CS 5785 Applied Machine Learning - Final Report
Build a Large-scale Image Search Engine

Shunzhe Yu
Master in Computer Science
Cornell Tech
New York, NY 10044
`sy679@cornell.edu`

Ran Ji
Master in Computer Science
Cornell Tech
New York, NY 10044
`rj369@cornell.edu`

Fei Li
Master in Computer Science
Cornell Tech
New York, NY 10044
`fl392@cornell.edu`

Abstract

In this paper, we develop the core algorithm of a large-scale image search engine with various machine learning models, which retrieve 20 most relevant images from an image dataset, given a natural language query. We investigate and extract different types of features from images, descriptions and tags in the given dataset. Based on our extracted features, we also build various machine learning and deep learning models such as, Partial Least Square Regression (PLSR), Multi-Layer Perceptron (MLP) and neural networks. We compare their performances and ensemble them by score weights, which finally achieves a score of 0.47029. In the end, we propose some future works that can be explored to improve our image search engine.

1 Introduction

1.1 Background and Related Work

Cross-modal matching has drawn increasing attention because of the widely-existed multimodal data. It involves the pattern matching among different modalities such as text, image, audio and video. And in this paper, we mainly focus on the common feature space between text and image in order to solve the problem of relevance measurement. Multiple different approaches have been proposed based on the neural language models (Kiros et al., 2014), log-bilinear models (Mnih & Hinton, 2007) and so on. In particular, our work focus on using Partial Least Square Regression (PLSR) and Multi-Layer Perceptron (MLP) to model cross-modal data.

Cross-modal matching has many practical uses in fields such as the advertisement, social media and so on. Consider the following tasks

- finding the correct background images with a short advertisement description for business
- finding the best matching emojis/GIFs that match with users tweet sentences in social media

Specifically, in this project, we are asked to build a reverse image search engine that searches for relevant images from existing image set given a natural language query.

1.2 This Task

This task can be considered as a regression problem. Its training vectors are feature vectors extracted from descriptions, and its target vectors are image feature vectors. We can also reverse the training and target vectors to get generate other models.

The overview of our regression models is as follows. First, we should train a regressor on the training set, which maps the training feature vector space onto the target feature vector space. Second, we use the trained regressor to predict target vectors from training vectors on the testing set. Third, for each predicted target vector on the testing dataset, we find the nearest 20 (ranked by cosine distance) target vectors from the training set in the corresponding feature vector space as the search results.

The main contributions of our team can be summarized as follows:

- One person primarily focused on training models using existing data set, includes mapping descriptions to the Bag of Words feature vectors, fine-tune using tags, and using data generated by other teammates to train better models.
- One person primarily focused on exploring new data source, such as using open source frameworks to generate image captions, using word2vector to generate description feature vectors and exploring the deep neural network.
- One person primarily focused on writing pipelines and exploring technology to better ensemble result, as well as training models and generating input data.

The rest of this paper is organized as follows. We introduce our extracted features in Section 2, and describe our machine learning and deep learning models in Section 3. Then, to improve the performance beyond single models, we perform model ensembles in Section 4. We compare the model performances in Section 5 and conclude our work in Section 6. We also propose some future works in Section 7.

2 Features

We are provided 10,000 sample images for training. For training images, each image has five sentences description and one text file contains the tags in the image. We have another 2000 images that are used for testing. Each testing image also contains a tag file. When the user types a natural language search query, we return 20 most relevant images from the test image set in the ranking of confidence. To simplify our work, the professor also provides us with the feature vectors for all the training and testing images.

We used various features in our models. Some features are given, some features are generated by ourselves using pre-processing scripts or open-source frameworks. The following are all the features in our models.

2.1 Image Feature Vectors

The image feature vectors are provided by professors. We have two versions: pool5 and fc1000 layer. Pool5 has more features so it will contain more information. We used fc1000 to test and submit result trained by pool5 to achieve better performance.

2.2 Description - Bag of Words (BOW) with/without PCA

The BOW is a feature vector that each cell is a weight for a unique word that appears in all the descriptions. To generate it, we first cleaned the text descriptions by removing signs and loaded all the English word into a HashMap. In the HashMap, the key is the word and the value is the number of times it occurs in all the descriptions. Then we removed all the auxiliary words using a list created manually by looking at the most frequent words in the train descriptions. We also tried several open-source libraries to only keep the nouns and verbs. After getting a meaningful word list, we sorted all the word by its frequency. Now we have a list of N unique words. For each image, we create a binary feature vector with size N, 0 in index i indicate the i-th word does not appear in that image description and 1 means it does. Instead of a binary array, we also tried to set the value to be the occurrence times in each image description.

In addition, we tried to filter some words using a frequency threshold when creating the BOW feature vectors to reduce dimensions. PCA is another method we used for this purpose. By reducing dimensions, we can save time and resource when training our models.

From the above methods, we generated different types of input and use them to generate different models and predictions. We compared their performance, and ensembled them to achieve higher accuracy.

2.3 Description - Word2vec

For the task of finding the correct embedding representation for words, we also explore the usage of word2vec model (Mikolov et al., 2014) proposed by Google. Specifically, we are using the word2vec model pre-trained on three billion Google News corpus, which encodes each English word into a 300-dimensional vector. That yield a fine property for word similarity comparison as the model encodes words semantic meaning and is able to work with the arithmetic analogies such as king - man + woman = queen. Having that in hand, we are able to construct the sentence embedding representation by word embeddings by weighting. There are two potential directions here: (a) by equally averaging and (b) weighted by TF-IDF weights. In this paper, we mainly focus on the former method that equally averages each word embedding to form a sentence vector.

2.4 Description - Google Universal Sentence Encoder Embedding

Another tool that can help us encode descriptions into embedding vectors is Google's Universal Sentence Encoder. The encoder encodes each English text into a 512-dimensional vector which can be used in various NLP tasks, according to its original paper. In the given dataset, there are five sentences for each description. By intuition, five sentences carry more information than a single sentence, so we decide to encode all five sentences into one vector for each description file. Since the encoding model is not linear, we cannot simply average the encoded vectors of all five description sentences. Thus, we use the encoder to encode the concatenation of five sentences into an embedding vector for each description file. We also tested the encoding of the longest sentence in each description file, but it performed worse than the concatenated one.

2.5 Tags - Bag of Words (BOW)

In the given dataset, there is a list of tags indicating objects appeared in each image. We count unique tags in all tag files, and it turns out that there are only 80 distinct tags in the whole dataset. So it is clear to use Bag-of-words (BoW) model to represent the list of tags for each image. In the BoW vector, 80 entries represent 80 different tags, and for each entry, 1 means the corresponding image has this tag, 0 otherwise.

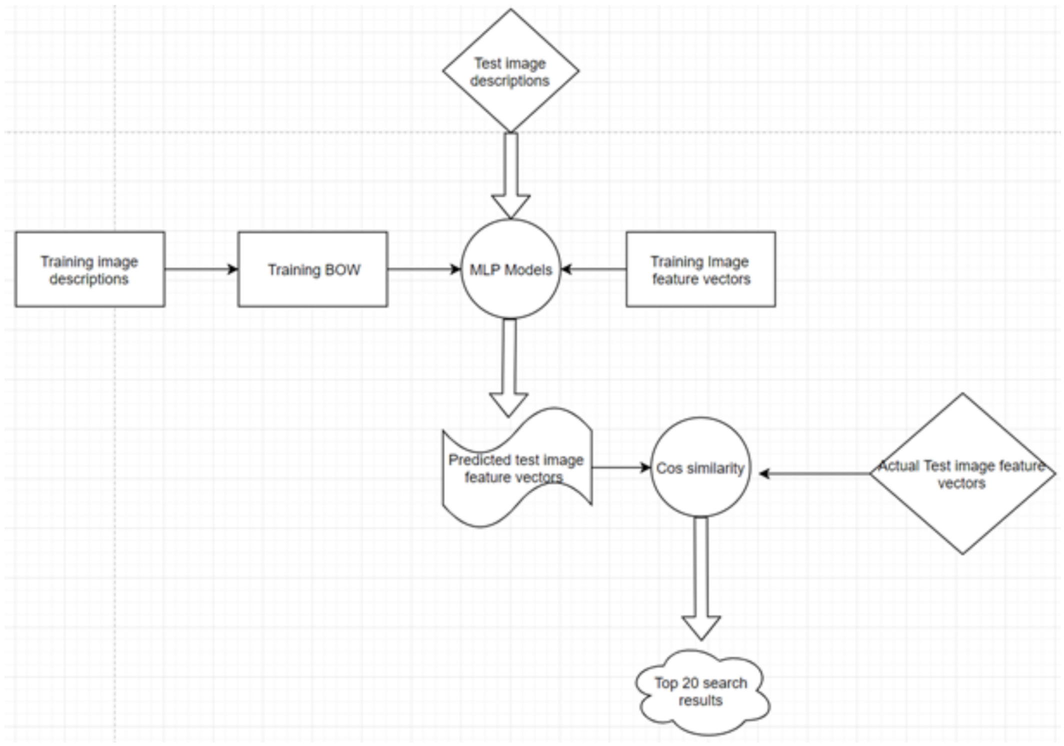
2.6 Image Captioning - LSTM

Compared with mapping the words or sentences into a feature space, we instead turn into another direction of using LSTM with a recurrent network to obtain image caption in words directly. The model usually served as the decoder network in a typical image caption model, and as we have image pooling features provided, we simply trained such decoder network individually and obtain image description in return. Such text output has some disadvantages compared with the above embedding representations as it presents already in the text as the final result and therefore loses the rich information from the hidden state. Further details and comparisons will be shown in the following experiments section to demonstrate that embedding features are more desired than descriptive features in this specific task.

3 Models

3.1 Description BOW - MLP Regression - Image Feature Vectors

From 2.2, we generate the description BOW feature vectors for each training image. Then we trained MLP model with training data to be BOW and target data to be pool5. After training the model, we input the test description BOW to the MLP and it will output the predicted image pool5 feature vector. For each predicted feature vector, we compare it with all test image feature vectors, calculate their cos similarity, and rank the top 20 results from high to low. The workflow is shown in the graph:


Test image descriptions
Training image descriptions
Training BOW
MLP Models
Training Image feature vectors
Predicted test image feature vectors
Cos similarity
Actual Test image feature vectors
Top 20 search results

The code is shown as follow:

```
from sklearn.neural_network import MLPRegressor
clf = MLPRegressor(hidden_layer_sizes=(2048,1024,512), max_iter=500, solver='sgd', verbose=True)
clf.fit(train_des, train_feature_vector)
predict_image_feature_list = clf.predict(test_des)
```

The parameters are selected by cross-validations and subject to computing resources. This method achieves a testing score of 0.21110.

Parameters: hidden_layer_sizes=(2048,1024,512), max_iter=500, solver='sgd'.

3.2 Description BOW - PLS Regression - Image Feature Vectors

This method is the same as 3.1 except we use PLS regression instead of MLP regression. This method achieves a testing score of 0.17098.

Parameters: n_components=200.

3.3 Description BOW - PLS Regression - Tag BOW

This model is similar to 3.1, but we use the description BOW feature vectors as inputs, and BOW from 2.4 as outputs, PLS as the regressor. We also apply PCA to the description BOW feature vectors to reduce their dimension to 2048, which accelerates the training speed. This model achieved a testing score of 0.26727.

Parameters: n_components=100.

3.4 Description Embedding - PLS Regression - Image Feature Vectors

This model is similar to 3.1, but we replace the MLP regressor by PLS regressor and use description vectors encoded by Google's Universal Sentence's Encoder instead of description BOW. Its testing score is 0.19557.

Parameters: n_components=256, max_iter=1000.

3.5 Captioning BOW - PLS Regression - Image Feature Vectors

This method is the same as 3.1. The difference is we used LSTM recurrent network to generate captions for each image. Then we use captions as input for BOW. In this method, we are not using the training descriptions provided by the professor. Its testing score is 0.31181.

Parameters: n_components=400.

3.6 Image Feature Vectors - PLS Regression - Description BOW

From 2.2, we get BOW feature vectors for each train images. Then we trained PLS regression model. The training vector is pool5 and target vector is BOW. This is the reverse direction of 3.2.

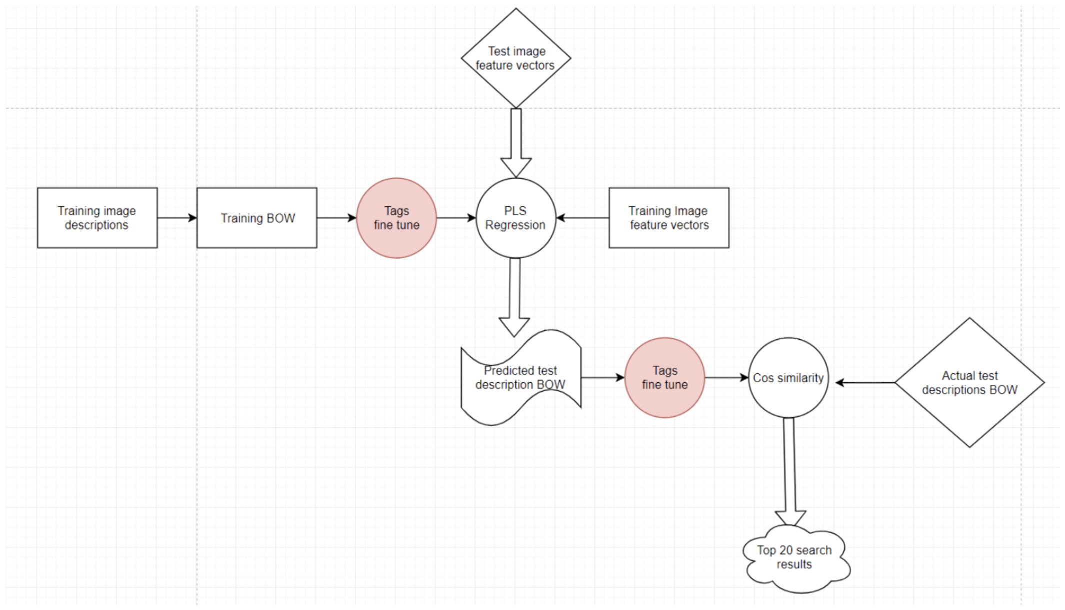
After training the model, we input the test image pool5 feature vector and predict the test image description BOW. We use cos similarity to compare it with the actual test image descriptions to get the top 20 results. This gives us 0.37724 in the score.

Tricks: Tag fine tune (boost result from 0.37724 to 0.40498)

For trained description BOW, we check if the image tags are in the BOW binary feature vector. If not, we add tags to it.

For predicted test image descriptions BOW, we also check its value for each tag. Since the predicted BOW comes from PLS regression, so each cell is usually a small float. For each tag word, we set the cell value to be min(original_cell_value, 1). By using the tag fine tune, our score boosted from 0.37724 to 0.40498. This is the best single model we built.

The workflow is shown in the graph:


Test image feature vectors
Training image descriptions
Training BOW
Tags fine tune
PLS Regression
Training Image feature vectors
Predicted test description BOW
Tags fine tune
Cos similarity
Actual test descriptions BOW
Top 20 search results

Parameters: n_components=600.

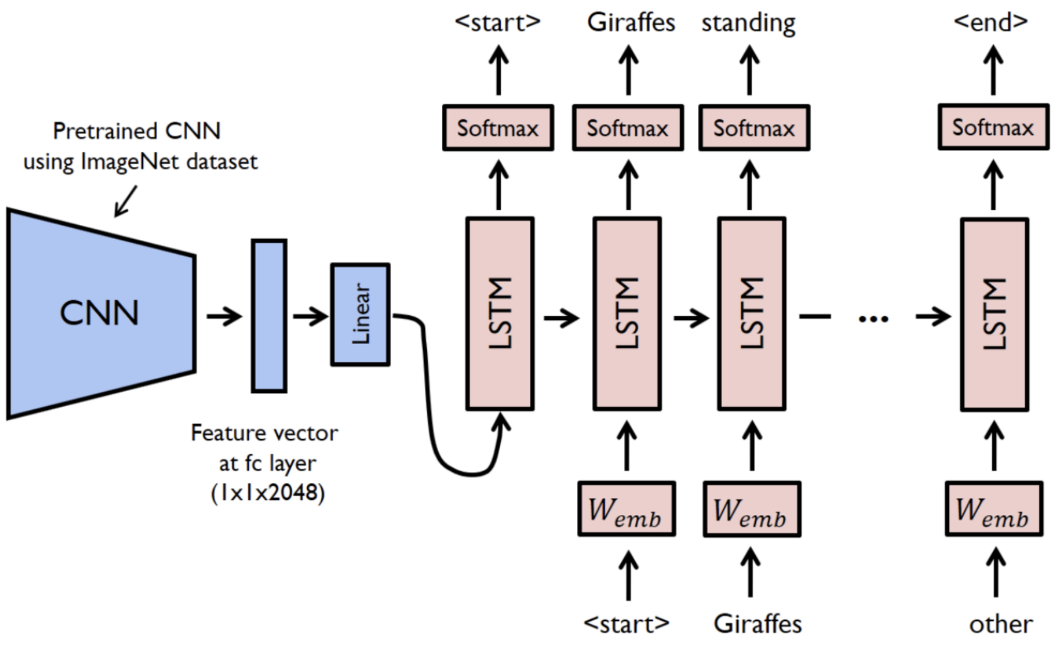
3.7 Image Feature Vectors - MLP Regression - Description BOW

This is same as above except we use MLP instead of PLS to predict result. Its testing score is 0.20922.

Parameters: hidden_layer_sizes=(2048,4096)

3.8 Image Captioning - Text Similarity

As for image captioning model, our initial idea is to leverage LSTM gates combined with a recurrent neural network to generate text output. And therefore, after we obtain the output we can compare the generated text with the target description using metrics such as BLEU scores, editing distance and so on. One disadvantage that has been mentioned above is that when using such approach, we actually lose more information than directly processing the raw image feature, as such approach undergoes two information transformation process as illustrated in the below graph. After comparing the generated text with the target descriptions, we obtain the final score of 0.13756 in Map score, which is not a desirable score, but is able to be ensembled later with other better models.



3.9 Deep Neural Network (Why it does not Work)

Another approach we employed in this project is the deep neural network, more specifically, a network of 12 layers, consisting hidden neurons layers with ReLU activation functions and dropout layers. We posit the image pooling feature vector as the network input and the bag of words binary representation as final network output and use binary cross-entropy as loss function in order to convert it into a multi-label problem. With 500 training epochs, we found out that the final result is not desirable and way under the expectation. We reflect the practice and think that such transformations cannot be easily converted into the multilabel question, which causes the dataset unbalanced with respect of each label. We finally discard the model and focus more on the regression approximation.

4 Ensembles

To achieve a better predictive performance than single models listed in the above section, we present the model ensemble techniques in this section. There are a large number of ensemble types, such as bagging, boosting, and stacking. But in this task, we find out a simple and flexible ensemble approach that can boost the predictive performance.

We first select a subset of models from the above section. In their prediction results, each row is a list of 20 ranked images for the corresponding testing example. For each row (testing example), we assign a score to the ith-ranked image using the formula of MAP@20: $score = \frac{20+1-i}{20}$. After accumulate scores from all selected models, we have a dictionary of image scores for each row (testing example). We use the top 20 ranked images in the dictionary as the ensembled prediction results for that row.

In the above approach, we treat different models with the same importance (called "averaged" method). We can also assign weights to different models according to their performances. One straightforward way is to use the testing scores of single models as their weights in the ensemble. For example, we selected model A, B, and C for ensemble, and their testing scores are 0.4, 0.3, and 0.2, respectively. Then when we accumulate score each image in each row, we calculate its total score by the formula $total\_score = score_A \times 0.4 + score_B \times 0.3 + score_C \times 0.2$. We find that this "score-weighted" method generally performs better than the "averaged" method.

We test the “averaged” and “score-weighted” methods on different subsets of models, and it shows that the predictive performance is generally better when we increase the quantity and diversity of the models in our selected subsets. Our ensembles finally achieve a score of 0.47029 as a combination of 7 models.

The testing scores of more model ensembles are listed below in the next section.

5 Results

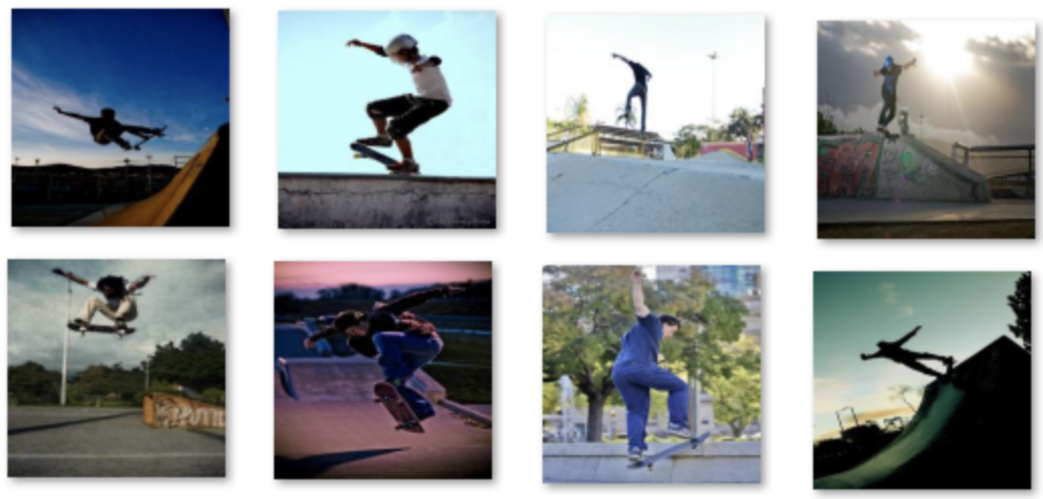
The “best” scores of single models and ensembles are listed below. The highest testing score among all models and ensembles is 0.47029 which is the prediction result of an ensemble of 7 models.

Model	Score
3.1. Description BOW - MLP Regression - Image Feature Vectors	0.21110
3.2. Description BOW - PLS Regression - Image Feature Vectors	0.17098
3.3. Description BOW - PLS Regression - Tag BOW	0.26727
3.4. Description Embedding - PLS Regression - Image Feature Vectors	0.19557
3.5. Captioning BOW - PLS Regression - Image Feature Vectors	0.31181
3.6. Image Feature Vectors - PLS Regression - Description BOW	0.40498
3.7. Image Feature Vectors - MLP Regression - Description BOW	0.20922
3.8. Image Captioning - Text Similarity	0.13756
Ensemble of 3.5 + 3.6	0.39651
Ensemble of 3.3 + 3.5 + 3.6	0.41507
Ensemble of 3.1 + 3.3 + 3.5 + 3.6	0.44426
Ensemble of 3.1 + 3.3 + 3.4 + 3.5 + 3.6	0.44476
Ensemble of 3.1 + 3.3 + 3.4 + 3.5 + 3.6 + 3.8	0.45078
Ensemble of 3.1 + 3.3 + 3.4 + 3.5 + 3.6 + 3.7 + 3.8	0.47029
Ensemble of 3.1 + 3.2 + 3.3 + 3.4 + 3.5 + 3.6 + 3.7 + 3.8	0.46490

Below is an example of an image search result based on our model:

Input query: A man is doing a trick on a skateboard. A skateboarder doing a trick in the air, on a wooden ramp with a blue sky and clouds in the background. A person is performing a trick on a skateboard. There is a skateboarder that has jumped on to a ramp. A man is jumping on a skateboard in a park.

Output images:



6 Conclusions

The model with the best performance is image feature vector to description BOW using PLS regression with tags fine tune. However, other models also give us good result and combining them gives us an even better result. By experiments, when ensemble models, the more diverse of the modes and input, the better result we will get. The final best score is 0.47029 which means for almost half of the time we return the best image in the top 10 out of 20 results.

7 Future Works

In this paper, we explore different algorithms to model the mappings between cross-modal data and essentially we believe it is a regression problem. Besides the methods we mentioned above, there are still a lot of rooms for further experiments. Some qualified techniques and their pros and cons, specifically for this image reverse search task, are summarized below:

- linear regression, ridge regression, lasso, elastic net
- partial least squares regression(PLSR), principal component regression(PCR)
- decision trees (CART, CHAID, C5.0)
- gradient boost
- support vector machine regression
- neural networks (and deep learning)

The other course section's teaching assistant has provided a baseline algorithm notebook based on ridge regression combined with word2vec features, achieving final Map score of 0.11. It's not a desirable score, and we believe parts of the reason is that it is essentially a linear classifier, and will handle high-dimensional and sparse data poorly even though it is good at finding the correct features dimensions. Considering our tasks of mapping between image feature and sentence bag of words model, we finally opt out this option as we want to treat the image feature vector as a whole and won't discard any dimension. But for other scenarios, these are well-designed techniques to find the best-suited feature dimensions given data.

Compared with other methods mentioned above, we heavily exploited PLSR with find-tunes tricks on tags, as shown from the above result table.

References

[1] Kiros, R., Salakhutdinov, R. & Zemel, R. (2014). Multimodal Neural Language Models. *Proceedings of the 31st International Conference on Machine Learning, in PMLR.* 32(2):595-603.

[2] Mnih. A., & Hinton, G. (2009). A Scalable Hierarchical Distributed Language Model. *Advances in Neural Information Processing Systems 21* 1081–1088.

[3] Bengio, Y., Ducharme, R., Vincent, P., & Janvin, C. *A neural probabilistic language model.* J. Mach. Learn. Res., 3:1137–1155, March 2003.

[4] Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., & Dean, J. (2013). Distributed representations of words and phrases and their compositionality. *Advances in neural information processing systems* (pp. 3111-3119).

[5] Daniel, C., Yinfei, Y., Sheng-yi, K., Nan, H., Nicole, L., Rhomni, St. J., Noah, C., Mario, G., Steve, Y., Chris, T., Yun-Hsuan, S., Brian, S., & Ray, K. Universal Sentence Encoder. *arXiv:1803.11175*, 2018.

[6] K. He, X. Zhang, S. Ren and J. Sun, Deep Residual Learning for Image Recognition, *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 770-778.

[7] PLSRegression: `https://scikit-learn.org/stable/modules/generated/sklearn.cross_decomposition.PLSRegression.html`

[8] MPLRegression: `https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPRegressor.html`

[9] Google Universal Sentence Encoder: `https://tfhub.dev/google/universal-sentence-encoder/2`