Building a Large-Scale Image Search Engine with Various Methods

Abstract

Our challenge was to find relevant images based on a text description. We were given a training set of 10,000 photos and a testing set of 2,000 photos, each with a set of descriptions, tags and feature vectors. Our team used 4 different approaches, each with varying results. The best performing algorithm was one that incorporated boosting, which received an accuracy of 28.164% on Kaggle. The boosting algorithm leveraged a Random Forest Regression model for approximating tags as well as an Image Features to Bag of Words model.

1. Intro

Our goal was to take in a text description made up of 5 sentences that describe a photo, and rank 20 possible photos that fit the description. It is a supervised learning problem as we are given a set of 10,000 training photos with the corresponding descriptions. We are given 2,000 testing photos and 2,000 testing descriptions that are out of order.

Other inputs we are given for both testing and training sets are feature vectors of 1,000 and 2,000 dimensions and textual tags, all of which correspond to a specific image.

Our approaches used primarily a textual based algorithm to find the right photo. This essentially translated to breaking down the input test description and finding the corresponding description or photo in the training set. Once this is obtained, we then find the nearest neighbors in the test set of photos based on different distance measurements calculated.

This paper describes the different approaches in which we mapped the tests description to some estimated metric (such as descriptions or resnet features), and then mapping it back to the test set of photos to produce the most likely images.

1.1 Bag of Words for Description to Tags

In this method, we mainly fit the description training set and the tags as the labels into a model, and predicted the tags for the description testing set. For the description training and testing set, we first constructed a bag-of-word for each. Instead of recording a binary number for whether the word exists or not, we calculated the TF-IDF of each word in the bag of words for all descriptions.

TF-IDF

TF-IDF is term frequency-inverse document frequency, and it is often used in mining text and comparing word documents. TF-IDF is used to calculate weights of words within documents which takes account into two components: term frequency and inverse document frequency.

Term frequency determines how often a word/term appears within a specific document, and the equation is *Frequency of term t in the document/Number of words within the document*. In our case, for each term in the bag of words, we calculated the frequency of the term within a training description.

Inverse document frequency determines how rare a word is within the group of documents, and the equation is $\log_e$ *(Total number of documents / Number of document with the term in it)*. In our case, to calculate the inverse document frequency for a term in the bag of word, we knew the total number of descriptions in the training data is 10,000, and we calculated the number of descriptions that has the term in it.

After computing the TF-IDF, we made a bag-of words for the tags in both the training and testing data. The bag-of-words keeps track of whether or not a tag is associated with an image. Then, we used a few algorithms, with Logistic Regression and Random Forest being the best ones, to train our model.

Logistic Regression

Logistic Regression is a supervised predictive analysis that analyzes one or more independent variables to determine an outcome. The outcome has to be binary, such as TRUE/FALSE, success/failure etc. In other words, it can only have two possible outcomes. In our case, our outcome was "0" or "1", which showed whether a tag is associated with the descriptions or not.

Random Forest

Random Forest is an ensemble learning method for regression and classification that fits decision tree classifiers on the training set and classifies the input into a category. It uses averaging to improve the predictive accuracy and controls overfitting.

We fit the TF-IDFs of the training descriptions as X and the tag bag-of-words as labels into the algorithms, and predicted the tag bag-of-words for the TF-IDF of the testing descriptions. In other words, based on our trained model, we tried to predict what tags would be associated with the testing descriptions. Based on the predicted list of associated tags, we then used KNN to determine which images are the most related.

KNN (K-Nearest Neighbors)

KNN is a supervised method for classification and regression. If k =1, then the input is assigned to the class of that single nearest neighbor. In our case, we want to predict 20 images that are most associated to the given tags, hence, our k would be 20.

Using KNN, we ranked the top 20 images based on the predicted tags.

1.2 Bag of Words to Image Features

For this approach, we began by converting each description to a bag of words feature vector. Before producing our bag of words, we preprocessed the descriptions to lowercase all words, remove punctuation, remove stop words, lemmatize words, and remove white spaces. We were careful not to mix our training and testing data. Therefore, we build our bag of words model using only the training descriptions.

With the bag of words produced, we created a feature vector using the bag of words to represent each description. The feature vector took the frequency of a word occurring within a description into account, but we used the L1 Norm to normalize each feature vector by the number of words in the description.

Next, we built a KNN regressor that fit the training description feature vectors. We used it to predict a set of training image IDs given test descriptions. We mapped the training image IDs to their corresponding training image ResNet features. We then computed the mean for each set of training image features. Thus, we matched each test description to an estimated image ResNet feature vector.

Finally, we fit a KNN with all of the test image ResNet features and used the estimated ResNet feature vectors to predict the top 20 test images.

1.3 Image Features to Bag of Words

This approach works quite similar to the Bag of Words to Image Features model but is in reverse order. We perform the same type of preprocessing on the training and testing descriptions to build a the description feature matrixes.

However, we begin by building a KNN regressor that fit the training image ResNet feature vectors. We used it to predict a set of training image IDs given test image ResNet features. We mapped the training image IDs to their corresponding training image descriptions. We then computed the mean for each set of training descriptions. Thus, we matched each test feature to an estimated description.

Finally, we fit a KNN with all of the estimated descriptions and used the test descriptions to predict the top 20 test images.

1.4 Word2Vec

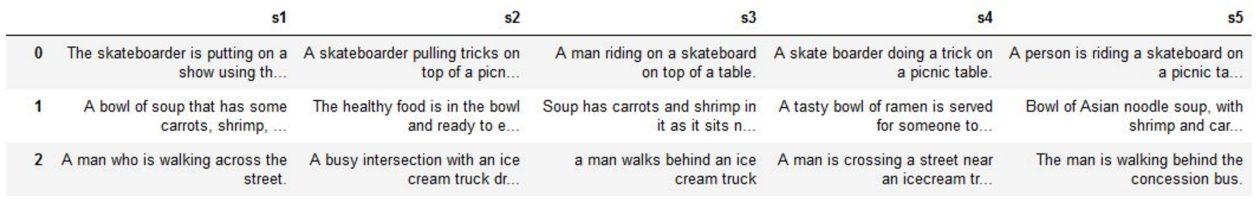
The Word2Vec model is based on word embeddings, which is a technique to translate a word into a numerical vector value. Word2Vec specifically encodes in the vector the contextual information of that word based on an input corpus model.

The corpus model uses a list of sentences use to train and build a model that learns what other list of words are commonly used and associated with a given word. For example, for the word "chocolate", it might be commonly associated with candy, ice cream, or dessert. Given a (large) list of sentences, the Word2Vec model will create a vector space based on all input sentences. Then, when feeding the model a single word, a single vector representation of that word in that space is outputted. So when passing a test sentence, one can find the vector representation by summing or averaging each word vector from that space.

This vectorization allows you to compare 2 sets of descriptions based on a distance since they can both be represented by a vector from the same space. We used this approach to map the test description to the closest training description. Once we had a training description, we used its corresponding feature vector, and then used this to find the nearest 20 test feature vectors to output.

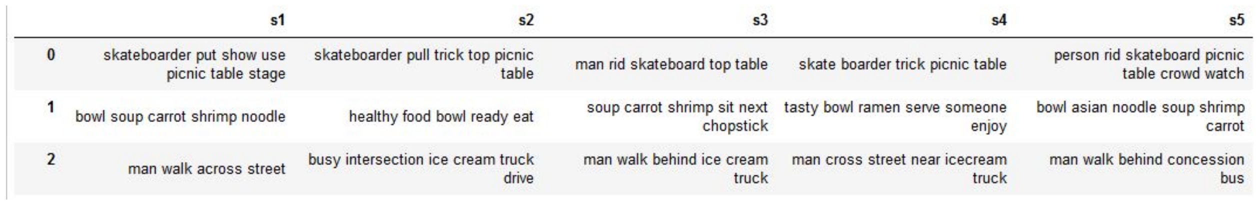
2. Experiment

For preprocessing our descriptions, we 1) removed stop words and punctuations 2) made our words all lower-cased 3) auto-corrected any misspellings 4) lemmatized the words e.g. from ran to run and quietly to quiet.



	s1	s2	s3	s4	s5
0	The skateboarder is putting on a show using th...	A skateboarder pulling tricks on top of a picn...	A man riding on a skateboard on top of a table.	A skate boarder doing a trick on a picnic table.	A person is riding a skateboard on a picnic ta...
1	A bowl of soup that has some carrots, shrimp, ...	The healthy food is in the bowl and ready to e...	Soup has carrots and shrimp in it as it sits n...	A tasty bowl of ramen is served for someone to...	Bowl of Asian noodle soup, with shrimp and car...
2	A man who is walking across the street.	A busy intersection with an ice cream truck dr...	a man walks behind an ice cream truck	A man is crossing a street near an icecream tr...	The man is walking behind the concession bus.

Figure 1: Original Training Descriptions



	s1	s2	s3	s4	s5
0	skateboarder put show use picnic table stage	skateboarder pull trick top picnic table	man rid skateboard top table	skate boarder trick picnic table	person rid skateboard picnic table crowd watch
1	bowl soup carrot shrimp noodle	healthy food bowl ready eat	soup carrot shrimp sit next chopstick	tasty bowl ramen serve someone enjoy	bowl asian noodle soup shrimp carrot
2	man walk across street	busy intersection ice cream truck drive	man walk behind ice cream truck	man cross street near icecream truck	man walk behind concession bus

Figure 2: Pre-processed Training Descriptions

2.1 Bag of Words for Descriptions to Tags

In this method, we first constructed a bag-of-word from the training description, and instead of recording a binary number, we calculated the TF-IDF of each word, using *TfidfTransformer()*, for all descriptions. We will call this matrix TF-IDF matrix, and we have one each for training and testing data. For the training data, the TF-IDF matrix has a dimension of 10000x3500, because here are 10000 descriptions and around 3500 unique words in the training descriptions. For the testing data, the TF-IDF matrix has a dimension of 2000x3500.

After computing the TF-IDF matrices, we made a bag-of words for the tags in both the training and testing data, using *CountVectorizer()*. If a tag is associated with an image, the tag has a "1" within the feature matrix. We will call this matrix the feature matrix. The feature matrix for training data has a dimension of 10000x80, whereas the one for testing data has a dimension of 2000x80. There are 2,000 testing images and 80 tags in total.

To predict the tags associated, we used a few algorithms, such as Support Vector Machine, Multi-Layer Perceptron etc. However, Logistic Regression and Random Forest turned out to be the best models. We fit the training TF-IDF matrix as X and training tag feature matrix as labels into the algorithms, and with our testing TF-IDF matrix, we predicted the testing tag feature matrix, which is a 2000x80 matrix indicating which tags are associated with respective description. Using KNN with k = 20, we determined the top 20 images that are ranked closest to the description provided.

2.2 Bag of Words to Image Features

For the Bag of Words to Image Features model, we performed several experiments to fine-tune its performance.

When experimenting with the Bag of Words model, we found that normalizing the description feature vectors resulted in better scores. The L1 Normalization is a good choice because it divides the occurrences of a word by the total number of words in the description. This type of normalization helps us evaluate the relative contribution of a word to the meaning of the image description. For example, if the the word `"skateboard"` appeared five times in a 25 word description, the L1 Norm gives `"skateboard"` a higher weight than most other words. Normalization helps to prevent overfitting as well.

Since we performed KNN to get a set of training image feature vectors for each test description, we tried several different *k* values to vary the amount of train image feature vectors we use to compute the estimated ResNet features.

2.3 Image Features to Bag of Words

Just as wto the Bag of Words to Image Features method, we also performed several experiments to fine tune the performance of this algorithm. We used the L1 distance to normalize our description feature vectors and tried several different *k* parameter values for our KNN to vary the amount of training descriptions we consider when computing an estimate description feature vector.

2.4 Word2Vec

The first step was to train our Word2Vec model with a corpus of sentences. We took all the descriptions and sentences in the training set and preprocessed to gather all the sentences in a list, from all the descriptions. We stripped out all the stopwords, punctuations, lower cased them, and did lemmatization. We used the gensim Word2Vec library and fed our sentences into it. The dimension for the output of each Word2Vec vector was 500. We started with 100 dimensions but increased it for better accuracy, but also kept a balance for speed.

Once we had a Word2Vec model, we were able to calculate a Word2Vec vector for each word passed to it using the library, and then normalize the vector. We then used a logistic regression classifiers on the mapping of test Word2Vec to train Word2Vec. To do this, we needed a Word2Vec for each training sentence (averaged overall the words in a sentence), and fit each sentence against the corresponding image id. We then created a Word2Vec vector for each testing description, averaging the vectors across entirely over the description. We used this as input to our logistic regression, which outputs a corresponding image id. With predicted id, we found its feature vector, and used KNN to find the top 20 nearest feature vectors by distance in the test image features, which map to a test image id.

In future competitions, other implementations we could change to improve the accuracy would be to instead taking the average over a sentence over a sentence, we could try concatenating a Word2Vec vector for each word in a sentence and then normalize sum. Also, we could try other classifiers like SVM from the test to train Word2Vec mapping. And perhaps more important, we could train on larger corpus available out there online.

2.5 Boosting Methods

Through our results, we noticed that the "Description to Tags with Random Forest" and "Image Features to Bag of Words" were our top performing algorithms. Accordingly, we used a boosting method to combine the results of the two weaker algorithms to produce a more accurate classifications. We did the following steps for boosting:

1. Compute the scores for the results of each algorithm
    a. For each test image, give the classes of predicted images `20 - r` points
    b. Where `r` is the ranked index of that image label

2. Sum up the scores of the two sets of results
    a. Thus, if the first algorithm ranked picture ***I*** in the *j*-th indexed position and the second algorithm ranked picture ***I*** in the *k*-th indexed position, the sum of the scores would give picture ***I*** a score of *j+k* points.
3. Sort the scored images from the sum of the two sets and gather the combined top 20

We did not add any further weights to the Boosting algorithm at this time since both the "Description to Tags with Random Forest" and "Image Features to Bag of Words" had similar accuracies to begin with.

3. Results

Some of our key results from experiments are listed below.

Method	Parameters	Score
(1) Bag of Words to Tags with Random Forest	N/A	0.24615
(2) Bags of Words to Tags with Logistic Regression	N/A	0.20111
(3) Bag of Words to Image Features with KNN	KNN with k=10	0.16285
(4) Image Features to Bag of Words with KNN	KNN with k=15	0.23212
(5) Word2Vec	Dim. = 500	0.09881
Boosting (1) and (4)		0.28164

Overall, we found that Description to Tags with a Random Forest Regressor worked best. We also observed that Image Features to Bag of Words was a better model than its reverse counterpart. This leads us to believe the relationship between image features is more significant than descriptions. Ultimately, we used a classic machine learning strategy of boosting to combine multiple classification algorithms.

4. Discussion

There are many strategies on how to find the relationship between the training and testing data, such as using bag of words to find predicted tags, using image features to find the predicted bag of words and Word2Vec. Based on the strategy we used, different algorithms are optimal. During our experiment, Random Forest worked best with our Bag of Words to tags, and KNN worked best with image features to Bag of Words. Therefore, we realised that it is important to experiment with the data to achieve better results. Experimenting with data also tells us more about the nature of the data. For example, if Random Forest and Logistic

Regression work better with Bag of Words to Tags, this shows that the relationship between descriptions and tags has a pattern of having multiple layers and folds instead of being clumpy.