Adaptive Virtual Humorist using Online Learning

Vivek Kantilal Maganlal

Instituto Superior Técnico, INESC-ID
Av. Professor Cavaco Silva, 2744-016 Porto Salvo, Portugal
vivek.maganlal@ist.utl.pt

Abstract. The growing of the number of recommender systems and lack of their usage on the context of humor has driven us on the development of this work. In this thesis we applied online learning techniques to recommender systems in the context of joke telling and it resulted on an improvement of the feedback of the user on an early stage, compared to alternative techniques.

Key words: Prediction Problem, Expert, Recommender Systems, Online Learning, Collaborative Filtering

1. Introduction

In this thesis we are interested in maintaining a humorous interaction between a human user and an artificial agent. This interaction is such that the artificial agent must adapt himself to the human’s sensibility. This problem will be analyzed as a joke recommendation scenario, where the interactions between the user and the agent are incidental, and are not part of a long-term interaction. This adaptation requires the agent to have an online adaptation to the user’s immediate feedback provided when is proposed jokes by the system.

On this work we show that, with casual interactions as considered on this thesis, the sequence of recommendations used by typical collaborative filtering algorithms can have significant impact on the enjoyment of the user on the early stages of interaction. This is illustrated with a well-known online joke recommender system which uses collaborative filtering. Our results show that, rather using regret-based algorithms, we may improve the user’s general experience significantly.

Additionally, we present results that illustrate several interesting phenomena observed in the joke recommendation scenario that opens doors for future research: (i) in the presence of clustered data, recommender systems which are based on clustered data may perform worse than a simple global recommender; (ii) in the presence of differentiated user populations, adversarial online adaptation algorithms (i.e. such algorithms that do not assume statistical properties behind the user preferences) shown to be more robust than other methods.

2. Related Work
Our work focuses on recommending jokes, thus there are two main works that are similar to ours. The Robot Theater is a framework that capacitates a robot to have social capabilities to tell humorous stories. This system uses red and green colored cards to perceive the feedback from the user, which is gathered with an incorporated camera on the robot that counts how many green or red cards are displayed by the users on the audience, and with this feedback the robot recalculates the weights of its items and proceeds to tell another joke. The attributes for the items on the database to tell are classified using attributes, and those are taken into account when determining the next story to tell. [24] The items are classified with such attributes as item’s topic, length, interactivity, movement-level of the robot while telling the joke, appropriateness and hilarity.

![Figure 1. Framework of the Robot Theater.](image)

In this case, our solution may bring better results, since the most relevant features are extracted with base on the data, thus no pre-made classification is needed. The Jester online joke teller [5] is a joke recommender system that is based on the Eigentaste algorithm. It begins by polling the user for the eight jokes of the gauge set, and then proceeds to project the user on to the eigenplane using the eigenvectors calculated by the PCA, and attributes the user with the corresponding expert, which will recommend the user remaining jokes in the order of the best to the worst average joke rating for the users of that cluster. This algorithm provides constant time recommendation, but we need to study whether this tradeoff between online adaptation and constant time recommendation is or not beneficial.

3. Concepts

For a thorough understanding of the problem we are dealing with, its main concepts will be briefly presented in this section.

a. Exploration versus Exploitation

The exploration versus exploitation problem consists on deciding whether it would be more beneficial to explore the world looking for new information on our problem, or it would be the best option to use, or exploit, the information we already have. This duality of *modus operandi* is a well present issue on RS [1].
For a better understanding of this paradigm, let’s consider a user that interacts with a RS for the first time. As it is a new user, the system doesn’t really know which type of items to recommend to him, so it starts an exploration phase by showing the user some items and evaluating his feedback. After sufficient initial items have been tested, the RS may finally start the exploitation phase, which is when the user is presented items that the RS suspects with a certain confidence he will enjoy.

b. Multi-armed bandit paradigm

In many different times of real-life situations, we are presented decisions that will yield in a reward. These decisions should be wisely taken, so that the maximum reward is obtained. Inspired by this problem, comes the definition of the Multi-Armed Bandit problem (MAB).

Imagine we have a gambler, who bets on $N$ different slot machines, each giving a prize according to a certain unknown distribution.

The gambler playing with those $N$ slot machines has the objective of maximizing its bets' income, that is, it has to choose which of the levers to pull at each time $t$ to get the maximum reward sum overall.

The performance of the algorithm may be measured by the difference between the actual rating that the user gave to an item (or reward) and the expected rating that the algorithm has predicted for that item, or, by other words, the regret.

As our problem of recommending items to the user belongs to the class of opaque MAB problem, and as this paradigm seems more suited to real life situations, we’ll only explore this one.

If we have $K$-armed bandit, we have a set of distributions $\beta = \{R_1, ..., R_k\}$, being each of these distributions associated with a reward given when a specific lever is pulled at each time $t$. An algorithm, or gambler, will observe the rewards received and accordingly choose which lever to pull next, being his main objective to maximize the reward income [2].

This is the most suited model to our problem, where we have a set of experts, each of those can be corresponded to a slot machine, and the gambler, or algorithm, has to choose expert to choose for the next recommendation to the user, and the algorithm has the objective of maximizing the reward given by the user.

4. Employed Techniques

In this work, we will be testing three main techniques for the recommendation of jokes. This techniques model the users in different ways and choose a different options on the exploration and exploitation problem. As an algorithm will always have the exploration phase present throughout the recommendation session (Exp4), another will follow a greedy approach (UCB1), meaning that will only mostly rely on the exploitation of the information.

a. Exp4

The Exp4 is a non-deterministic algorithm that helps on solving the MAB problem and it always considers an exploration hypothesis throughout the recommendation session. It starts by giving the same weight to all the experts. Afterwards, each expert is asked for their advice vector, i.e. the list of items and respective probability of the user enjoying them. After all the experts are probed, the probability for each joke is calculated depending on the expert's prediction probability and respective voting weight. In this point, we have the probability for the selection of each joke, and so, using a random function, a joke is selected and presented to the user.

When the user gives his feedback, the weights of the experts are changed according to the confidence the expert recommended that item, and then the process is restarted.
Exp4 Algorithm

Initialization: Every experts starts with weight one.

Loop:
1. Get advice vectors $\xi_1(t), ..., \xi_N(t)$.
2. Set $W_t = \sum_{i=1}^{N} w_i(t)$ and for $j = 1, ..., K$ set
   \[ p_j(t) = (1 - \gamma) \sum_{i=1}^{N} \frac{w_i(t)\xi_j(t)}{W_t} + \frac{\gamma}{K} \]
3. Draw action $i_t$ randomly according to the probabilities $p_1(t), ..., p_K(t)$.
4. Receive reward $x_{i_t}(t) \in [0, 1]$.
5. For $j = 1, ..., K$ set
   \[ \hat{x}_j(t) = \begin{cases} \frac{x_j(t)}{p_j} & \text{if } j = i_t, \\ 0 & \text{otherwise}. \end{cases} \]
6. For $i = 1, ..., N$ set
   \[ y_i(t) = \xi_N(t). \hat{x}(t), \]
   \[ w_i(t + 1) = w_i(t) \exp\left(\frac{\gamma y_i(t)}{K}\right) \]

Table 1. The Exp4 algorithm.

b. UCB1

The UCB1 is a deterministic algorithm which chooses the expert that yielded the best reward until each time $t$. It is a greedy approach to the problem, since there is low margin left for exploration. The UCB1 will choose the expert with the best average reward to the moment. Initially the algorithm polls each expert once, tests the corresponding reward and attributes the average feedback of the expert to its weight. After all the experts are polled, the empirically best expert is always chosen, following a policy that has proven bounds. [3]

UCB1 Algorithm

Initialization: Test each expert once.

Loop:
- Play the expert $j$ which maximizes $\hat{x}_j + \sqrt{\frac{2 \ln n}{n_j}}$, where $\hat{x}_j$ is the average reward obtained from expert $j$, $n_j$ is the number of times expert $j$ has been played so far, and $n$ is the overall number of plays done so far.

Table 2. The UCB1 algorithm. [adapted from 6]

c. Eigentaste
This algorithm is important because it guarantees a constant online time recommending. On a general matter, the algorithm works as follows: every user that interacts with the system first has to rate an initial set of eight jokes, being these the jokes the classifier of the person into a cluster on the eigenplane. This eigenplane is built with the two most relevant eigenvectors that we obtained when applying the PCA [4] on the gauge set of the already gathered data. After the person is attributed a cluster, he will be recommended jokes from the corresponding expert, and that expert will recommend the jokes based on the best average ratings the users from that cluster gave to the jokes. For a more detailed description of the algorithm, see [5].

d. Proposed Modification: Content Aware Random

The existing non-deterministic algorithms, especially those we are using, don’t usually take into their advantage the use of all the information on the environment. So our proposal stands considering one more factor that will help on dictating the probability of a certain item being chosen by the random function. This factor takes into account the average point, or mean point (MP), of the users on the eigenplane, the density of each cluster (\( \frac{\# \text{ of users}}{\text{cluster area}} \)) and the distance of each cluster to the MP. This factor may computed as follows:

<table>
<thead>
<tr>
<th>Content Aware Random weight computation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Define ( g ) as ( 0 &lt; g &lt; 1 )</td>
</tr>
<tr>
<td>Define ( MP ) as the average position of the users</td>
</tr>
<tr>
<td>Define ( MAXDIST ) as the maximum distance between any cluster and the ( MP )</td>
</tr>
<tr>
<td>Define ( density ) as the density of each cluster</td>
</tr>
<tr>
<td>For each expert ( e_i \in E )</td>
</tr>
<tr>
<td>1. ( distanceWeight_i = MAXDIST - distance(\text{centerOfMass}(e_i), MP) + g )</td>
</tr>
<tr>
<td>2. Normalize ( distanceWeight_i )</td>
</tr>
<tr>
<td>3. ( weight_i = \frac{density_i \cdot distanceWeight_i}{\text{totalDensity}} )</td>
</tr>
<tr>
<td>4. Normalize ( weight_i )</td>
</tr>
</tbody>
</table>

Table 3. Computation of the Content Aware Random weight.

e. Other techniques

For us to compare the results of the previously described techniques, other simpler algorithms were also implemented, such as the random joke recommender, which recommends a joke randomly from the set of 100, the random expert recommender, which randomly chooses an expert in each interaction and presents the joke to the user. We also implemented a recommender which chooses an expert that will recommend the jokes through the entire session, and the mean item rating technique, which recommends the items based on the global average ratings of all users on that item.

5. Experimental Setup and Evaluation Methodology

For the experiment, we will setup the base data that will be common to all algorithms. This data is calculated in the same way as the Jester application does it. First, we will use that Jester raw dataset, available on its webpage\(^1\), and apply the PCA to the gauge set items. After the two principal

\(^1\) http://eigentaste.berkeley.edu/jester-data/
components are found, the users are projected on to the eigenplane using those eigenvectors, or principal components. On this eigenplane, we will do several clustering with different number of clusters, and obtain the corresponding performance for each of the algorithms. The number of clusters will vary as 5, 10, 16 and 28. We are only interested in knowing what is the raw rating the user gives to the sequence of jokes he is told. In this way, we will measure what truly counts from the user perspective.

\[ AUF = \frac{\sum_{i=1}^{\sigma} u_t(s_i(t))}{|C| \times \sigma} \]  

(16)

Simply, this metric will let us know the average feedback we received from the user (AUF) after \( \sigma \) jokes were told, thus we may more accurately analyze the amusement of the user. Regarding this metric, what it says to us is, if we consider the first \( \sigma \) jokes told by the algorithm, this is it's average user feedback, and as higher the value is, the better performing the algorithm has.

Each algorithm was run 10 times, and on each run 1000 random users were selected. This method of testing was chosen due to the existence of several thousands of users to test, and the lack of testing time.

6. Results

By running this experiment, we obtained the results presented on the Table 4. Using this new metric, we have a knowledge on which algorithms indeed provide the best experience to the user. Based on these results, we can conclude that increase on the number of clusters doesn’t bring much information for us to achieve better results with the user, but instead we find some loss of performance. It's interesting to notice how the Eigentaste's performance approximates to the random joke teller. This might be telling us that, although the Eigentaste might have a low NMAE [5], it’s static nature doesn’t seem suited to joke recommending. It is also interesting to notice that the UCB1 has a better performance than the Exp4 algorithm, meaning that the tested users follow a more statistical behavior, and the fact that the exploration is always present on the Exp4 doesn’t quite help on having a better performance.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>5 clusters</th>
<th>10 clusters</th>
<th>16 clusters</th>
<th>28 clusters</th>
<th>Algorithm Average</th>
</tr>
</thead>
<tbody>
<tr>
<td>Best AUF</td>
<td>3.302376465</td>
<td>3.302376465</td>
<td>3.302376465</td>
<td>3.302376465</td>
<td>3.302376465</td>
</tr>
<tr>
<td>MIR</td>
<td>1.725640842</td>
<td>1.725640842</td>
<td>1.651544176</td>
<td>1.651544176</td>
<td>1.66960775</td>
</tr>
<tr>
<td>Best Expert</td>
<td>1.7240088</td>
<td>1.651494176</td>
<td>1.651494176</td>
<td>1.651354645</td>
<td>1.66960775</td>
</tr>
<tr>
<td>Random Expert</td>
<td>1.538257645</td>
<td>1.557016866</td>
<td>1.470551547</td>
<td>1.487809792</td>
<td>1.51340996</td>
</tr>
<tr>
<td>UCB1</td>
<td>1.54654186</td>
<td>1.495127</td>
<td>1.451633652</td>
<td>1.4439958</td>
<td>1.48432458</td>
</tr>
<tr>
<td>Exp4</td>
<td>1.34601629</td>
<td>1.253766</td>
<td>1.136859912</td>
<td>1.26681511</td>
<td>1.25087593</td>
</tr>
<tr>
<td>Worst IE</td>
<td>1.196780895</td>
<td>1.137534482</td>
<td>0.854987978</td>
<td>0.854987978</td>
<td>1.01107283</td>
</tr>
<tr>
<td>Eigentaste</td>
<td>0.843980608</td>
<td>0.873582173</td>
<td>0.83846816</td>
<td>0.479359334</td>
<td>0.75884223</td>
</tr>
<tr>
<td>Random Joke</td>
<td>0.777277073</td>
<td>0.777277073</td>
<td>0.777277073</td>
<td>0.777277073</td>
<td></td>
</tr>
<tr>
<td>Cluster Average</td>
<td>1.5555662</td>
<td>1.530424</td>
<td>1.467696</td>
<td>1.443296</td>
<td>-</td>
</tr>
</tbody>
</table>

Table 4. Results on the performance of the algorithms with the different setups.

From the results obtained with this work, which are summarized on the Figure 1, we can conclude that the clustering with few clusters generally performs better than those who have lots of clusters. Although the best technique in this case is to choose the best independent expert as responsible for recommending items, we verify that the greedy approach to model the users seems to adapt the best to the variation of clusters, rather than the Exp4 or Eigentaste algorithms.
Considering the Content Aware Random proposal, it was tested on the Exp4 and Random Expert algorithms, which are the ones that are non-deterministic and for which we can use this technique. The results are shown on the Figure 3 and 4, which show the AUF metric applied from the joke 1 to 100. As we can see, on general, our proposal improves the AUF on almost every algorithm testing, except the case where there are 28 clusters on the clustering model. The results of the AUF are shown on the Table 5. If we choose the right number of clusters, we experienced gains on the AUF that went up to 18% with the Exp4 algorithm.
Figure 4. Performance results of the Exp4 recommender with the CAR.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>5 clusters</th>
<th>10 clusters</th>
<th>16 clusters</th>
<th>28 clusters</th>
<th>Algorithm Average</th>
<th>Δ</th>
</tr>
</thead>
<tbody>
<tr>
<td>Exp4</td>
<td>1.34601629</td>
<td>1.253765503</td>
<td>1.136859912</td>
<td>1.266861511</td>
<td>1.250875804 +1.24%</td>
<td></td>
</tr>
<tr>
<td>Exp4 CAR</td>
<td>1.390463746</td>
<td>1.251098176</td>
<td>1.172415199</td>
<td>1.251385792</td>
<td>1.26634072825</td>
<td></td>
</tr>
<tr>
<td>Random E.</td>
<td>1.538257645</td>
<td>1.557016866</td>
<td>1.470551547</td>
<td>1.487809792</td>
<td>1.5134089625 +0.98%</td>
<td></td>
</tr>
<tr>
<td>Random E. CAR</td>
<td>1.6565575</td>
<td>1.634135</td>
<td>1.522743494</td>
<td>1.299525712</td>
<td>1.5282404265</td>
<td></td>
</tr>
<tr>
<td>Cluster Average</td>
<td>1.4828238</td>
<td>1.424004</td>
<td>1.325643</td>
<td>1.326396</td>
<td>-0.12%</td>
<td></td>
</tr>
<tr>
<td>Δ</td>
<td>+5.5%</td>
<td>+2.37%</td>
<td>+3.37%</td>
<td>-0.12%</td>
<td>-0.12%</td>
<td></td>
</tr>
</tbody>
</table>

Table 5. Performance of the algorithms when CAR is applied.

It seemed natural to analyze whether there is a performance tradeoff between the cluster's density and its AUF. Having obtained the results presented on Figure 25 and 26, we can say that there is an obvious tradeoff, being this value of optimal density of the cluster between 15 and 250 users.

7. User Testing

A prototype was also built to test the studied algorithms with real users, using an interface that only permitted discrete feedback, which only let the user say if he liked didn’t like the joke. This prototype was with this type of feedback was also built because there exist many interfaces that only permit the user to give discrete feedback, thus we need to understand which type of algorithm
is more suited for this cases. Such interfaces can be, as an example, facial recognition systems, which detect if a certain person is smiling, meaning that he enjoyed the joke. As the period for user testing was short, we could only get 29 users, which were randomly attributed by the application one of the four algorithms we're testing, and yielding the following results.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th># users</th>
<th>AUF</th>
</tr>
</thead>
<tbody>
<tr>
<td>Exp4</td>
<td>6</td>
<td>0.424347</td>
</tr>
<tr>
<td>Eigentaste</td>
<td>7</td>
<td>0.361494</td>
</tr>
<tr>
<td>Random E</td>
<td>8</td>
<td>0.34174</td>
</tr>
<tr>
<td>UCB1</td>
<td>8</td>
<td>0.209663</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td>29</td>
<td><strong>-</strong></td>
</tr>
</tbody>
</table>

Table 6. Results of testing the algorithms with users.

Although we have greatly less user information than on the testing with the Jester's data, we can conclude some things. With this results, we can see that the initial jokes of the Eigentaste are indeed a bottleneck on the overall algorithm performance, but after the user is attributed an expert, the performance becomes good. This difference on the performance from our previous testing might have various variables attached to it. First, as we are gathering discrete information, we are reducing the quantity of information given to us by the user, and this seems information more suited to the algorithms we are testing.

Also, as we use the $k$-Nearest Neighbor, in this case 250 NN, altogether with the Eigentaste, it may be resulting on the approximation of the users into were the most users are, yielding the attribution of clusters more populated.

Contrary to the results we got from the continuous rating scale, the Exp4 performed better than the UCB1, which has the poorest scoring algorithm.

Again, these conclusions must be understand carefully, as the supporting data is scarce, and the universe of the population for that tested the algorithms was limited.

8. Conclusion and Future Work
This paper presented an analysis and comparison of different algorithms that have different base of functioning, adapted to the context of joke recommendation.

As with the results obtained, we can see that:

- Mean Item Rating has the best results overall, meaning that the best recommendation of jokes are done when the users are considered of enjoying the jokes globally in the same way.
- The jokes present on the gauge set, even if they are the ones that bring more information about the user, which are used by the Eigentaste contribute to the general poor performance of the algorithm, having almost the same performance as a random joke teller.
- The increase on the number of clusters doesn’t seem to be helping us on providing more accurate recommendations to the user, but instead we note a degradation on the performance of the algorithms.
- The clusters have a optimal number of users versus the cluster's performance such that tradeoff is optimal, meaning that if we have that number of users in our cluster, we can expect a good performance from it.
- The right choosing of the number of clusters and the usage of the Content Aware Random may increase the overall performance of the algorithm, regardless of the non-deterministic algorithm chosen.
- Exp4 algorithm has given results on being the most stable and adaptable algorithm to the situation where the feedback is continuous or discrete.
- The results presented here indicate us that we should only tell an initial sub-set of the jokes, meaning that the algorithms are, on average, telling the most suited jokes first, leaving the worst ones for the end. To analyze the improvement of this procedure, we need the user to tell us what was his enjoyment level after using our system.

When we use recommendation systems expert based, we may hardly achieve the best performance, since each expert has a predefined sequence of jokes, meaning that the recommendation process be limited from the start.

A natural step for this thesis seems to be the integration of this system with an automatic information gathering system, such as the a facial recognition system, and then verify the performance of the algorithms. In this way, it may increase the user enjoyment of the system, since no unnatural interaction has to be made.

For the complete results of this work, see the original dissertation.

References