AmIGod

Amigo Ding, Ben Chou, Brian Huang, and Steven Lin

***Abstract*—Search utility powered by text mining is commonly seen in real life implementations. However, there are a few systems, such as Instagram and Tumblr, where image-based searching or indexing is more suitable as well as necessary. In this project, utilizing information on images, we implemented three models, which are VSM, LM, and word2vec, for Tumblr Recommendation System, and we evaluated the performances of the models respectively.**

***Keywords*—word2vec, VSM, LM, tumblr, image to text, OCR, recommendation**

I. Introduction

A. Why Tumblr

The aim of this project is to apply searching techniques on a platform where images are dominant over other types of contents. The reason why we chose Tumblr as our target platform is that most of the posts on Tumblr are with photos and videos and usually they come with few words.

B. Tumblr Blogs

On Tumblr, the basic unit agent for any interaction or activity is **blog**. Blogs can like blogs, blogs can follow blogs, and blogs can leave notes on posts of other blogs. Every blog has a unique **blog name**.

C. Tumblr Posts

According to Tumblr API, there are 7 kinds of posts in total: Text, Photo, Quote, Link, Chat, Audio, Video, and Answer posts. They have common fields such as *id*, *tags*, etc; however, each of them has some unique fields. For example, in Audio posts, we can retrieve the *track_name* field where the title of the song may be held.

D. Our System: AmIGod

In this project, we implemented a search system which can be used to discover related blogs. There are 3 models involved in this system: **Vector Space Model**, **Language Model**, and **word2vec model**. When a blog name is given by a user, the system outputs 3 lists of blogs generated by the 3 models respectively.

E. Evaluation

Since it is costly to have our data labeled, we showed the performance of our system by giving some execution instances. We also elaborate several possibilities for future work in order to make the image-based search system more sound and more powerful.

II. AmIGod

A. Data Preprocessing

1) TumblrAgent: Since web API calls are expensive and time-consuming, we are barely able to do online-computing; that is, it is nearly impossible to fetch data from Tumblr with its API at the moment when needed. Therefore, before any IR-related techniques are implemented, we created a **TumblrAgent** which helps hold a **TumblrAgentCache** with 991 blogs and 176180 posts inside. TumblrAgent also serves as a wrapper for accessing data, providing many useful APIs such as *getBlogByName()*, which returns the TumblrBlog object in the cache corresponding to the given name or calling Tumblr API if no match found in cache.

2) VocabAgent: This is a very essential class for all of our 3 models because of these two methods: *extractTermsFromPhoto()* and *extractTermsFromPost()*. Firstly, *extractTermsFromPost()* helps retrieving all words occurred in the given TumblrPost. It helps strip the HTML tags, deal with different encodings, and remove meaningless symbols. Also, in this function, which fields to extract words from is determined by the type of the given TumblrPost. As for *extractTermsFromPhoto()*, we utilized a tool called **Caffe Demos** to convert images to a list of string tokens. Similar to TumblrAgent, VocabAgent stores some pre-fetched data due to the high cost of web API calls.

B. VSM

For each blog, get all the terms after the data preprocessing, and terms from different sources are treated equally. And then use the terms to form vector space model, and the weight of each term is decided by term frequency normalized by Okapi-BM25 multiplied by IDF. Finally, use cosine similarity to rank the most relative blogs with respect to the query blog.

C. LM

For each blog, record appearing terms' frequency of the blogs, then build the language model. We adopt naive-bayes uni-gram model. We count every term's probability in query, then we count the sum of the logarithm values. After a little computation, we can figure out the best response to the query (the one with biggest sum of the logarithm value). But how to decide the term's probability? The probability-deciding formula is referring to "Text Classification from Labeled and Unlabeled Documents using EM" (K Nigam, 2000).

For smoothing, we adopt additive smoothing because the corpus isn't very large.

D. word2vec model

Download the pre-trained word2vec vectors with dimension of 300, which are trained on part of Google News (about 100 billion words). There are a total of 3 million vectors, and each represent a word in English. We construct a 300-dimension vector for each blog by summing over all 300-dimension vector generated by word2vec model given all occurrences of words in that blog. The reason I use the sum of all vectors to represent the blog is because word2vec contains semantic meanings, which can make adding and subtracting operations of vectors meaningful. For example, vec('king') - vec('man') + vec('woman') = vec('queen'); we can clearly see that word2vec captures the semantic and syntactic meaning of words. Thus, it is reasonable to think a document can be the sum of all word vectors. Similar to the VSM model described above, cosine similarity is utilized to determine the relevance of two blogs, and the final list is then ranked in descending order.

III. Evaluation

A. Different kinds of blogs

The blogs without a specific subject are harder to predict precisely, because there are fewer deciding features, like blogs of private daily life. However, when there is a specific subject, like sport, music, and news, the models predict well. Also, if the majority of the posts in a blog is full of images with few captions, the performance is rather bad since the information is too scarce for any of our models.

B. Photo to text

In this task, we found that Caffe Demos can do well in some classifications, for example, food, animals. Whereas it performs bad on scenery (special ones), settings under dim light, which are more complicated and the features are harder to be extracted. In some cases, it generates words that are totally irrelevant to the picture, so when it provides some useful information to our prediction, it also create noises that potentially deteriorate the performance.

C. Improvement of 3 models

In our setting, all the words in word2vec have the same weight, which means that they are equally important, so it acts like a raw term frequency without normalization. However, the information each word contains are different, and the more frequent word should be of less importance. From this viewpoint, we can combine the tf-idf in VSM model with word2vec model, and then it should perform better.

IV. Conclusion

In this project, we implemented a search utility on an image-based platform, Tumblr, where text is usually insufficient for feature extraction. To enable image searching, we used **Caffe Demos** to transform images to a list of string tokens. Finally, we implemented three different models: Vector Space Model, Language Model, and word2vec model to produce three lists of related blogs respectively. In our evaluation, we found the performance of our system depends a lot on the subjects of posts. For example, more desirable results come from blogs about news, animals, food and sports, while blogs about daily lives tend to produce worse results.

V. Job Distribution

A. Amigo Ding: 25%

VSM model, report on VSM.

B. Ben Chou: 25%

word2vec model, report on word2vec and evaluation.

C. Brian Huang: 25%

Language model, report on LM.

D. Steven Lin: 25%

data processing, wrapper classes, models integration and most of the report.