An Approach for Building a Large-scale Image Search System

Haomiao Han, Hyein Baek
Cornell Tech, Cornell University
New York, NY
{hh696, hb437}@cornell.edu

Abstract

In this paper, we propose an approach for searching relevant images given a text input: we used the image vectors generated by ResNet as well as image tag vectors generated by word2vec to build our feature matrix and to use it to predict descriptions, which is our input data. We used Ridge Regression as our algorithm to train the data and to perform the prediction. Our approach was able to achieve a much higher performance than the baseline.

1 Introduction

Image searching has always been a popular research topic in the machine learning field and has been addressed by various approaches. In this project, we were given a training set with 10,000 images with 10,000 corresponding tags (human-generated, used to describe the images within a few keywords). We were also given 10,000 text descriptions of the images, each containing several sentences, as our search queries. Our objective is to come up with the most relevant images (out of 2,000 images in the test set) for each text description (out of 2,000 descriptions in the test set).

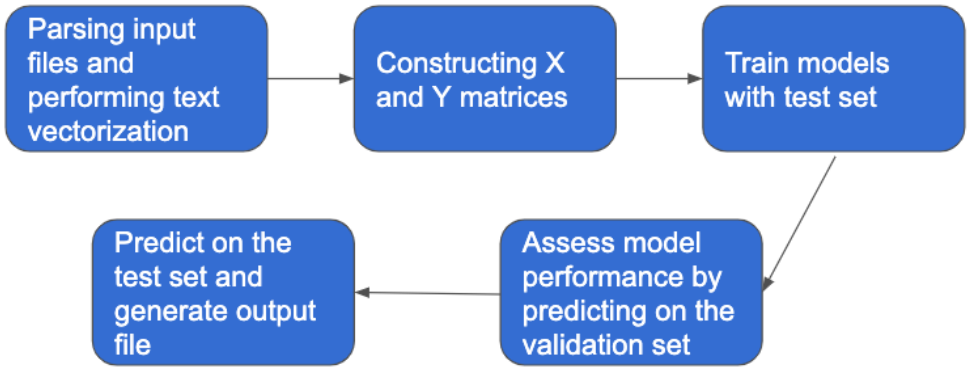
For this project, we experimented with several preprocessing techniques as well as different ways to construct the feature matrix. We also tried a number of different machine learning algorithms to fit our data.

2 Experiments

We have experimented with several approaches before coming up with our best-performing approach. A overview of the approaches we have tried and their performances is shown by the table below:

Approach #	Type of Task	Feature Matrix (X)	Target Variable (Y)	Machine Learning Algorithm	Similarity Metrics
1 (Baseline)	Regression	Text Descriptions	Image vectors (after random projection)	Ridge	Euclidean
2	Regression	Text Descriptions (paragraph vectors)	Image vectors (after random projection)	Ridge	Cosine
3	Regression	Text Descriptions (paragraph vectors)	Image tag vectors	Ridge	Cosine
4	Regression	Image vectors	Text Descriptions (paragraph vectors)	Ridge	Cosine
5	Regression	Image vectors + Image tag vectors	Text Descriptions (paragraph vectors)	Random Forest	Cosine
6	Regression	Image vectors + Image tag vectors	Text Descriptions (paragraph vectors)	Extra-trees	Cosine
7	Regression	Image vectors + Image tag vectors	Text Descriptions (paragraph vectors)	Ridge	Cosine
8	Classification	Text Descriptions (paragraph vectors) + Image vectors	0/1 (0 for correct match, 1 for incorrect)	Random Forest	Cosine

The pipeline of our program is shown as follows:



2.1 Feature Matrix Construction

We have used different ways to construct our X Matrix as well as our Y Matrix. The baseline provided to us uses the text description vectors as the X Matrix; for the Y Matrix, this approach first constructed a random matrix of size $(1000, 100)$, then performed matrix multiplication between the original image vector matrix and the randomly generated matrix. After that, the dimensionality of image vector matrix is reduced from 1,000 to 100. Then, this dimensionality reduced matrix was used as the Y Matrix. Our approach 2 also uses description vectors as the X matrix, but we used a different way to vectorize the text, which is detailed in section 2.2.

In our approach 3, we parsed the image tag files and converted them into vectors using `word2vec`. This matrix was then used as the Y matrix. The X matrix is unchanged from approach 2.

For all of the approaches above, we trained the models and predicted on the test set to get our prediction results. After that, for each result, we sorted the 2,000 images in the test set by their similarities to the result, using two different similarity measures detailed in section 2.3. The output data consists of the top 20 images that are most similar to the prediction results for each input.

Our approach 4 flipped the process: instead of using descriptions as the X matrix and images (or image tags) as the Y matrix, we used images (and image tags) to predict the descriptions. The main reasoning behind our approach is that in approaches 1-3, we would need to perform a random projection in order to reduce the dimensionality of the Y matrix; this is not ideal as dimensionality reduction may not preserve all of the features in the original data. However, if we do not perform dimensionality reduction, we will then be trying to predict a Y matrix with 1,000 dimensions using an X matrix with only 300 dimensions. This setup will also negatively impact the performance. However, in our approach 4, since we are using the image vectors (which has higher dimension) as

the X matrix and the description vectors (which has lower dimension) as the Y matrix, we no longer need to perform dimensionality reduction; we believed this might improve the performance of the model. Approach 5 is similar to approach 4: we concatenated the image tag vectors with the image vectors and used the combined vectors as the X matrix.

The training and prediction process for approaches 4 and 5 is similar: we trained the models and predicted on the test set to get the prediction results. Then, for each description vector in the test set, we calculate the cosine similarity between that vector and all of the predicted description vectors. We then `argsort` the similarities and the indices returned by the `argsort` would be the indices of the most relevant images to the description text.

In approaches 6 and 7, we used the same X and Y matrices as approach 5, but experimented with different algorithms to fit the data.

In our approach 8, we change the type of the task from a Regression task to a Classification task. For each description, we concatenated the description vectors and the corresponding image vectors. We then assign a target variable of 1 to this row, meaning that the description matches the image. Additionally, for each description, we randomly generated three different image vectors that do not correspond to the description. These rows are assigned target variables of 0 as the description vectors do not match image vectors. This combined matrix of image vectors and description vectors will be used as the X matrix, and the target variables of 0 and 1 will be used as the Y matrix. In other words, the shape of the X matrix will be $(24000, 1300)$, and the shape of the Y matrix will be $(24000, 1)$.

After training the model on the training set, we construct the validation set by concatenating all of the 2,000 image vectors in the validation set to each of the 2,000 description vectors. The row with the image vector matching the description vector is assigned a target variable of 1, and the rest of the 1,999 rows are assigned target variables of 0. This results in a X matrix with size $(4000000, 1300)$ and a Y matrix with size $(4000000, 1)$. We then predict ton the validation set and get the probabilities of each class for each row using the `predict_proba` function in various `sklearn` models. Then, the top 20 images selected for each description will be the images whose row have the largest probability of having a target variable of 1 given our prediction results. We should note that due to the relatively large size of the validation set matrix, we were only able to experiment with the first 400 descriptions (and all 2,000 images) of the validation set, resulting in a X matrix with size $(800000, 1300)$ and a Y matrix with size $(800000, 1)$

2.2 Data Preprocessing

We used `word2vec` to convert texts into their vector representations. The baseline solution (approach 1) used `gensim` as a natural language processing toolkit and converted each word in one description text paragraph into vectors, then used the average of all individual word vectors of the paragraph to represent the paragraph. In all subsequent approaches, we used `spaCy` to convert the entire paragraph into vectors. We believed that converting the entire paragraph into vectors could result in a better representation of the paragraph than simply using the average of individual word vectors, as using individual word vectors does not take the sequence of the words in the paragraph into account.

The image tags are also converted into vectors using `word2vec`: when we parse the text, we only include the sub-category of the image tags, and all of the sub-categories of one image are combined and are considered as one "paragraph". We then generate the vectors of the tags using `spaCy`.

The images are vectorized using `ResNet`. These vectors are provided in the original dataset.

2.3 Algorithm Selection

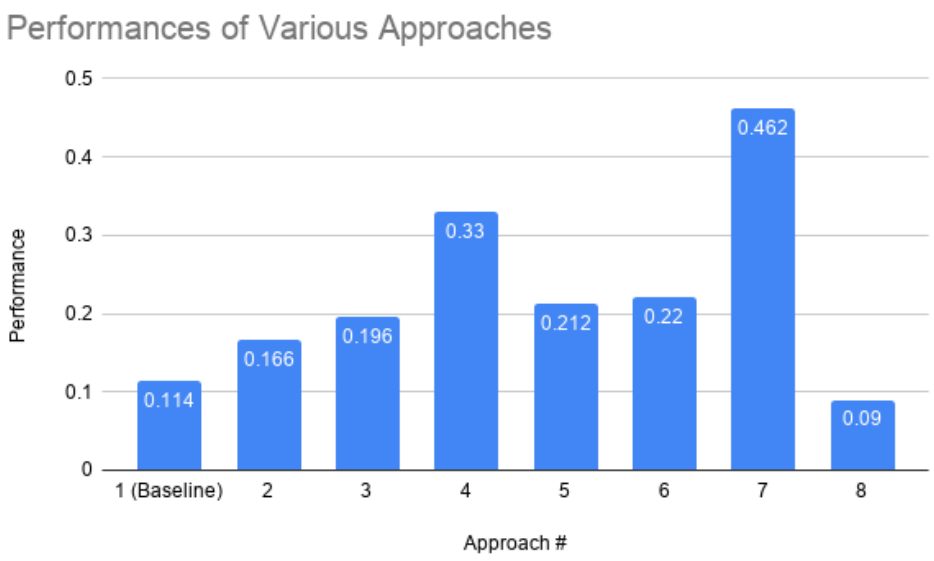
We experimented with different machine learning algorithms to train our model. We have used Ridge Regression, Random Forest and Extra-Trees in our experiments. In the training process, we used cross validation to select the hyperparameters for each model.

2.4 Similarity Metrics

The baseline solution used Euclidean distance to measure the similarity between vectors; however, in our opinion, Cosine similarity (or Cosine distance) is a better way of measuring the similarity between text or image vectors. Thus, we used Cosine similarity as a similarity metrics in our experiments.

3 Results

We used MAP@20 (which is the specified performance metric) to evaluate the performances of our models. The graph below shows our experimentation results:



3.1 Result Analysis

From the results, we can see that Ridge Regression seems to be the best model to fit the dataset, as our approaches 5 and 6 (which used other regression algorithms) have a significantly lower performance than the approach 7, which has the same input X and Y matrices and used Ridge as the regression algorithm.

In addition, our approach 2 has a better performance than the baseline. This demonstrated that our method of converting the description text into vectors by vectorizing the entire paragraph outperforms the baseline method of taking the average of individual word vectors in the description text and using the average vector to represent the entire paragraph.

Interestingly, we can also see a rise in performance in approach 3 compared to approach 2. It seems that not using the image vectors in the Y matrix at all and substituting it with the image tag vectors is a better way to construct the input X and Y matrices, compared to the baseline solution.

The highest rise in performance comes to our approaches 4 and 7, where we flipped the X and Y matrices and used image vectors to predict description vectors. Concatenating image tag vectors to the X matrix (in approach 7) also seems to be a good way of improving performance.

Contrary to our prior beliefs, our approach 8 was not able to produce a satisfying performance. It seems that this task may best be addressed using regression, rather than classification.

3.2 Final Results

We chose approach 7 as our best performing approach and used it to predict on the test set. Our approach was able to achieve a MAP@20 performance of 0.4654 on the test set, according to the Kaggle public leader board. On the validation set, the predictions generated by our approach also have a median index of true image of 2.0; in other words, in at least half of the validation set, our model was able to find the corresponding image to the description text and rank the image as either the most or the second most relevant image to the search query.

4 Related Work and References

Some parts of our code - specifically, the parts of parsing input files, predicting on the validation/test sets and assessing model performance - are based on the provided baseline solution. The list of stop words that we have used in text preprocessing also came from a previous class that one of the authors have taken.

Additional references are listed below:

- (Removing punctuation from strings) `https://bit.ly/36lZdjm`
- (Ways of converting text into vector) `https://bit.ly/2RHqk4t`