Large-Scale Image Search Engine

December 5, 2017

Abstract

In this paper, we proposed a solution to deal with an image searching problem leveraging both of the descriptive information and ResNet features in training data. We designed a new format of feature that combines both the description and tag information of one image into one mixture feature that we call DT. Utilizing Partial Least Squares Regression to map the ResNet features to the DT features and lazy learning models like KNN, we finally achieved an accuracy of around 0.40 on Kaggle.

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 our experiments we performed lazy learning methods utilizing the input queries and the candidates information. However the results are disappointing so we tried to perform eager learning with Neural Networks and Random Forest. Finally we proposed a mixture of lazy learning and eager learning that combined the image features and DT(Description+Tags) features and achieved an accuracy of around 0.40 on Kaggle.

1.1 Problem Definition

D**uring 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

Until recently, image retrieval was merely based on scanning and indexing images. The scanning part involves image processing to extract features from the image. The trick here is to achieve a balance between overly sensitive features and yet ones that are distinctive enough that we don't get too many duplicates. The indexing part involves large scale database technology to store and retrieve features very quickly.

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 neighbors 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. Here are the few preprocessing strategies we used.

For the five-sentence description, we

- converted to lower case
- stripped punctuations
- removed stop words
- 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.

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 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. 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. $similarity(X,Y) = cos\alpha = \frac{\vec{A} \cdot \vec{B}}{||A|| \cdot ||B||}$ 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.

2.2.2 Building the features

For the features, we only build the feature vector for the description data of the input the same way as what we've done with the KNN approaches. And we used the ResNet image feature directly.

2.3 The Random Forest Model

With Random Forest model, we try to map the relationship between image description and its tags again through eager learning. Recall that Random Forest can use for regression with the following formula:

$$\hat{f}_{rf}^{B} = \frac{1}{B} \sum_{b=1}^{B} T_b(x) \tag{2}$$

Given training descriptions and their corresponding tag results, the model will bag repeatedly B times and selects a random sample from the training data and fits trees to these samples. This approach produces better performance than KNN because it decreases the variance in description without increasing the bias.

2.4 Image Feature to DT (Description + Tag) Model

In our previous models, we tried to create the mapping between the tag and description, which was based on the assumption that these text features are basically accurate. However, we found that sometimes the description and tags may not be correct and reliable. In this case, we try to approach by using the image ResNet features, which can be more trust-able and can represent the actual image in a more accurate way, for mapping to description and tag.

The basic idea is to train a model and create the mapping between image feature and the mixture of description and tags (DT). After that, we can apply it to predict the DT for all the candidate images. At the prediction step, we employ different models such as Neural Network and PLS. We will discuss further about which model we choose in the subsection below. When we receive the user query, we can then use KNN (k = 20) to locate the most similar candidate images by comparing the query with their DTs.

Since we use the models from open libraries, our main work here is preprocessing and post processing.

2.4.1 Preprocessing Strategy:

This is where the mixture happens. The purpose of this step is stated above.

Loading step:

For description, we simply load it as raw data and return as a list with index is file number. We will do preprocessing for description at mixing step. For tag, before returning it as a list like description, we simply split by colons and stem the tokens.

Mixing step:

As mentioned, we will mix tags with their corresponding description by appending them. In this step, before appending, we increased the word stemming quality by splitting chunk of words to tokens, lowercasing them, removing any word that is stop word, finally we stem the words. After preprocessing the description here, we make them become a single string and append tags.

Eventually, after this step, we get a list of string where each string is a mixture of description and tag that have been stemmed for each image.

For example:

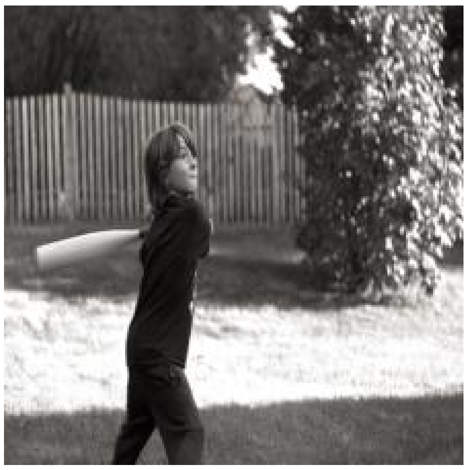

Figure 1: 4.jpg from the training data.

Description: young child yard hold bat boy rare back basebal bat yard littl boy basebal bat yard boy black cloth hold basebal bat shoulder near fenc tree littl boy play yard basebal bat person

Tags: person person sport basebal bat

Result: young child yard hold bat boy rare back basebal bat yard littl boy basebal bat yard boy black cloth hold basebal bat shoulder near fenc tree littl boy play yard basebal bat person person sport basebal bat

2.4.2 Bag of Word

With the list of mixed strings that we got from the previous step, we start bagging the words by counting with CountVectorizer. The purpose is to have a numerical representation of each word so we can weight them later in model training. See example in the next subsection.

2.4.3 Post Processing Strategy

In order to train the 'MLPClassifier' model, we need to normalize the BoW result into a binary format. We simply traverse the BoW result and set all the non-zero frequency value to one. In this way, we got the BoW feature which contains only zero and one values. Here is one example:

$$\underbrace{[0, 12, 1, 3, 0, 3, 4, 2, 0]}_{BoW} \longrightarrow \underbrace{[0, 1, 1, 1, 0, 1, 1, 1, 0]}_{Normalized \quad BoW} \tag{3}$$

2.4.4 Model Training

Essentially, we employ "Hit-and-Trial" strategy with some in-class knowledge to choose our model:

Neural Network:
From what we learnt in class, Neural Network models have great strength in extracting structural information in images. Therefore we leveraged this method to train a model with 3 hidden layers and 1000 nodes in each layer to create a mapping between image features and their DT features.

Partial Least Squares Regression:
While researching, we came across Partial Least Squares Regression. The model bears some relation to principal components regression and deals with multicollinearity. It's thus well known as a Regression method that takes into account the latent structure in both datasets. Clearly it helps improve the mapping between image features and DT data better comparing to Neural Network due to modeling several responses variables at the same time taking into account their mixture structure that we created when doing preprocessing.

2.4.5 DT Prediction

Once we got the model, we can apply it to predict the DT for all the candidates. In addition to that, we can also add the tags information to the predicted DT in order to include more details (Final DT).

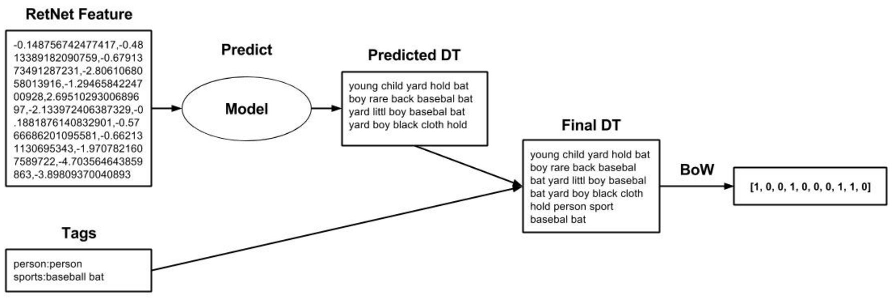

Figure 2: DT Prediction Pipeline

2.4.6 KNN

After performing the DT prediction, every candidate in the testing set has a predicted DT, which can be used to compare with the user input query. In order to find the top 20 most related images from the testing set, we can compare the input query with every predicted DT and apply the KNN algorithm to find the nearest neighbors.

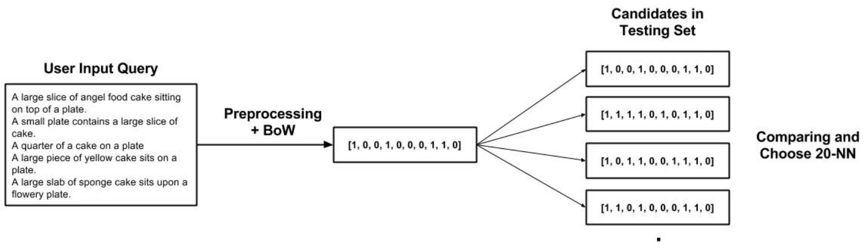
User Input Query
A large slice of angel food cake sitting on top of a plate.
A small plate contains a large slice of cake.
A quarter of a cake on a plate
A large piece of yellow cake sits on a plate.
A large slab of sponge cake sits upon a flowery plate.
Preprocessing + BoW
[1, 0, 0, 1, 0, 0, 0, 1, 1, 0]
Candidates in Testing Set
[1, 0, 0, 1, 0, 0, 0, 1, 1, 0]
[1, 1, 1, 1, 0, 1, 0, 1, 1, 0]
[1, 0, 1, 1, 0, 0, 1, 1, 1, 0]
[1, 1, 0, 1, 0, 0, 0, 1, 1, 0]
Comparing and Choose 20-NN

Figure 3: Retrieve Top 20 images

3 Experiment

Up to this point, we have listed the models and the reasons we decide to go with in details and chronological order.

In brief, there are 3 stages we have been through while building this image serch engine:

3.1 Lazy Learning Approach

- First attempt, we tried the most simple approach with cosin distance mapping between description and tag. It gives us less than 0.10 accuracy. But, we learnt that cosine is better than euclidean distance.

- Second attempt, simple KNN mapping between feature and description. We simply use default euclidean distance, but the result is around 0.11 which indicates that the mapping between image feature and description gives us better result than tag and description.

3.2 Eager Learning Approach

- Third attempt, random forest between description and tag. The reason we choose random forest is because it does well with multi-class classification.

- Fourth attempt, neural network between description and feature. We choose the model because it can extract and maintains the structural features from image.

3.3 Combination of Models

Final attempt, we combine the knowledge we learnt from the previous four attempts.

1. Feature $\rightarrow$ Description/Tag is in general better: we employ feature to mixture of description and tag.

2. Eager learning does better: we used PLS model for dealing with multi-collinearity.
3. Cosine distance is better than euclidean: we employ this at our KNN step for choosing top 20 candidates.

4 Results

Here are the results of all our experiments:

Approach	Kaggle Score
Cosin Distance: Description-Tag	0.09806
KNN: Feature-Description	0.11896
Random Forest: Description-Tag	0.21744
Neural Network: Description-Feature	0.23710
PLS: Feature-DT	0.39701

Table 1: Kaggle Results

5 Libraries

- nltk
- scikit-learn: TfidfVectorizer
- sklearn.neighbors:KNeighborsClassifier
- sklearn.neural network: MLPClassifier MLPRegressor
- sklearn.cross decomposition: PLSRegression
- sklearn.ensemble: RandomForestRegressor
- sklearn.neighbors: KDTree

References

[1] Hasegawa R, Hotta K., *Stacked partial least squares regression for image classification.* Pattern Recognition (ACPR), 2015 3rd IAPR Asian Conference on. IEEE & 2015: 765-769.

[2] BoW Tutorial: https://www.kaggle.com/c/word2vec-nlp-tutorial#part-1-for-beginners-bag-of-words

[3] PLS: https://www.youtube.com/watch?v=WKEGhyFx0Dg