Movie Recommendations Using Low-dimensional Codes and User Specified Feature Relevance

Christopher Curro, David Katz, Harrizon Zhao
The Cooper Union
Electrical Engineering

***Abstract*—We present a movie recommendation system that finds a weighted set of nearest neighbors to an arbitrary desired movie based on user specified interests in a latent space learned by an autoencoder. We learn a low-dimensional representation to make recommendations in from a much larger feature space consisting of approximately one thousand tags and their relevancies to about ten thousand movies.**

I. Introduction

Movie recommendation is complex task that generally involves high dimensional feature spaces; a common approach today is the collaborative filtering approach [1]–[3]. The collaborative filtering approach is powerful because it successfully reduces a complex high-dimensional feature space into a rich low-dimensional latent space. We aim to present an alternative approach to generating a rich low-dimensional latent space capable of producing high quality movie recommendations based on a nearest neighbor approach where the queried sample is determined by user input. We work with the dataset created by Vig et al. [4]. This dataset provides approximately ten thousand movies with relevance weightings for approximately one thousand tags. We labeled each of our latent features in a human understandable manner. We also provide users with the ability to adjust both the target value and the relevance of each of these features at each query.

A. Autoencoders

An autoencoder is a particular type of neural network. A neural network is a computational graph where nodes can be defined in the form:

$$\mathbf{y} = f\left(W\mathbf{x} + \mathbf{b}\right) \tag{1}$$

Where $\mathbf{x}$ is an input vector, W and b are parameters that define a linear relationship, and $f(\cdot)$ is an activation function. The activation function is generally a non-linear function so each node in the graph can perform a non-linear mapping from $\mathbf{x}$ to $\mathbf{y}$.

Nodes can be connected successively to one another in a feed-forward fashion to create increasingly complex representations of the input data. Nodes in these networks are generally referred to as layers, and can be represented diagrammatically as in Figure 1.

Figure 1 in particular shows an autoencoder. Unlike an ordinary neural network an autoencoder contains a bottleneck in the middle. It is this bottleneck that creates the rich dimensionality reduction power of an autoencoder. This bottleneck essentially splits the network into two stages: the encoder stage and the decoder stage. We interpret the autoencoder as having two stages because we optimize the learned parameters of the autoencoder to minimize the reconstruction error at the output of the decoder. By following this objective and applying other constraints, such as a sparsity constraints, rich coding schemes can be achieved at the bottleneck.

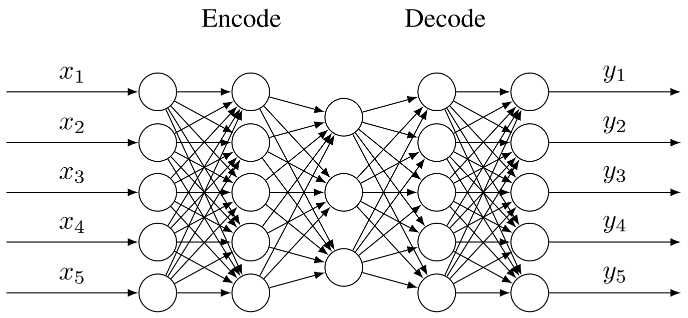

Fig. 1: A simple autoencoder with a two layer encoder/decoder pair. This autoencoder would code a five dimensional space into a three dimensional space. Each neuron represents a weighted sum passed through an activation function.

B. Nearest Neighbors Recommendations

After selecting a latent feature space in order to find recommendations for a query, a recommendation engine must perform a nearest neighbors query. The movies with the closest distance to the user's input or latent features are selected as recommendations [5].

For multi-dimensional spaces, the only method which is guaranteed to return exact nearest neighbors is a linear search [6]. This becomes unfeasible with larger datasets with many users. For this reason, recommendation systems tend to use approximate nearest neighbors methods, as there is little consequence for recommending a set of movies which are close to the best recommendations rather than the exact best. To do so, an index is constructed on the dataset once. Each query applies this index will only search a small subset of the entire dataset but will still return close to the best results.

Typically for relatively low dimensional spaces the Euclidean distance metric is used. However, to handle the constraint of user specified dimension relevance we used a modified Euclidean distance metric shown in Equation 2. The x and y represent two movies, while v represents the user specified weight on each dimension. Since v is normalized to have a sum of 1, when v is uniform this metric is equivalent to Euclidean distance.

$$distance(x, y, v) = \sqrt{\sum_{i=1}^{D} ((x_i - y_i) \times v_i \times D)^2} \quad (2)$$

II. System Description

A. Autoencoder

The autoencoder we created to perform the dimensionality reduction encodes a length 1128 vector down to length 10 code. This is performed with a single hidden-layer encoder and a single-hidden layer decoder. All of the neurons are hyperbolic tangent neurons. These were selected over sigmoidal neurons because their derivatives are symmetric around zero and therefore they learn their parameters more quickly [7]. The encoding and decoding stages shared the same number of neurons in each layer. Both stages were constructed from two layers consisting of 1128 neurons and one layer consisting of 10 neurons. (This accounting includes the unweighted input neurons for both the encoder and decoder.)

We implemented and trained the present autoencoder with the Torch 7 [8] framework; the computation was accelerated through the use of an Nvidia Tesla K40. We selected the mean squared reconstruction error as our criterion and used Nesterov accelerated gradient descent [9] to optimize it.

B. Feature Tagging

We first selected the top k movies which maximally activated each feature. We then aimed to find tags prevalent in these movies in order to help us come up with meaningful class descriptions. Popular and high quality tend to have more tags. As such, in our latent space they tend to have higher than average values across all features.

To analyze which tags were most related to each feature, we used a weighted tag prevalence score based on the top normalized and averaged tags in the top k. We also penalized this metric against high variance tags.

$$TagPrevalence = w1 * normAvg + w2 * Avg - w3 * var \quad (3)$$

Increasing w1 puts more emphasis on normalized tag values. This pulls out tags which appear more prevalent in this class relative to other classes. Tags like “poverty” and “star wars” will tend to appear for some classes which contain some related movies.

Increasing w2 puts more emphasis on raw tag values. Tags which are popular within the class but not necessarily more popular than the entire population begin to appear. Examples: “original”and “oscar” tend to appear.

Increasing w3 penalize tags with large variance within the class. For example if 1/3 of the movies in the class are related to poverty, then the “poverty” tag on these movies will tend to be high, but close to zero for the rest of the class. With the normalized average, the class will appear highly related to poverty. However by penalizing tags with high variance, we can put weight against tags which are highly prevalent on only a subset of the top k, as we want tags related to all of the movies.

After manually tuning and examining the tags which were produced, we settled on k=200, w1=3, w2=6, and w3=2. From manually looking at the top 20 movies, and prevalent tags for the top k movies, we came up with concise, human understandable class descriptions for each class. The chosen tags are displayed on our web interface.

C. Interface

The webserver consists of a Node.js backend and a frontend built using Angular.js. The webserver backend proxies the C# backend, which performs the KNN calculations, and relays the result to the end user. The interface has two sections: tuning features and recommended movies. The tuning features section consists of 10 features which correspond to the ones obtained from the autoencoder. Slider bars for each of the features allow the user to tune the value and relevancy for each of the features. A search bar is available for the user to search for a given movie and preload the searched movie’s feature values. Buttons allow the user to set the relevancy of all the features to either 0, 50, or 100. Clicking the ”Recommend Movies” button sends a request to the Node.js server which proxies the C# backend and loads the top 15 recommended movies. The recommended movies are then loaded in the recommended movies section. A recommended movie can then be clicked to go to the IMDB page corresponding to the clicked movie.

D. Recommendations

Since this dataset is relatively small and we do not expect a large number of users, a linear search for the top k movies is feasible. However, to scale to a larger userbase and a larger dataset, and approximate nearest neighbors index should be used. While typical approximate nearest neighbors indexes are not adaptable to query specified dimension relevance. [10] describes an efficient index for handling this type of nearest neighbor query.

III. Results

While it is difficult to quantitatively measure the quality of the recommendations, aside from explicit user feedback on them, we are able to measure the performance of the autoencoder network. In Figure 2 we see the usual type of training curve for a neural network with the error decreasing and then reaching a steady bottom. Ideally this curve would get significantly closer to zero, but considering the number of dimensions in the reconstructed space compared to the reduced space, the performance of 80 MSE is impressive. The apparent quality of recommendations confirms the good performance of the autoencoder.

We keep logs of all of the queries performed. These logs of real user data could serve as a valuable dataset for maximizing the efficiency of an ANN index.

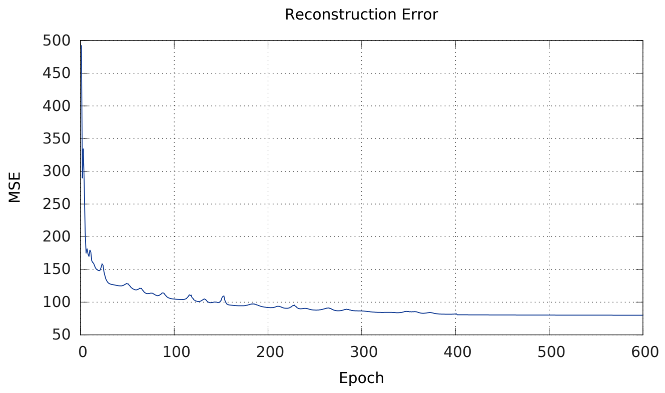

Fig. 2: Reconstruction error versus number of training epochs of the autoencoder. The reconstruction error is measured as the MSE between the original high-dimensional vector and the the approximate high-dimensional vector decoded by the decoder from the low-dimensional code. Each epoch consists of ten thousand example exposures to the network. This network converges at an MSE of about 80.

IV. Conclusion

We have presented a movie recommendation system that operates in a low-dimensional space produced by a feature space reduction of 99% using an autoencoder. We have hypothesized what each of the latent features in the low-dimensional code represents. In the future we would like to crowd source an understanding of the latent variables using Amazon Mechanical Turk as Zhou et al. have done [11].

References

[1] M. D. Ekstrand, J. T. Riedl, and J. A. Konstan, "Collaborative filtering recommender systems," *Foundations and Trends in Human-Computer Interaction*, vol. 4, no. 2, pp. 81–173, 2011.

[2] N. Good, J. B. Schafer, J. A. Konstan, A. Borchers, B. Sarwar, J. Herlocker, and J. Riedl, "Combining collaborative filtering with personal agents for better recommendations," in *AAAI/IAAI*, 1999, pp. 439–446.

[3] Y. Koren, R. Bell, and C. Volinsky, "Matrix factorization techniques for recommender systems," *Computer*, no. 8, pp. 30–37, 2009.

[4] J. Vig, S. Sen, and J. Riedl, "The tag genome: Encoding community knowledge to support novel interaction," *ACM Transactions on Interactive Intelligent Systems (TiiS)*, vol. 2, 2012. [Online]. Available: http://www.grouplens.org/system/files/tag_genome.pdf

[5] B. Sarwar, G. Karypis, J. Konstan, and J. Riedl, "Item-based collaborative filtering recommendation algorithms," in *Proceedings of the 10th international conference on World Wide Web*. ACM, 2001, pp. 285–295.

[6] M. Muja and D. G. Lowe, "Scalable nearest neighbor algorithms for high dimensional data," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 36, 2014.

[7] P. Sibi, S. A. Jones, and P. Siddarth, "Analysis of different activation functions using back propagation neural networks," *Journal of Theoretical and Applied Information Technology*, vol. 47, no. 3, pp. 1264–1268, 2013.

[8] R. Collobert, K. Kavukcuoglu, and C. Farabet, "Torch7: A matlab-like environment for machine learning," in *BigLearn, NIPS Workshop*, no. EPFL-CONF-192376, 2011.

[9] Y. Nesterov *et al.*, "Gradient methods for minimizing composite objective function," 2007.

[10] D. Katz and C. Sable, "An efficient index for approximate nearest neighbors with query specified dimension relevance weights," 2015.

[11] B. Zhou, A. Khosla, A. Lapedriza, A. Oliva, and A. Torralba, "Object detectors emerge in deep scene cnns," *arXiv preprint arXiv:1412.6856*, 2014.