Building Recommendation System for `CatMatch`

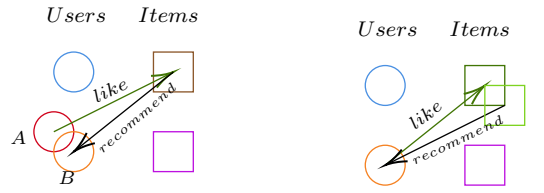
Marijan SORIĆ
Cogito Norwegian University of Science and Technology for `CatMatch`
`marijaso@stud.ntnu.no`

I. Introduction

`CatMatch` is an application build by 5 members of Cogito NTNU. The aim of this project was to create a *Tinder for cats*. The application recommends cats based on user's preferences: users can swipe right (like) or left (dislike) on cats. We built from scratch a small recommendation system based on similarity between content. We thought about how to improve our recommendation system with adding parameters to optimize.

II. Overview

There are three main types of recommendation system. The first one is collaborative filtering, which is either user-based or item-based. On the latter, the system recommends items to a user B based on preferences from users similar to B (user A's preferences in 1a). On the former, the system recommends items similar to previous likes from the user.



(a) User-based (○) (b) Item-based (□)

Fig. 1: Collaborative filtering

The second one is content-based recommendation system. They are based on the similarity between items to recommend them to users. Since our application `CatMatch` will get more items *cats* than users, it is more pertinent to look at item similarity. We choose to implement this type. Finally, the third one is hybrid filtering, the more advanced, which utilizes both the two first types. The similarity between users or items can be computed by different ways: Euclidean distance, cosinus similarity or dot-product. They all catch different information. The dot-product similarity takes in account the length of embeddings, which is not the case for cosinus similarity.

III. Our algorithm

Assume that there is no *cold start problem* which means that the application counts few users having already *rated*[1] some cats. Let denote $R \in \mathcal{M}_{n_u,n_c}([0,1] \cup \{\texttt{NaN}\})$ and $E \in \mathcal{M}_{n_c,d}(\mathbb{R})$ respectively the rating and the embedding matrix[2]. R contains `NaN` values, those we want to predict. The cats are represented as embeddings which catch important features of the cat as we think of it (physical characteristics): this was the purpose of the other part of the `CatMatch` team with a convolutional neural network.

The goal is, for each user u, to recommend new cats c_n. To do so, we want to predict the rating $R(u, c_n)$ for every new cats c_n, in order to know which cats the user might like, and then recommend the top k results. To predict $R(u, c_n)$, the algorithm computes similarity S between cats rated by user $c \in C_r(u)$ and the others $c_n \in C_r(u)^c$. Then, for each *new* cat, we predict the rating $R(u, c_n)$ by weighting the rates given by the user u with the similarity between the new cat c_n and those rated $c \in C_r(u)$. Let's break down each important steps.

A. Similarity matrix S

Let $x, y \in \mathbb{R}^d$ be the embedding of the i^{th} and j^{th} cats. We compute cosinus similarity between cats. Let $S \triangleq (s(E_i, E_j))_{i,j}) \in \mathcal{M}_{n_c}(\mathbb{R})$. With $s : \mathbb{R}^d \times \mathbb{R}^d \mapsto [-1, 1]$ defined as:

$$S_{i,j} = s(x, y) = \frac{x \cdot y}{\|x\|\|y\|} = \frac{\sum_{i=1}^{d} x_i y_i}{\sum_{i=1}^{d} x_i^2 \sum_{i=1}^{d} y_i^2}$$

When each embedding of each cat is normalized (noted E_N), then $S = E_N E_N^T$. This matrix allows us to know the similarity between a new cat and the previous ones seen by the user. These coefficients will weight the rating.

B. Compute $\hat{R}$

Once we have a similarity matrix, we can predict the rating with the following formulas. We assume that $C_r(u) \neq \emptyset$.

$$\hat{R}(u, c_n) = \frac{1}{\sum_{c \in C_r(u)} |S(c_n, c)|} \sum_{c \in C_r(u)} S(c_n, c) R(u, c)$$

That way, $\hat{R}(u, c_n) \in [0, 1]$. The algorithm following synthesizes each step[3]. Actually, it returns the k cats with the highest predicted rate.

[1] A rate is here 1 *like* or 0 *dislike* (swipe right or left).

[2] Number of users: n_u. Number of cats: n_c. Number of dimensions used to represent a cat as a vector: d (emebedding).

[3] With $\alpha_{c,u} = \frac{S(c_n,c)}{\sum_{C_r(u)} |S(c_n,c)|}$.

Algorithm 1 $R(u)$ algorithm

function $R(u)$
  $C_r(u) \leftarrow \{c \mid R(u,c) \neq \texttt{NaN}\}$
  for c_n **in** $C_r(u)^c$ **do**
    $\hat{r}(u,c_n) \leftarrow \sum_{C_r(u)} \alpha_{c,u} R(u,c)$
  return $\arg\max_{c_n}\{\hat{r}(u,c_n)\}$

We finally managed to predict ratings, by looking at similarity between previous and new cats. But that way, the algorithm doesn't change over time, it learns nothing.

IV. Theoretical improvement

From now on, we would like to look at the performance of our algorithm. One of the main ways of improvement might be considering modification over E. As a matter of fact, in content-based recommendation system, cat's embeddings should catch most relevant information about a cat's rating (rather than cat's physical attributes). That is why we added a weight matrix $W \in \mathcal{D}_d(\mathbb{R})$ for each feature (columns) in E, so d parameters. Initially, $W_0 = I_d$. We also could add deeper transformation with $W \in \mathcal{M}_d(\mathbb{R})$ and its d^2 parameters. We can update embeddings, the new one are denoted $\tilde{E} = EW$.

$$\hat{R}_W(u,c_n) = \frac{1}{\sum_{c\in C_r}|S(c_n,c)|}\sum_{c\in C_r} E_{c_n} W W^T E_c^T R(u,c)$$

Once the rating matrix is filled, we recommend a fixed number of new cats k to the user. Using the user feedback $R(u,c_n) \in \{0,1\}$, we can use the loss function: square error.

$$\mathcal{L}(R(u,c_n),\hat{R}_W(u,c_n)) = \frac{1}{2}(R(u,c_n) - \hat{R}_W(u,c_n))^2$$

The goal is now to minimize the risk function $\mathcal{R}$ by choosing the right parameters W. The risk is defined as the expected values, over the data set $\mathcal{X}$ of known data point (R where there is no `NaN`), of the loss. This quantity is approximated with the empirical risk.

$$\mathcal{R}(W) = \mathbb{E}_{\mathcal{X}}\left[\mathcal{L}(R,\hat{R}_W)\right] \approx \frac{1}{|\mathcal{X}|}\sum_{(u,c_n)\in\mathcal{X}} \mathcal{L}(R,\hat{R}_W)$$

To optimize the parameters w_i, we use the gradient descent on the risk function $\mathcal{R}$ with the learning rate η. At each step, the parameters are updated toward the direction of the gradient:

$$w_j \leftarrow w_j - \eta\frac{\partial\mathcal{R}}{\partial w_j}$$

We can compute, for $W = \text{diag}(w_1,...,w_d) \in \mathcal{M}_d(\mathbb{R})$.

With $\alpha_W := \sum_{c\in C_r(u)}\sum_{i=1}^{d}|E_{c_n,i}E_{c,i}|w_i^2$.

$$\begin{aligned}
\frac{\partial\mathcal{R}}{\partial w_j} &= \frac{1}{|\mathcal{X}|}\sum_{(u,c_n)\in\mathcal{X}}\frac{\partial\mathcal{L}}{\partial w_j}\\
\frac{\partial\mathcal{L}}{\partial w_j} &= (\hat{R}-R)\frac{\partial\hat{R}}{\partial w_j}\\
\frac{\partial\hat{R}}{\partial w_j} &= \frac{\partial}{\partial w_j}\left(\frac{1}{\alpha_W}\sum_{c\in C_r(u)}\sum_{i=1}^{d}E_{c_n,i}w_i^2E_{c,i}R(u,c)\right)\\
&= \frac{w_j}{\alpha_W^2}\Bigg(\sum_c E_{c_n,j}E_{c,j}R(u,c)\sum_c\sum_i|E_{c_n,i}E_{c,i}|w_i^2 -\\
&\qquad \sum_c\sum_i E_{c_n,i}w_i^2E_{c,i}R(u,c)\sum_c|E_{c_n,j}E_{c,j}|\Bigg)
\end{aligned}$$

V. Conclusion

Finally, it was a very nice experience with a good team on a funny topic. Unfortunately, we only have tested the recommendation system without parameters and weren't able to validate the way we processed it. There were no time for evaluating our model with split train-test set, even though it is very important. Furthermore, the cat's embeddings weren't ready at the end of the first semester of work.