Tensorflow.js를 활용한 상점 추천 학습

조재영O, 이상원*, Tai Myoung Chung*
Dept. of Software, Sungkyunkwan UniversityO
Dept. of Software, Sungkyunkwan University*
e-mail: cjy12182000@naver.comO, {swlee, tmchung}@skku.edu*

A shop recommendation learning with Tensorflow.js

Jaeyoung ChoO, Sangwon Lee*, Tai Myoung Chung*
Dept. of Software, Sungkyunkwan UniversityO
Dept. of Software, Sungkyunkwan University*

● 요 약 ●

Through this research, the rating data of shops were analyzed. The model was designed for discrete multiple classification as to the corresponding data, and the following experiments were initiated to observe the learned machine. By comparing each benchmarks in the experiments, which contains different setting variables for the machine model, the hit ratio was measured which indicates how much it is matched with the expected label. By analyzing those results from each benchmarks, the model was redesigned one time during the research and the effects of each setting variables on this machine were clarified. Furthermore, the research result left the future works, which are related with how the learning could be improved and what should be designed in the further research.

키워드: Supervised Learning, Tensor, Classification

I. Introduction

In recent years, the machine learning area has grown up significantly [1]. This year, Turing awards were awarded to AI researchers, who pioneered neural network in machine learning, which became fundamental parts in modern AI research projects. This paper addresses the basic parts and flows in machine learning to tackle a simple shop recommendation algorithm by 'supervised learning'. All the used code for this paper could be found in the Github page [5].

During the past decades, the interest on pets has increased. Many families started to adopt pets as members in the family and it was also not exceptional to my family. That is the background how the desire to make recommendation service for pets and pet owners came out. Pet owners would get recommendation from the app where they go with their pets. Since human has had some obvious mistakes in real learning, training machine with plenty of data and observing the result from the learning are also worthy simulation activity compared with human. That is how machine learning was adopted for this motivation.

To achieve this purpose, the idea was outlined to fill the gap between the technical side in tensorflow and the motivation for pets. 'Supervised Learning' was chosen to train the model because Supervised Learning is the fundamental part of machine learning. In addition, the logic to train was also simplified as choosing highest rating among rates for each shop. In this research, each shops represent a virtual pet-shop and giving rating activity for each shop was replaced by generating data in a programming way. Eventually, the model would be able to give recommendation for the pet owner in the application. For the further research, the recommendation algorithm could be advanced referring human choice mechanism.

II. General Features

There are three major parts in this application, data, model and visualization, which is supported by 'tfjs-vis' library, as seen in Figure 1. As it mentioned before, the data was generated in this application, and it would be updated from the real data in the further project. The model consists of 4 layers, including

input and output layers. The weight would be updated among the hidden states in each layer. The visualization part shows the progress in training and allows us to compare the performance of the model. Measuring how good the model is, is important to set a plan to move forward.

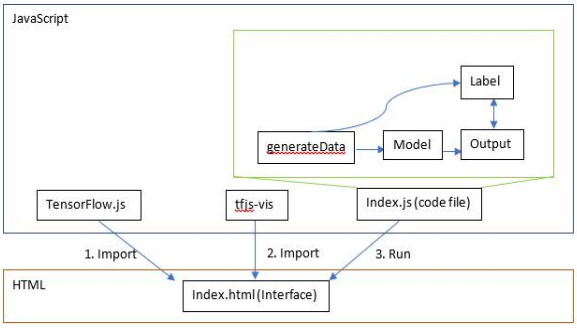

Fig. 1. Overview of the recommendation learning application

Data was generated randomly. One data consists of several sets of the element, and one element consists of the number of shops of randomly generated number and the index which contains highest number. For example, if the number of shops is 5, and the randomly generated numbers are [1.5, 2.8, 5.8, 3.7, 7.4], the matched element will be [1.5, 2.8, 5.8, 3.7, 7.4, 4]. The index was calculated, starting from 0.

Model has 3 layers in this application. The layer could be added if it is necessary, but I fixed the number as 3. First layer gets input data which consists of the given amounts of elements that each element consists of the given randomly generated numbers. And it passes next calculated values to the second layer after getting them through the 20 hidden units. The second layer receives those values from the first layer and gets them through 18 hidden units to the next layer. The third, last layer produces the last output data in the form of [batch size, and the number of shops].

Batch size is the number of elements, which are trained together [2]. For example, if the data has 500 elements in total, the training could be done in 500 iterations with 1 batch size. Or if the batch size is 10, 50 iterations are needed for one epoch time.

Epoch is the count of period which is when an entire data is moved forward and backward through the neural network only once [2]. For example, if the training set has 500 elements, epoch means that the period time which moves all elements forward and get one time backward through the neural network.

Iteration is the number of batches in each epoch [2]. For example, if batch size is 10 and 500 is the total number of elements, the number of batches is 50. The model should iterate 50 times for each training data in one epoch.

Neural Network is an interconnected assembly of simple processing elements, units or nodes, whose functionality is loosely based on the animal neuron. The processing ability of the network is stored in the interunit connection strengths, or weights, obtained by a process of adaptation to, or learning from, a set of training patterns [3]. In other words, in computer science, neural network usually contains several layers which also consist of several nodes, which called ‘units’ or ‘nodes’.

III. The Proposed Schemes

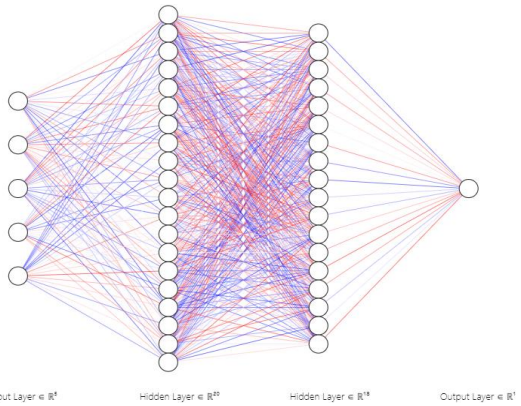

Fig. 2. first designed model with one output node, drawn by NN-SVG [6]

The first model was designed with one output node as seen in Figure 2. For example, if one element with 5 shops is [1.2,2.3,3.2,4.6,3.1,3], The label output was set as [3]. With 10 elements, the label output would be [3,1,0,2,4,3,1,2,4,2], which contains the index of the highest rating shop in each element. This model kept failing, because training with the discrete number in one output node was not appropriate for training this model. It turned out that the model is well-trainable with the contiguous values. So, the new model was designed.

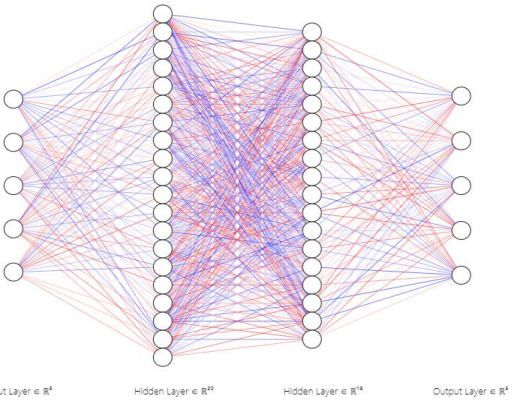

Fig. 3. Second model with the number of shops output nodes, drawn by NN-SVG [6]

To overcome the limit in first model, the second model used the number of shops for the number of output nodes as seen in Figure 3. The model would increase the number of output nodes as the number of shops increases. In the above example, instead producing [3] per the suggested element, [[0,0,0,1,0]] was set for the label output. Since [0-1] is in normalized range of values, this label output format was much better than the first one. By changing the label output format in the model, the model could be trained as expected.

IV. Implementation

```
async function run() {
    const numShops = 5;
    const numElements = 500;
    const range = 10;
    const data =
generateData(range,numShops,numElements);
    //Convert the data to a tensor form we can use for
training.
    const tensorData = convertToTensor(data,numShops);
    const {inputs, labels} = tensorData;
```

Fig. 4. beginning part of the code in run function which acts as main in this application

The application only contains two files in the project folder, 'index.html' and 'index.js'. The application starts from index.html and it runs the script in Javascript, which is written in 'index.js' file. In this 'index.html' file, it imports tensorflow.js library and tfjs-vis for using machine learning part in this application and visualizing the result from it.

In 'index.js' file, run function is called first. Figure 4 shows the first part of the run function. In the run, generateData() function receives the number of shops, the number of elements and range for generating input data and labels to train the model. After finishing generating the data, the data is returned as the array data as it was designed. To train the model, this array should be converted into the tensor which is a generalization of vectors and matrices to potentially higher dimensions [4]. tensorData contains 2 tensors which represent input and label and 4 tensors for each Max/Min in each input and label tensor.

After preparing the tensorData for training the model, the run function creates the model with the number of shops. Passing the number of shops to createModel() function is required because input Shape would be changed depending on the number of shops. After creating the model, tfvis library shows the information of the model whether it is trainable or not. The trainModel() function trains the model with inputs and labels tensors, and tests the model with other testing data which are generated for testing. The other testing data is generated in the testModel function with numShops and numDatas. Figure 5 shows these procedures after the tensorData was prepared.

```
const model = createModel(numShops);
//Print the status of models
tfvis.show.modelSummary({name: 'Model Summary'}, model);
await trainModel(model, inputs, labels);
//Test the model
testModel(model, data, tensorData,numShops, numDatas);
```

Fig. 5. last part of the run function in index.js

V. Experiments and Analysis

Table 1. The summary of result in each benchmark, B: Batch Size, Ep: Epochs, S: the number of Shops D: the number of Elements R: Range of each rating, H: Hit ratio (%)

Benchmark	B	Ep	S	E	R	H
1 (Primary)	10	200	5	500	10	91.4
2	10	200	3	500	10	98.6
3	100	200	5	500	10	85.2
4	10	200	5	200	10	87.5
5	10	100	5	500	10	91.2
6	10	200	5	500	100	91.0

Several experiments were performed in each benchmark settings as seen in Table 1. The benchmark 1 would be the primary one to be compared with others. Other benchmarks were set to have different values in each variable, such as Batch Size, Epochs, etc. Hit ratio was calculated as the matched number out of total tested number, such as 450 out of 500 per the same testing data.

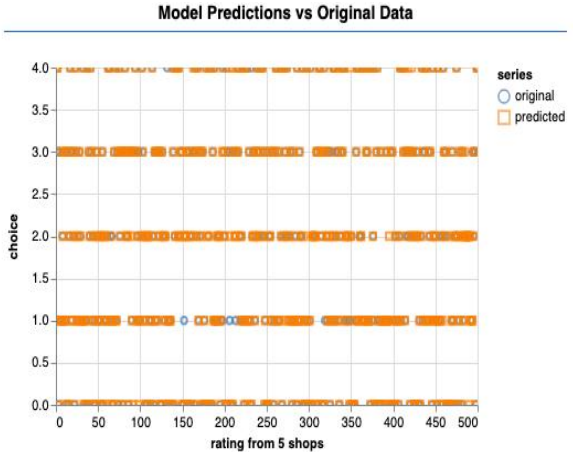

Fig. 6. Compare the expected result data and the predicted result data based on the benchmark 1

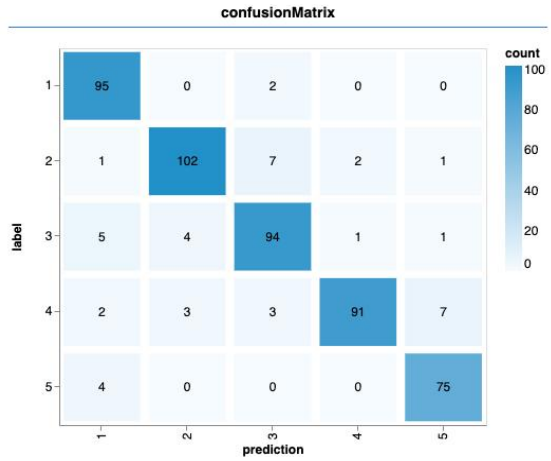

Fig. 7. Confusion Matrix to see the detailed information in the prediction based on the benchmark 1

Figure 6 and Figure 7 shows that how the result was in detail in the benchmark 1 setting. There was 91.4 hit ratio with the model, but it is still unnoticeable to find the pattern in unmatched data around the hit data in Figure 7.

Comparing other benchmarks, starting with benchmark 2, when the number of shops, classes internally, is reduced, the hit ratio is getting better. For the model, it is easy to classify the given data into less the number of classes, but when the number of classes increases, the model shows the difficulty to classify the data correctly.

With benchmark 3, the hit ratio is getting worse when the batch size increases. This is because of underfitting by the increase of batch size. In this experiment, the number of elements should also be increased if we expect to get similar performance with the benchmark 1. By this comparison, it shows how batch size affects the model.

Regarding benchmark 4, the hit ratio is getting low when the number of elements decreases. This is also because of the same reason with benchmark 3. Underfitting was caused by scarcity of training data.

When comparing with the benchmark 5, it was remarkable to observe that there was trivial change in hit ratio compared to benchmark 1. The underfitting was expected as similar as benchmark 3 and 4, but it turned out that 100 is enough as the number of epochs.

Lastly, the benchmark 6 showed that the range of rating did not affect much on the performance. Normalization in the convertToTensor function made the range be insignificant factor. Because the normalization, which made all numbers in data in [0,1], rarely change the distribution in the data, the hit ratio was not affected much by the range factor.

VI. Future Works and Conclusions

Comparing benchmark 1 and benchmark 2, there is a tendency that the performance is reduced when the number of classes increases. When the model was tested with 10 classes without changing any other settings, the hit ratio was reduced to 58.2%. To avoid underfitting, an increase of the number of elements was tried. As a result, the hit ratio was back up to 67.8%. But this model is still not good enough to be used as a reliable model with the increased number of classes. The other model or benchmark should be suggested to get better performance.

Through this research, the general and fundamental parts of supervised learning were addressed and learned. For the same logic, the machine could be trained better or not depending on how the model was designed for the given data. We could assume that the role of excellent Data scientist would be analyzing raw data well and transforming the data into well-trainable data for the model. By analyzing data and modifying the model, this research could be done as expected. For further research, the new benchmark and model should be addressed.

REFERENCES

[1] Tom Simonite, The godfathers of the ai boom win computing's highest honor, Wired, March.2019. https://www.wired.com/story/godfathers-ai-boom-win-computings-highest-honor/

[2] Sagar Sharma, Epoch vs Batch Size vs Iterations, Towards Data Science, Medium, Sep.2017. https://towardsdatascience.com/epoch-vs-iterations-vs-batch-size-4dfb9c7ce9c9

[3] Gurney, Kevin, An Introduction to Neural Networks, UCL Press, 1997. https://www.inf.ed.ac.uk/teaching/courses/nlu/assets/reading/Gurney_et_al.pdf

[4] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, et al. TensorFlow: Large-scale machine learning on heterogeneous distributed systems. Page 4, 2016. https://arxiv.org/abs/1603.04467

[5] Jaeyoung Cho, Graduate_project-v-000, GitHub, 2019. https://github.com/Joeycho/graduate_project-v-000

[6] LeNail, NN-SVG: Publication-Ready Neural Network Architecture Schematics. Journal of Open Source Software, 4(33), 747, 2019. https://doi.org/10.21105/j