Image Searching Engine:A Brilliant Name for Our Supper Awesome Paper

Your names and group number

December 5, 2017

Abstract

In this paper, we proposed a solution to deal with an image searching problem.

1 Introduction

In this paper, we are trying to build an image searching engine that can search for relevant images given a natural language query. For instance, if a user types "dog jumping to catch frisbee," Our system will rank-order the most relevant images from a candidates' pool and return the top twenty images that are relevant.

In this paper, we will introduce the experiments we performed involved KNN, random forest, neural network and SVM.

1.1 Problem Definition

During training, we have a dataset of 10,000 samples. Each sample has the following data available: A 224x224 JPG image. A list of tags indicating objects appeared in the image. Feature vectors extracted using ResNet. A five-sentence description.

During testing, our system matches a single five-sentence description against a pool of 2,000 candidate samples from the test set. Each sample has: A 224x224 JPEG image. A list of tags for that image and the ResNet feature vectors for that image.

Output: For each description, our system will rank-score each testing image with the likelihood of that image matches the given sentence. Then the system returns the name of the top 20 relevant images, delimited by space.

For Evaluation: There are 2,000 descriptions, and for each description, we compare against the entire 2,000-image test set. That is, rank-order test images for each test description. Then we use MAP@20 as the evaluation metric. If the corresponding image of a description is among the algorithm's 20 highest scoring images, this metric gives a certain score based on the ranking of the corresponding image as follow.

$$Score = \frac{20 + 1 - i}{20} \tag{1}$$

1.2 Related Work

Will talk a little bit of how KNN, random forest and neural network are important in doing image searching For image searching, there are several models that are

2 Models Architecture

2.1 The KNN Approaches

For the naive version of our solution, we are using a KNN approach. And we split the task into 4 steps:

1. data preprocessing
2. building feature vectors for test queries and candidates
3. calculate pairwise similarities between queries and candidates
4. retrieve the top 20 nearest neighbor from the candidates' pool

2.1.1 Data Preprocessing

For this project, data preprocessing takes up a large portion of the work and by choosing better strategies of preprocessing, we could achieve better performance of the overall model. Here are the few preprocessing strategies we used.

For the five-sentence description, we (a)converted to lower case, (b)stripped punctuations, (c)removed stop words, (d)converted all words to stem words. This is just a regular preprocessing strategy for text input. After we reviewed some literatures, we found that IFTDF would be an useful tool to perform preprocessing on small chunks of text. To be continued. . . .

2.1.2 Building Features

Since we need to retrieve images based on five-sentences of description queries, so we need to build the feature vectors for these queries. In addition, we would perform KNN on the features, which means we need to compare each query feature with every other candidate's feature. And we do not have the descriptions in the candidates' data. Therefore we need to perform a mapping that maps the description of the input queries with the tags of the candidates'. We will discuss this part in detail in the next section.

2.1.3 Mapping Descriptions to Tags

By mapping descriptions to tags, we want to express both of the input queries and the candidates into numerical features based on the same dictionary of words. Therefore we first built a dictionary of tags from the training data. We looked into all the tags in the training data set and count the frequency of each tag. Then we assign the top 100 most frequent tags as our dictionary.

Upon building the dictionary, we could convert the testing queries and the candidates' tags into numerical features through Bag of Words model.(???Am I right???) By now, we have mapped the description data of a query to tags.

2.1.4 Cosine Similarity

After the last step, we got the feature vectors for the test data and candidate data. Then we compute the cosine distance or Euclidean distances between a test sample and all the candidates. After this step, we simply sort the distance and find the top 20 neighbors as the output data.

As expected, the result of our first attempt was not perfect. We got an accuracy of 0.09806 on Kaggle.

2.1.5 Classic KNN

Apart from the cosine distance we implemented for the KNN model, we also tried to perform the KNN classifier from the scikit learn library and achieved a better score of 0.1189.

2.2 The Neural Network Model

For KNN models, we basically only leveraged the information of input queries and candidates. However, there are several modalities of data in the training set that we have not made use of. In order to use the ResNet features in the training set, we want to use the neural network model to find the pattern between these two spaces and hope to find the mapping between the description and the image features.

2.2.1 TFIDF

We performed the same data preprocessing like we did for the two previous models. Moreover, we utilized another preprocessing tool which is TFIDF, a useful tool kit which helps to generate the most important words in a short paragraph. We used that tool to

2.2.2 Building the features

2.2.3 Results

2.3 The Random Forest Model

3 Experiment

3.1 Our Steps of Refining the Models

4 Results

4.1 Quantum regime

5 Conclusion

References

[1] K. Grove-Rasmussen og Jesper Nygård, *Kvantefænomener i Nanosystemer*. Niels Bohr Institute & Nano-Science Center, Københavns Universitet