as well as calculating the responsibility matrix R.

The calculation of the responsibility matrix with equation (1) can be seen on line 33-37.

The next step is to update the prototypes which is done by calculating the mean of each cluster and setting the new prototype to the mean of the corresponding cluster.

```
48  def update_prototypes(mu, clusters):
49      new_mu = []
50      keys = sorted(clusters.keys())
51      for k in keys:
52          new_mu.append(np.mean(clusters[k], axis=0))
53      return new_mu
```

Figure 3: The clusters are stored in a dictionary where each points has a corresponds to 0 or 1 depending on which cluster it belongs to.

The dataset that was generated with 75 points from cluster 1 and 150 from cluster 2 can be seen in Figure 4.

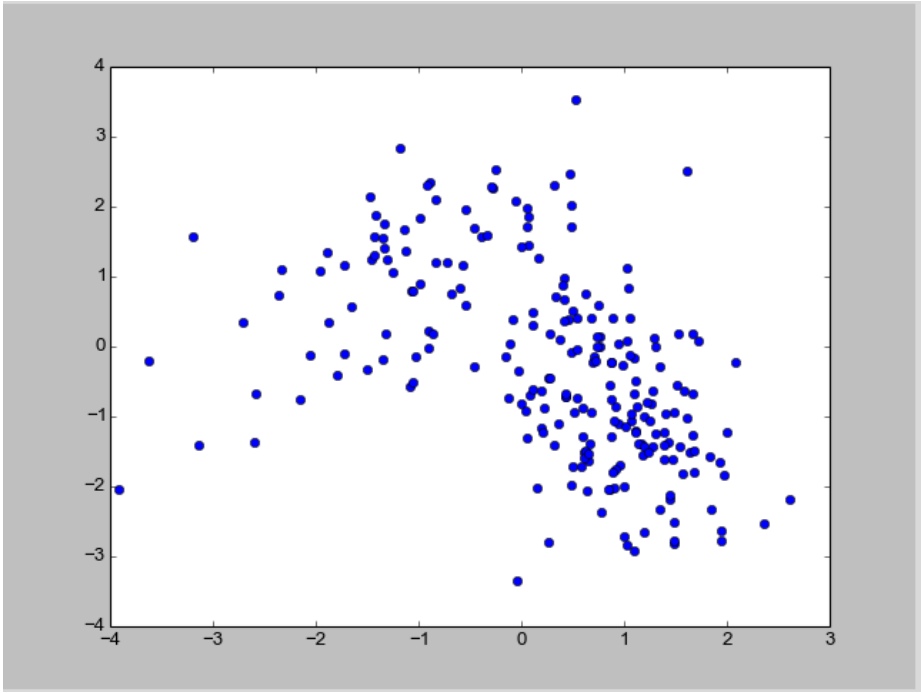

Figure 4: The generated dataset.

With K = 2 the result shown in Figure 5 was gained from running the kMeans program on the dataset.

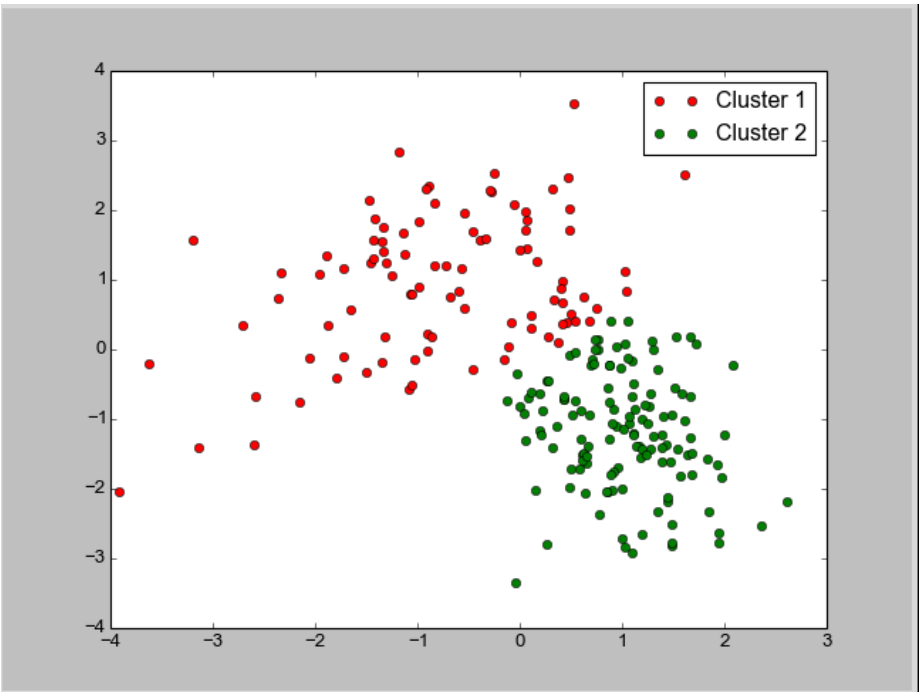

Figure 5: The plot shows the result of running the kMeans algorithm on the generated dataset.

b) Computing J

The J is computed as follows(Eq. 9.5 in Bishop):

$$J = \sum_{n=1}^{N} \sum_{k=1}^{K} r_{nk} ||x_n - \mu_k||^2 \tag{2}$$

This can be seen in the cluster function on lines 43-45.

```
43      for n in range(len(X)):
44          for i in range(K):
45              j += R[n][i] * la.norm(X[n] - mu[i])
```

Figure 6: Using Eq. (2) to compute J for each iteration.

By computing J for each iteration a sense of how much the distortion reduces for each iteration is gained.

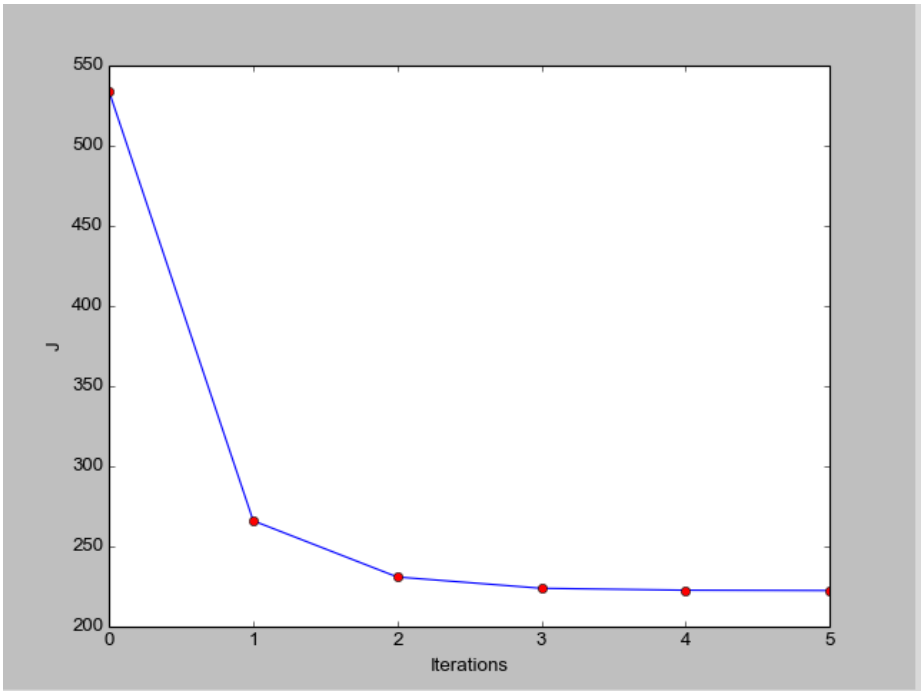

Figure 7: A plot of the distortion measure J with 5 iterations.

It can be seen in Figure 7 that after 3 or 4 iterations is enough. From iteration 4 to 5 it barely changes at all.

Instead of setting the number of iterations beforehand it would be more appropriate to set a converging criterion. Instead of running the steps a certain number of times we say that we run it until J has reached convergence.

```
def converged(mu, oldmu):
    return (set([tuple(a) for a in mu]) == set([tuple(a) for a in oldmu]))
```

Figure 8: Implementation of the convergence criterion. It is as simple as checking if the old prototype equals the new one. If it is then J has converged.

```python
def kMeansConv(X, K):
    J = []
    old_mu = random.sample(X, K)
    mu = random.sample(X, K)
    while not converged(mu, old_mu):
        old_mu = mu
        clusters, R, j = cluster(X, mu, K)
        J.append(j)
        mu = update_prototypes(old_mu, clusters)

    return mu, clusters, R, J
```

Figure 9: The kMeans program is altered slightly to use the convergence stopping criterion.

By plotting the J function as earlier we can see the difference.

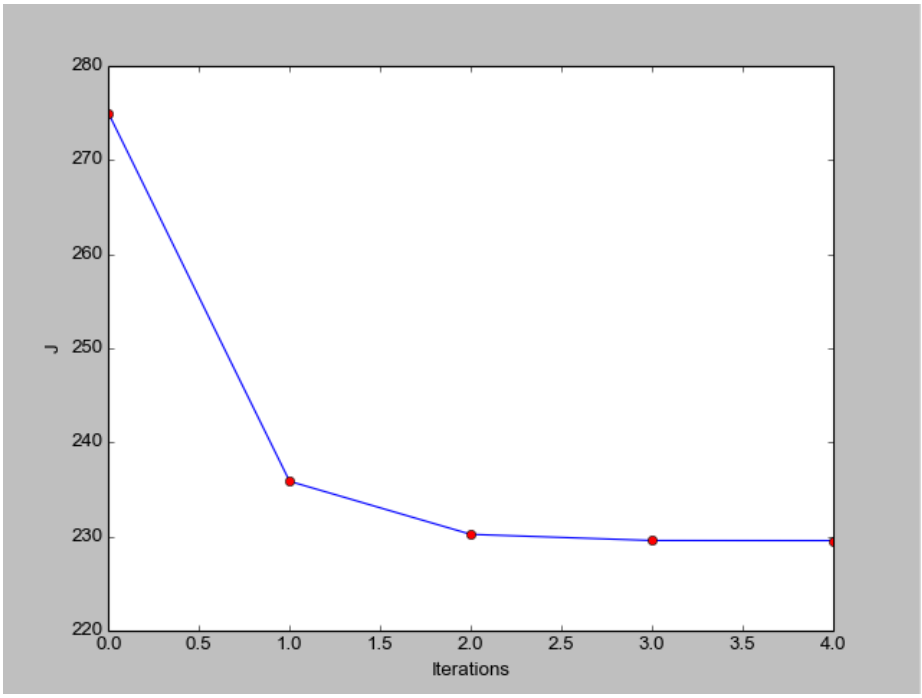

Figure 10: The program now stops after 4 iterations. Although this depends on the initialisation of the prototypes in the start. It does not always stop after 4 iterations.

In the remaining exercises the convergence stopping criterion was used instead of Niter.

c) Ratio in each cluster

By adding all the points in each cluster and then dividing by the total number of generated points the following rates were given:

- Cluster 1: 0.3822224. $225 \cdot 0.3822224 = 86$. If we compare this to the fact that we know there is 75 points generated from cluster 1 we can see how well the classifier did.
- Cluster 2: 0.6177776. $225 \cdot 0.6177776 = 138$

d) Ratio for different values of K

Table 1 displays the rations in each cluster for different values of K.

K	Cluster 1	Cluster 2	Cluster 3	Cluster 4	Cluster 5
3	0.1288889	0.2666666	0.6044444	0	0
4	0.1911112	0.2844444	0.3244442	0.2000001	0
5	0.1688889	0.3288889	0.2355555	0.1555556	0.1111111

Table 1: Ratio in each cluster for varying number of clusters. K refers to the number of clusters.

Since the dataset is generated from two different clusters I'm unsure of what to make of this result. Perhaps it is possible to say that cluster 1+2 = Cluster 1 and cluster 3+4+5 = Cluster 2 or something like this.

e) Applying to Iris dataset

The Iris dataset has three classes. So it makes sense to pick three as the value for K.

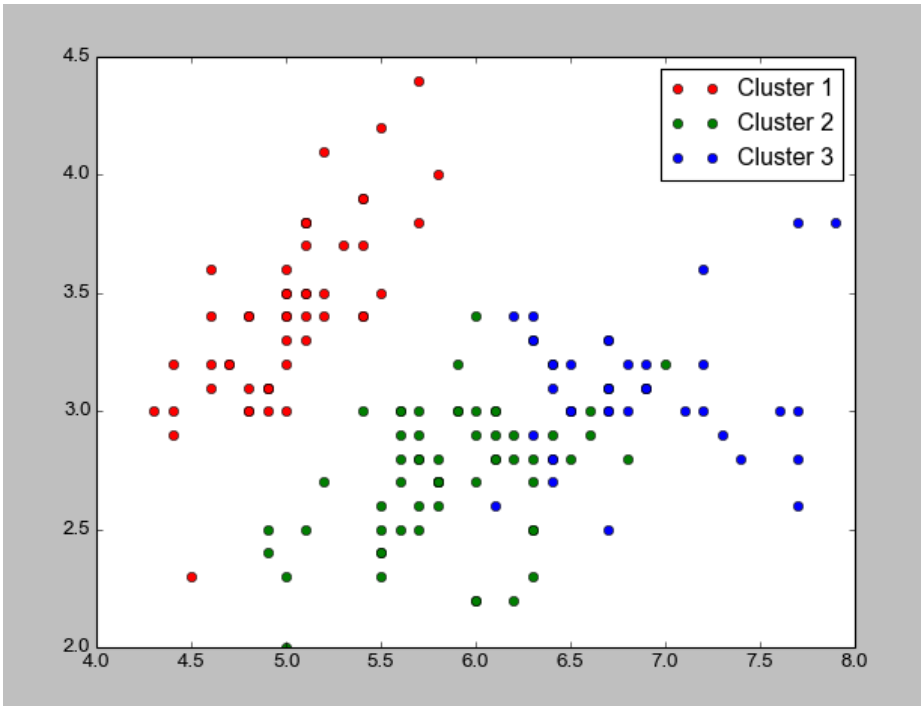

Figure 11: The kMeans algorithm applied to the Iris dataset. K = 3. Cluster 1 = Iris-setosa, cluster 2 = Iris-versicolor and cluster 3 = Iris-virginica.

Predicted / *Actual*	Iris-setosa	Iris-versicolor	Iris-virginica	Misclassification rate
Iris-setosa	50	0	0	0
Iris-versicolor	0	46	4	$\frac{4}{50}$
Iris-virginica	0	11	39	$\frac{11}{50}$

Table 2: Confusion matrix for the classifications.

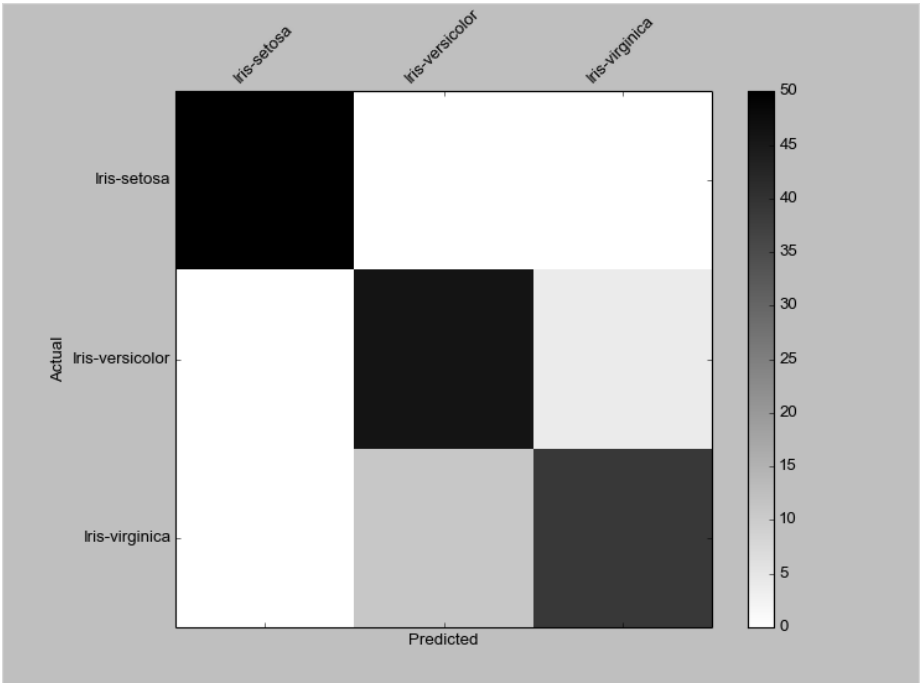

Figure 12: Same confusion matrix as Table 2 but presented differently.

Considering that the kMeans algorithm uses unsupervised learning I would say that it does very well on the Iris dataset. It has some problems distinguishing between Iris-versicolor and Iris-virginica since these two classes are harder to define as two separate clusters. Iris-setosa is effectively classified.

2

MovieGEN: A Movie Recommendation System

Authors: Eyrun A. Eyjolfsdottir, Gaurangi Tilak, Nan Li

Link:<http://www.cs.ucsb.edu/~nanli/projects/CS265-MovieGEN.pdf>

For consumers, there has probably always been some difficulty in choosing the right product. Consumers today in a world where the internet is a huge marketplace with so many possibilities the choice can seem impossible. From looking at a restaurant to choosing good investment options, consumers are dealing with too much information.

To help people deal with all the information some companies try to develop recommendation systems. These systems can guide a consumer through all this information. Research in this area has been going on for decades but it is still very relevant because of more and more applicable applications. Example of systems include recommendation systems for books at Amazon.com or Libra, for movies at MovieLens.org etc.

In this paper, the authors are using Support Vector Machines based learning techniques to predict genres and years that a user prefers based on that users

personal information.

The model takes an input from the user, quantifies it into a vector and then feeds it to the machine learning algorithm. If their model is then well trained it can predict an output vector which then can be presented as the users movie preference.

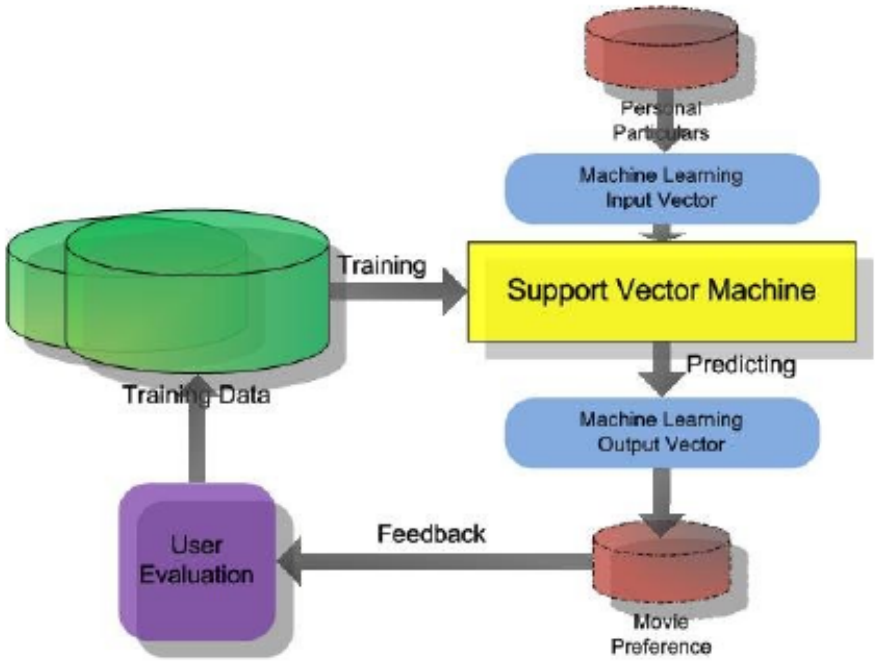

Figure 13: Figure presented in the study describing the model they use.

The Support Vector Machine(SVM) takes in a users personal particulars and outputs movie preferences.

The data has to be processed to fit the model. All personal particulars are mapped to predefined integers. For example Male = 1, Female = 2 etc.

Same is done with the movie preferences. For example Action = 1, Drama = 2 and so on..

In the system the study develops SVM is utilized to conduct correlation analysis between the users personal particulars and movie preferences. This is achieved by SVM regression.

The recommendation system then takes help of other machine learning tools to recommend a certain movie as can be seen in Figure 2.

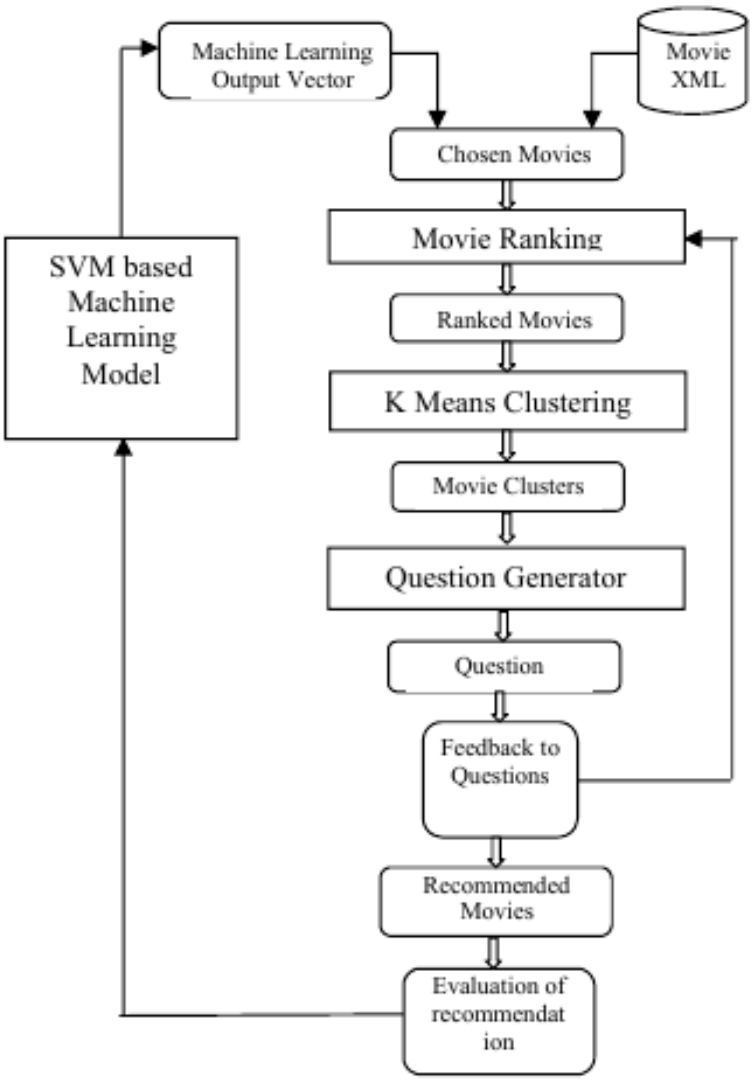

Figure 14: The figure shows how the recommendation system works in full and how the SVM model is related to it.

Conclusions: Since a recommendation can not be considered right or wrong in an easy way it is hard to measure the accuracy of the model. The authors mentions that the feedback they have got from users have been positive. Another thing that is mentioned is that using a larger dataset would provide more meaningful results. They would also like to try different machine learning algorithms to compare the results.