

CS 5785 FALL 2019 FINAL WRITE UP

ABSTRACT

Team Business Dogs used applied machine learning techniques to produce an image search engine. An iterative process was taken to find the best framework for the most accurate results. Many word processing methods, statistical models, and approaches were analyzed. The best of all the team's experiments were combined into the final algorithm. The team obtained a MAP@20 score of 0.61.

INTRODUCTION

The task set ahead was to create a search engine using statistical learning techniques. The objective of the machine learning algorithm was to be able to take sentence inputs and retrieve images. This technology is a competitor of other popular image search engines. It can provide users a seamless way to get images given a natural language query.

For this model, all inputs were five sentences long, and the output was a rank of the top 20 most relevant images. 10,000 training samples were given. Each sample had the following data:

- 224x244 JPG image
- Human generated tags
- ResNet feature vectors. Two were given:
 - 1,000-dimensional feature from classification layer (fc1000)
 - 2,048-dimensional feature from final convolutional layer (pool5)
- Five-sentence description

Testing data contained all the same available data but with some restrictions. Only 2,000 samples were given, and the five-sentence descriptions didn't have an image label. The task at hand was to match a five-sentence description to a test image. It was assumed there was not a one-to-one matching. Two descriptions could produce the same image.

METHOD

Please Note: The team refers to "images" as a combination of resnet features and tags. Actual image pixels were never used.

The final method is a Frankenstein machine that combines the best from all experimentation. The strongest tool was cross-validation. It allowed the team to judge what variables were adding accuracy. The

testing dataset was 10,000 samples. Five-fold cross-validation meant 2,000 samples were testing against 8,000 training samples. This gave an accurate indication of how well the algorithm would score on Kaggle. MAP@20 formula was applied to give the team a better idea of leaderboard rankings. Please note that the evaluation page of Kaggle was discovered to be wrong. The proper formula was discovered, from trial and error, to be:

$$MAP@20 = \frac{1}{1 + rank}$$

Where rank is the position (0-indexed) the correct image lies among the top 20 most relevant images.

Among all experimentation, Ridge regression reigned king. More advanced techniques gave good results but none compared to the high accuracy linear regression was able to produce.

Ridge¹ regression solves a regression model where the loss function is the linear least squares function and regularization is given by the l2-norm. The runtime is quick and allowed for the team to experiment quickly.

Ridge¹ regression was utilized using the following framework:

- train_images: Resnet fc1000 matrix + processed tags
- train_desc: Processed descriptions
- test_images: Test Resnet fc1000 matrix + processed tags
- pred_desc: Predicted description matrix

¹https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.Ridge.html

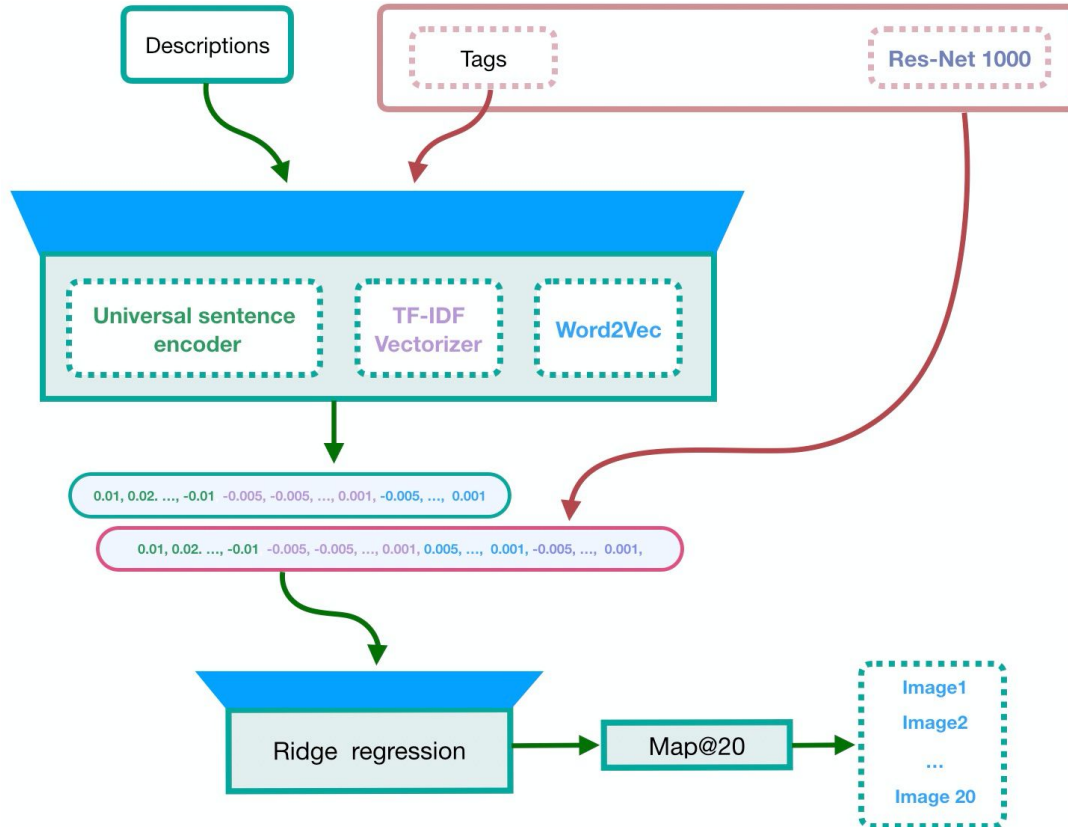


Figure 1: Final method overview

`Sklearn.linear_model.Ridge()`¹ was the module the team used. The coding of the `Ridge()` regressor is provided below as students and faculty are familiar with `sklearn`'s code format:

```
regr = Ridge()
regr.fit(train_images, train_desc)
pred_desc = regr.predict(test_images)
```

Please note that the final 2,048-dimensional feature matrix was omitted from the model. This was due to the nature of the final convolutional layer (pool5). The final convolutional layer is not the output for this kind of neural network. It still needs to be flattened, connected, and softmax'd. Removal of the final convolutional layer produced better results.

The greatest improvement of accuracy came from the data engineering of the words. The training/test

descriptions and training/test tags were given the same preprocessing. Please see all preprocessing steps below:

- 1) Stemming
- 2) Stop-word removal
- 3) Punctuation removal

The processing from document to vector was handled using the following three techniques:

- 1) TF-IDF² and then passed through PCA
- 2) Word2Vec³ values weighted by TF-IDF
- 3) Google's Universal Sentence Encoder-large⁴

²https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html

³<https://tfhub.dev/google/Wiki-words-500/2>

⁴<https://tfhub.dev/google/universal-sentence-encoder-large/5>

TF-IDF² stands for Term Frequency - Inverse Document Frequency. The concept is very similar to bag of words but word frequency count is logarithmic. This way high frequency words don't overwhelm the model. TF-IDF produces a large matrix and a dimension for every word. The team used PCA⁵ to reduce the dimensionality: to decrease runtime and to blend together word interactions. The Word2Vec³ model uses wikipedia for its word embeddings. Word2Vec produces a 500 dimensional vector and this was done to all words in each document. The vectors are then weighted averaged where the weight was derived from TF-IDF. Google's Universal Sentence Encoder codes text into high-dimensional vectors that can be used for text classification, semantic similarity, clustering and other natural language tasks. This three-step approach ensured that every last drop was milked from these words. Understand that each step was a different processing method done to the unprocessed sentences from the text files. All three word processing methods were combined to transform tags and sentences into matrices with thousands of dimensions.

Take note that the team compared image to description. The model outputs a predicted description matrix. Each description vector from the output was compared to the test description matrix. The best descriptions were found for each image. This information was then transformed so the top 20 images could be found for each description. Please see Figure 1 for an illustration of the team's overall architecture.

EXPERIMENT

Problem Approach

Experimentation began with exploring different ways of approaching the general problem. One of the first approaches was to treat the task as a classification problem. A processed train description vector plus a processed train image vector would score 1 if they were in the same row and 0 otherwise. The model would try different combinations of test description and test image vectors and label each with 1 or 0. Treating this as a classification task was useful since the more confident the model was about a classification, the

closer to 1 it would be. However, this approach was abandoned as it didn't produce good results. This was attributed to the fact that labeling was poorly done for examples of non-matches, i.e. images that didn't correspond to the description in the training set were being labeled as 0, which isn't representative of the reality of the problem since many images will not be exact matches but will have high similarity. If those samples are labeled as 0 it causes the model to *unlearn* what makes an image similar to a description. Moreover, there was no reasonable way to have good/sustainable labeling for the task.

It was determined the task would best be solved using mapping and regression. The model would take a vector input and output a vector. The output vector could then be compared to other vectors, and a distance formula could be used to produce a similarity ranking. At first, mapping was done from descriptions to images. This was producing promising results, however, the approach seemed to plateau at one point, getting us a maximum *MAP@20* score of 0.27 after a lot of tuning.

After some time we discovered that mapping from descriptions to image features wasn't the best approach, since we were essentially trying to map a human description to a highly specific resnet vector. For example, if the resnet vector contained two breeds of dogs as features and the description just mentioned "dog", then the model would be trying to predict the breed of the dog from only the word "dog". This would incorrectly affect the distance between two vectors that should be similar, since we are only interested in being as specific as the humans are in the descriptions. This led us to switch the mapping around, predicting less specific description vectors from more specific image vectors instead. Switching the mapping started producing better results right away, and after optimizing for this approach the score was almost doubled, getting us to our last *MAP@20* score of 0.61.

Text Preprocessing

As a general rule, we determined that preprocessing was needed on the descriptions and the tags, regardless of the approach. Also, we determined that there would be synonyms throughout the dataset that we wouldn't want to have as different features, so some kind of embedding would be needed.

⁵<https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html>

The success of our method was heavily dependent on finding the best vector representation for the text descriptions as well as the text tags associated with each picture. We ran a few experiments to find the best text to vector representation for the task. One was a model-free method to find the most versatile vector representation (or embedding). In this experiment we used a number of embedding models to transform the training descriptions and transform the training tags. Since there is a label which contains a one-to-one matching for each description and tag, we measured the cosine similarity between these vectors for each vector representation. The results are in Table 1.

Embedding	Average Rank	Evaluation Measure
TF-IDF	824.3	0.098
Gensim Doc2Vec (not pretrained)	2642.3	0.026
BERT (from “bert-as-service”)	3297.0	0.017
Glove	928.7	0.086
Tensorflow Universal Sentence Encoder	300.848	0.133

Table 1: Performance of Various Embeddings. Average Rank is average rank of label among 10,000 training examples. Evaluation Measure is the measure posted on Kaggle (which is not MAP@20)

Another experiment to find the best vector representation of the text features was to attempt to use each of these different embeddings in a linear model and measure performance in the information retrieval task. The results from this experiment are not shown for the sake of brevity and since they closely correlate with the rank-ordering in Table 1.

Testing Models

Our best model, as mentioned, was Ridge Regression. Ridge Regression naturally extends to the problem of multi-dimensional target variables which is what we

had in this case since we were predicting a vector representation of the description. We tuned the amount of regularization using cross validation.


Some of the models we did not end up using but did try include: Ordinary Least-Squares Linear Regression, Lasso Linear Regression, Random Forest Regression, Extra Trees Regression, Gradient Boosting Machine Regression, and Neural Networks. None of these models were able to come close to the results obtained by Ridge Regression either in accuracy or in speed (GBM came close in accuracy but was prohibitively slow). Particular effort was put into getting Neural Networks to work - different architecture, loss functions, and optimization methods were tried - but to no avail.

Finally, grid search was used in order to find the best combination of hyper-parameters and methods such as number of PCA components, whether to include 2-grams in the bag of words representation, and whether to apply stemming to words.

RESULTS AND ANALYSIS

Heavy experimentation proved to be key. Ridge regression was found to be best for the task at hand. The team tried many word processing methods to find the best combination. The winning combination was a mix of TF-IDF + PCA, Word2Vec with TF-IDF weights, and Google’s Universal Sentence Encoder. Our Kaggle score is a result of persistence!

Please see a screenshot of the team’s final score:

Team Members	Score ?
	0.60902

If more time allowed, more non-linear models would have been explored. The team’s best neural network resulted in a score of 0.45 MAP@20. With more experimentation, the neural network could potentially do better and maybe make use of the intermediate ResNet features.