CS5785 Fall Course Project Report

Build a large-scale image search engine

13th of December 2021
Cornell Tech, Fall 2021
Nathan Kallus

Jackie Peng, CS (jtp224)
Kexin Chen, ORIE (kc689)
Alex Legarda, ORIE (al926)

Abstract

In this project, we investigate how different machine learning methods can be used to build a large-scale image search engine. For the search engine, the input is natural language, which people usually use for describing or querying images, and the output is the relevant images. The main challenge of the project is to build a proper model to connect descriptions to relevant images. Following additional data processing, our team experimented with four different models including k-Nearest Neighbors, Neural Network, Ridge Regression, and Random Forest. The best performing algorithm is the Neural Network which received an accuracy of 23.383% on Kaggle.

Introduction

Our objective is to find the 20 most relevant images corresponding to a given text description. Also, we rank the 20 images according to relevance to the description, where the first image represents the picture that our model determines as most relevant to the description among the 20 pictures, and the last is what our model predicts to be the least relevant.

We have three different kinds of data: descriptions, images, and tags associated with each image.

Each description contains five sentences describing image content, which is used to produce the input to our model. For example, for the first description of the given training data, the five sentences are: 1. The skateboarder is putting on a show using the picnic table as his stage. 2. A skateboarder pulling tricks on top of a picnic table. 3. A man riding on a skateboard on top of a table. 4. A skateboarder doing a trick on a picnic table. 5. A person is riding a skateboard on a picnic table with a crowd watching.

The top 20 relevant images of each description are from the images dataset. The images are of size 224x224. There are in total 10,000 training images and 2,000 test images. Each image also comes with a set of tags. For each tag dataset, there are several tags indicating the category of human-labeled objects that appear in the image.

We convert the data into vectors so that we can use different machine learning models to make predictions from them. Images are vectorized using a pre-trained ResNet neural network, and text from descriptions and tags are converted to a vectorized form using a pre-trained word2vec language model.

The general steps we took to generate our predictions are as follows. First, we preprocess the data by converting descriptions, images, and tags to vectors using different methods, and then matching tag data with that of their corresponding images. Then, we scale the data before using it

to train a regression model. Parameters for models are selected using methods like k-fold cross validation and conducting repeated experiments using different training and validation dataset splits. From the resulting prediction output, we use a cosine distance measure to find the top 20 images in the dataset that are the most similar to the output from the prediction model. Details of each step in this process are given below.

Experiment

Preprocessing

Each of the descriptions we are given is in the form of sentences, which need to be post-processed after reading them in from their respective text files. We do this to reduce the amount of noise that is present in the data, as these descriptions are written by humans, and to try to extract only the most relevant data. The steps taken for post-processing are as follows: making all characters lowercase, removing non-words such as numbers and symbols, removing words that contain less than two letters, removing words that contain only consonants, removing stopwords, and lemmatizing. After the description text is post-processed, each word is transformed into a 300-dimensional vector using the pre-trained word2vec algorithm, and the vectorized words are averaged together to produce a final 300-dimensional vector representation of the entire text description.

To generate the feature vector representing each image in our model, we first transformed each image into a 1000-feature vector using the given pre-trained ResNet neural network. Then, we transformed the tags associated with each image into another vector. Each given tag includes a category, such as “animal” or “sports”, and a specifier, such as “zebra” or “snowboard”, and the two are separated by a colon. There are a total of twelve categories across the dataset, and using word2vec, each specifier word is transformed into a 300-dimensional vector. This 3600-dimensional vector representing the tags in an image is organized such that each tag category has 300 dimensions in the vector. Specifically, for each image’s tag vector, the first 300 dimensions represent the tag specifier associated with the category “accessory”, the next 300 dimensions represent the tag specifier associated with the category “animal”, and so on. If an image has at least one tag for a certain category, then the specifiers associated with that category are vectorized using word2vec, and the resulting vectors are averaged together and placed into the section of the tag vector for the image for the tag category. If any single specifier contains more than one word, we use only the first out of the two words to put into word2vec. If an image does not have any tags for a certain category, then the section of the tag vector for the image for that certain category is just set to a 300-dimensional vector of zeros. The resulting tag vector for the image is then appended to the ResNet feature vector for that image. After this process, each image is represented by a 4600-dimensional vector that incorporates both image data from ResNet and the given tags associated with the image.

As the image now has a very high-dimensional vector representation, to make the training process more efficient, we used PCA to reduce it down to 1500 dimensions for the neural network, random forest, and k-nearest neighbors, and 500 dimensions for ridge regression. This number was selected such that we were able to achieve a reasonable balance between the amount of time required to train the model, the overall accuracy of predictions, and having an accurate model that does not overfit the data.

We tried many other methods of preprocessing before we decided on this final method. When trying to post-process the text input, we tried removing words that appeared only once, but that ended up removing important information from certain descriptions. We also tried adding more instances of highly repeated words in a description to that description so that those words would have a higher weight in the average of the word vectors in the document, but this only lowered our prediction accuracy. We also tried training our own Gensim doc2vec model, which could potentially give a more meaningful vector representation of a document than simply averaging the vector representations of each word in the document together. However, this did not improve our prediction accuracy, likely because our dataset was not large enough to train this doc2vec model well.

Earlier in the process, we did not incorporate tags into our image data and instead used only the ResNet feature vector to represent each image. When using this, our prediction accuracy never exceeded 18%. In deciding how to handle tag specifiers that contained more than one word, we tried vectorizing just the second word out of the two words, or averaging the two words together. Upon careful inspection of the list of two-word tag specifiers, we saw that for most of them, the first word tended to be more indicative of the nature of the tag than the second word, and through testing were able to confirm this as using the first word tended to give us slightly better prediction accuracy than using the second word. Averaging the two words together resulted in lower prediction accuracy, likely because specifiers containing two words, such as “hot dog”, do not usually produce a meaningful result when the vector representations for each word are averaged together. Something else we tried for incorporating tags is based on a post we saw on Kaggle, which said that most tags can be easily predicted by using an image’s ResNet data alone. Using this information, we tried encoding only tag categories that were harder to predict from ResNet data. However, this resulted in lower prediction accuracy, so we decided to keep all the tag features in our image feature vector.

k-Nearest Neighbors

The first model we used is the k-Nearest Neighbors model. We used the KNN model on our preprocessed data to predict the best match image IDs for each description. We tried different K-values including 5, 10, 15, 20 and 25 to make sure the model is neither underfitting nor overfitting and to maximize our validation MAP score. However, the results we get from the KNN model are worse than we expected. All the Kaggle scores we received when using KNN model are less than 14%. The optimum K-value we found while testing is 15, and this model gave a score of 0.13206 on Kaggle.

Neural Network

Our best performing prediction model, which obtained a Kaggle score of 0.23383, is a neural network with one hidden layer, implemented using the Sequential class from Keras. Before passing data into this neural network, it was scaled to the range of 0 to 1 using a MinMaxScaler to help the neural network maximize prediction accuracy. The hidden layer contains 1,510 neurons and uses a ReLU activation function. The input layer contains 300 neurons, one for each dimension of the vector representing the description, and the output layer contains 1,500 neurons, one for each dimension of the PCA-reduced vector representing the image, and uses a sigmoid activation function. The activation functions were selected based on research, where multiple sources said that ReLU hidden layers were often the most effective, and a sigmoid activation function in the output layer scales output data to between 0 and 1, which matches our data scaling decision. Using more than one hidden layer did not improve results. The initial weights are taken from a uniform distribution, and this was selected through repeated experimentation. The learning rate of the model was reduced from the default value of 0.001 to 0.0001, which made a small improvement to the score during experimentation. This neural network is run for 200 epochs with a batch size of 120, and these parameters were selected through analysis of validation loss with each epoch through repeated experimentation. From the output of the neural network, which is still scaled between 0 to 1, the data is inversely transformed back to match the original input before scaling, and is then used for predictions.

Ridge Regression

We adapted the provided ridge regression model to work more effectively with our preprocessed data. First, we used PCA to reduce the dimensions of the expanded image feature vectors down to 500 dimensions. We found that using a higher number of dimensions for ridge regression caused prediction accuracy to decrease, likely due to overfitting. Then, we scaled the data with MinMaxScaler so that the magnitude of values of each feature are within the same range, which is a necessary requirement for a well-performing ridge regression model. By using k-fold cross validation, we found that the optimal alpha value for the model to be 20. The resulting accuracy score of this ridge regression model was 0.19573 on Kaggle.

Random Forest

Training a random forest model for this task was inefficient, due to the high dimensionality of the data. Limiting the number of features used to make splitting decisions improved runtime drastically. Using max_features='sqrt' lowered the runtime from around 90 minutes to around 2 minutes and a half while not losing any accuracy. Nonetheless, the score of 0.11593 for the model was quite low.

Making Predictions

After we predict an image vector from a given description, we need to find the top 20 most similar images based on a distance measure. Through repeated experimentation, we chose to use a cosine distance measure, which is typically used to determine the similarity between two documents. We found that this worked relatively well for vector representations of image data as well. Other distance measures we tried in the experimentation process included Euclidian, Manhattan, and Minkowski distance measures, but the cosine distance tended to give us better MAP scores.

Results

Prediction Algorithm	**Kaggle Score**
k-Nearest Neighbors	0.13206
Neural Network	0.23383
Ridge Regression	0.19573
Random Forest	0.11593

The model that performed best was the neural network, which received a Kaggle score of 0.23383. Ridge regression performed satisfactorily, with a resulting Kaggle score of 0.19573. The k-nearest neighbors and random forest models resulted in the least accurate predictions, both during experimentation and from their Kaggle scores, which were 0.13206 and 0.11593 respectively.