Building Multiple Models For Image Search Engine

Zhiling Hu, Yuekun Wang, Irene Wei
Cornell Tech
New York, NY 10044, USA
{zh347 & yw2222 & zw575 }@cornell.edu

***Abstract* - This paper attempts multiple approaches to build a search engine for searching relevant images given natural language queries. The basic understanding is to map textural information and image representation. Through the process of processing the data, we had applied methods in machine learning including Bag of Words Model, Random Forest Regression, Linear Regression, k-Nearest Neighbors Algorithm, Principal Component Analysis, and Partial Least Squares Regression. Among all the experiments, our highest score on Kaggle is 0.2921, which uses Partial Least Square Regression on images' 5-sentences descriptions and image features extracted from the intermediate layer of pre-trained Residual Network. The paper will discuss each approach and corresponding result in details.**

I. Introduction

The Kaggle challenge we were working on was to build an image search engine based on natural language queries. Our goal is to take the five-sentence description of the photo to fit top twenty possible photo guesses.

A. *Metrics*

1)Data: We were given a training set of 10,000 elements and a testing set of 2,000 elements. Each element in both datasets contains a 224x224-pixel JPG image, a five-sentence description, a list of tags, and two sets of image features.

2) Evaluation: Students are supposed to submit their top 20 candidates with the top relevance to the description given. The system will evaluate the results using the MAP@20 metric and return the score based on the ranking of the correct image.

II. Development Process

A. *Understanding the data*

The data that we use, and how we use it, will likely define the success of our machine learning models. Picking the right data of the problem may be the point of biggest leverage on this project. To get the most from the provided data, we parsed all data files we were most interested in, including descriptions, categories and tags, and intermediate features. We found out that there are 10,000 tags coming from 12 super categories, 80 subcategories.

The super categories are listed as below:

['outdoor', 'food', 'indoor', 'appliance', 'kitchen', 'sports', 'person', 'animal', 'vehicle', 'accessory', 'electronic', 'furniture']

The sub categories are listed as below:

['toilet', 'fire hydrant', 'sports ball', 'bicycle', 'kite', 'laptop', 'potted plant', 'tennis racket', 'teddy bear', 'donut', 'snowboard', 'carrot', 'motorcycle', 'oven', 'keyboard', 'scissors', 'chair', 'couch', 'mouse', 'airplane', 'boat', 'apple', 'sheep', 'horse', 'sandwich', 'banana', 'cup', 'tv', 'backpack', 'toaster', 'bowl', 'microwave', 'bench', 'book', 'elephant', 'orange', 'tie', 'stop sign', 'knife', 'pizza', 'fork', 'hair drier', 'frisbee', 'umbrella', 'parking meter', 'bus', 'suitcase', 'bear', 'vase', 'toothbrush', 'spoon', 'train', 'sink', 'wine glass', 'handbag', 'cell phone', 'bird', 'broccoli', 'refrigerator', 'remote', 'surfboard', 'cow', 'dining table', 'hot dog', 'car', 'clock', 'skateboard', 'dog', 'bed', 'cat', 'person', 'skis', 'giraffe', 'truck', 'bottle', 'baseball bat', 'cake', 'baseball glove', 'traffic light', 'zebra']

The number of digital images is growing extremely rapidly, and so is the need for their classification. But, as more images of pre-defined categories become available, they also become more diverse. For example, the text listed as below comes from one of the tags and categories files. There are only two categories here, but under the electronic category, there are four distinctive items.

electronic: laptop
electronic: mouse
indoor: book
electronic: keyboard
electronic: cell phone

In that sense, images associated with (super)categories are not quickly reachable through a rough category search. Ultimately, the categories themselves need to be divided into subcategories to account for that semantic refinement. Hence, in the consideration of accuracy, we expected to exploit subcategories instead of super categories, as subcategories are more fined and can improve the classification.

B. *Preprocessing Strategy/ Basic Cleaning*

In order to achieve better results from the predictive model in Machine Learning projects, the format of the data has to be in a proper manner. The data set should be formatted in such a way that more than one machine learning algorithms can be executed, and best out of them can be clearly chosen. Since these descriptions are text in the fashion of sentences, they may contain significant amounts of noise and garbage. We picked some preprocessing strategies to clean up the descriptions both from training and testing data sets.

We wrote a function that split each description into its individual words and returned them as a dictionary by the end. We converted all of the words to lowercase and stripped the very common words (such as "the", "a", "an", etc.), also known as stop words. Besides, we lemmatized of all the words, converting the words into their base word or stem word. We took advantage of the *NLTK* library.

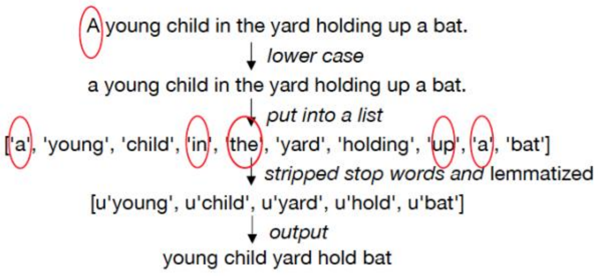
A young child in the yard holding up a bat.
lower case
a young child in the yard holding up a bat.
put into a list
['a', 'young', 'child', 'in', 'the', 'yard', 'holding', 'up', 'a', 'bat']
stripped stop words and lemmatized
[u'young', u'child', u'yard', u'hold', u'bat']
output
young child yard hold bat

Fig. 1 Simulation of data cleaning strategies provided by NLTK library.

III. Strategies

A. *Bag of Words*

We cannot work with text directly when using machine learning algorithms. They will need some sort of feature vectors in order to perform the task. The simplest way to convert text to a vector format is the bag-of-words approach, or *BoW*, where each unique word in a text will be represented by one number.

BoW is simple in that it throws away all of the order information in the words and focuses on the occurrence of words in a document. This can be done by assigning each word a unique number. By counting the frequency of words in a text message, any descriptions we see can be encoded as a fixed-length vector with the length of the vocabulary of known words, which is the dictionaries we created through running *BoW* before. This step helped us analyze part of descriptions and tags and reduce some noise so that we can map the most important information to images.

B. *Word Counts with CountVectorizer*

One way to represent a text document is to count the instances of every word in the document. We used the *CountVectorizer* in *scikit-learn* to generate a vector of word counts for the preprocessed text. We then plotted the top 50 most frequently occurring words. As you can see, they are not some commonly used words (the, in, and of), and thus could be quite unique for identifying objects. Therefore, the preprocessing strategies we used were helpful in identifying potentially meaningful words.

We were using *min_df = 5* which will include all the words which are found in at least five times among all. Here, *min_df* could be considered a hyperparameter itself to remove the most commonly used words.

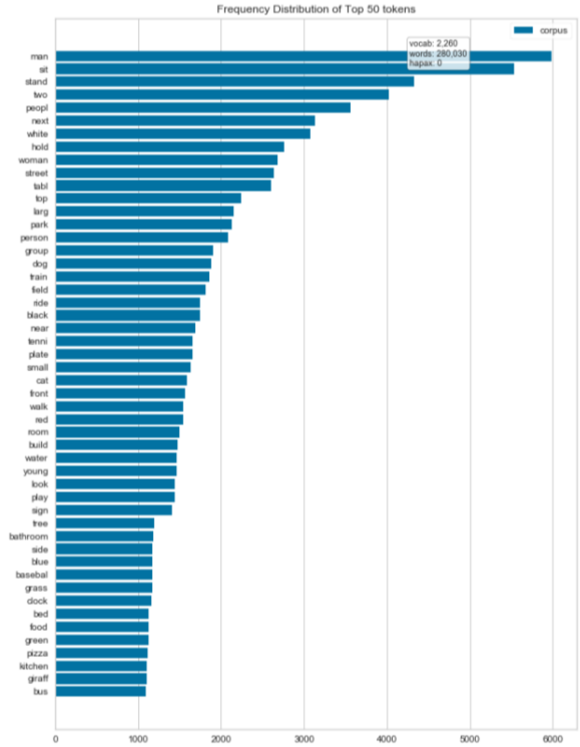

Fig. 2 Frequency Distribution of Top 50 tokens with CountVectorizer .

C. *TF-IDF(Term Frequency-Inverse Document Frequency)*

TF-IDF (term frequency-inverse document frequency) is a measurement to qualify the document. This approach calculated frequency of a term adjusted for how rarely it is used.

The inverse document frequency for any given term is defined as (1).

$$idf(term) = ln(\frac{n_{documents}}{n_{documents\ containing\ term}}) \tag{1}$$

We used such statistic to reflect how important a word is to a specific one relative to all of the words in a collection of text (the corpus). The tf-idf value increases proportionally to the number of times that word appears in the document but is offset by the frequency of the word in the corpus.

Already equipped with a learned *CountVectorizer*, we simply implemented it with the *TfidfTransformer* to just calculate the inverse document frequencies and start encoding documents. After computing the *TF-IDF* weight for each word that appears in a given collection of documents, we were able to come up with scores that are normalized to values between 0 and 1. For example, as we wrapped up two data frames for data processed with *CountVectorizer* and ones with *TF-IDF*, 3 on column "young" (one of the top 50 tokens) row 3 changed to 0.215919. The encoded document vectors can then be used directly with most machine learning algorithms.

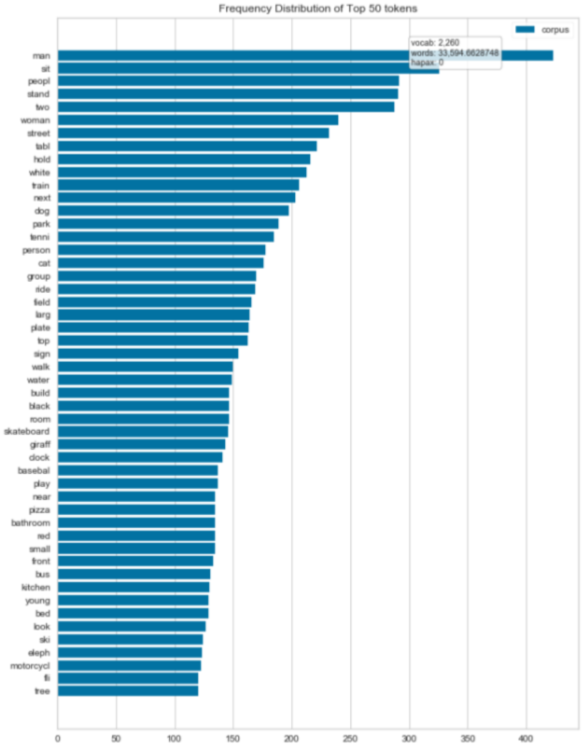

Fig. 3 Frequency Distribution of Top 50 tokens with TF-IDF.

D. *Word2Vec*

Based on word embedding, *Word2Vec* takes the semantic meaning of the words and their relationships between other words. It learns all the internal relationships between the words and represents the word in dense vector form. We computed the Word2vec of each of the words and added the vectors of each word of the sentence and divide the vector with the number of words of the sentence.

E. *PCA for bag of words model*

We supposed that the features in the bag of words model have large redundancy. Considering some models we might be using later on, such as logistic regression and random forests, we thus implement *PCA* to reduce the dimension of features from 2260 to 2048. 2048 is the dimension size per image intermediate features - *Pool5*.

PCA projects onto a smaller affine subspace. Intuitively, we expect *PCA* to work better.

IV. Models and Experiments

A. *Linear Regression*

Linear Regression is the technique used for fitting and studying the straight-line relationship between two variables. We sought to find an equation that describes or summarizes the relationship between descriptions and text, so we run a linear regression on image descriptions and tags. Just as what we expected, the result is not good, since this is a naive first attempt.

B. *Random Forest*

Random forests are a combination of tree estimators such that each tree depends on the values of a random vector sampled independently and with the same distribution for all trees in the forest. Using a single decision tree doesn't guarantee that the different patterns of the data are correctly learned. Meanwhile, Random Forest is a very good, robust and versatile method.

Therefore, we tried *Random Forest Regression* on descriptions, tags, and image features with *PCA*. The score is slightly improved compared to the Linear Regression result. Furthermore, we find that using tags feature is slightly better than image features, but it's also noticeable that it doesn't work friendly with high-dimensional sparse data. *BoW* representation is a perfect example of sparse and high-d data. However, we just had *PCA* to reduce the dimensions.

One of the approaches we took was to apply random forest to perform the hierarchical clustering of *BoW* that summarizes the pattern content of the input with respective images.

C. *Partial Least Squares Regression*

Partial least squares regress (*PLS regression*) is a statistical method that bears some relation to principal components regression. It works like a compromise between ordinary least squares and principal components analysis. *PLS regression* is the vector upon which an ordinary least squares regression line has the largest projection.

We use *PLS* on image descriptions and pool5 features. Although we only did one experiment due to the running time, the result of *PLS* is outstanding, so we believe this is so far the best approach that we could use, and it can be further investigated.

D. *KNN (K-Nearest Neighbors)*

KNN is powerful because it does not assume anything about the data, other than a distance measure can be calculated consistently between any two instances. It's non-parametric or non-linear as it does not assume a functional form.

In our case, we want to predict 20 elements that are closest to the predicted tags and features, which means that k equals to 20.

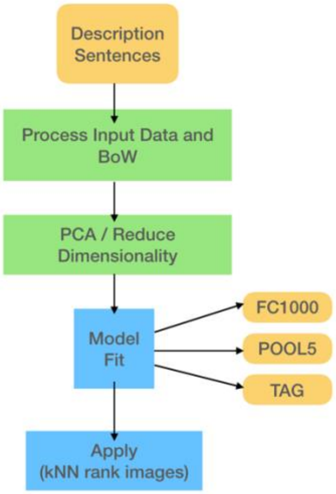

Fig. 4 General pipeline for models

V. Final Model

In this final model, we use *PCA* to reduce the dimensionality of the *POOL5* and descriptions features in training and testing data. Then we use *PLS regression* to fit the training set and predict with the testing data.

TABLE I
Algorithms comparison and analysis

Model	**Kaggle Score**
BoW + LinearRegression + (KNN) (descriptions and tags)	0.13968
BoW + RandomForestRegression + (KNN) (descriptions and tags)	0.22065
BoW + PCA + RandomForestRegression + (KNN) (descriptions and pool5 feature)	0.09402
BoW + PCA + PartialLeastSquaresRegression + (KNN) (Descriptions and pool5 feature)	0.29212

VI. Conclusion And Future Work

Other well-known methods such as Random Forest, Linear Regression models were used with the same datasets and classification results are shown in the table above. All of them demonstrated some classification rates, but not as good as those obtained by *Partial Least Squares Regression*.

The highest score obtained on the public leaderboard is 0.29212. This is an initial investigation. Further study might be to focus on tags instead of subcategories and descriptions. Furthermore, cross-validation gives a measure of out-of-sample accuracy by averaging over several random partitions of the data into training and test samples. It is often used for parameter tuning by doing cross-validation for several (or many) possible values of a parameter and choosing the parameter value that gives the lowest cross-validation average error. We should have implemented such a strategy with *KNN* if more time is allowed.