**武汉大学计算机学院**

**本科生课程设计报告**

**Use the enterprise annual report to build an industry dictionary and analyze the upstream and downstream of the enterprise**

专 业 名 称 ：软件工程

课 程 名 称 ：商务智能

指 导 教 师 ：朱卫平 副教授

学 生 学 号 ：2016302580203

学 生 姓 名 ：曹子轩

二○一八年11月

Content

[1. Background](#_Toc27998_WPSOffice_Level1) [3](#_Toc27998_WPSOffice_Level1)

[2. Requirement Analysis](#_Toc14547_WPSOffice_Level1) [4](#_Toc14547_WPSOffice_Level1)

[3. Solution](#_Toc7056_WPSOffice_Level1) [4](#_Toc7056_WPSOffice_Level1)

[4. Running Environment And Dependencies](#_Toc6622_WPSOffice_Level1) [4](#_Toc6622_WPSOffice_Level1)

[5. Technology Used](#_Toc5903_WPSOffice_Level1) [5](#_Toc5903_WPSOffice_Level1)

[6. Data Set](#_Toc13341_WPSOffice_Level1) [6](#_Toc13341_WPSOffice_Level1)

[7. Experiment](#_Toc26689_WPSOffice_Level1) [6](#_Toc26689_WPSOffice_Level1)

[8. Key Codes](#_Toc3676_WPSOffice_Level1) [8](#_Toc3676_WPSOffice_Level1)

[9. Details And Problems](#_Toc26411_WPSOffice_Level1) [14](#_Toc26411_WPSOffice_Level1)

[10. How to see results](#_Toc843_WPSOffice_Level1) [15](#_Toc843_WPSOffice_Level1)

[11. Whole Process](#_Toc25462_WPSOffice_Level1) [18](#_Toc25462_WPSOffice_Level1)

1. **Background**

Listed companies will publish their quarterly and annual company reports on time to disclose the company's production and operation status. But these reports are very lengthy, and it is very difficult for humans to extract effective information from them. Therefore, in the near future, using data mining methods to grasp the company's production and operation status is a very popular direction.

Most companies have their own upstream and downstream industries. The state of these industries has an important impact on the company's situation. Therefore, extracting this information from the company's annual report is of great significance for better understanding and forecasting the production and operation of listed companies. The extracted information can provide an important reference for our more in-depth analysis.

Through the search, I found that there are already some websites on the market that analyze the upstream and downstream of the industry([产业价值链咨询平台](http://ic.tpex.org.tw/)). However, most of these sites are analyzed for the entire industry, and there is a lack of analysis of the company's granularity. Moreover, most of their extraction uses manual extraction and keyword analysis, which lacks efficiency and accuracy. Therefore, I consider using some of the knowledge I have learned in the BI course, text mining the company's annual report, using support vector machines and neural networks to analyze key statements in the company's annual report, establish an industry dictionary containing all industries and achieve The upstream and downstream extraction of the industry in the company's annual report.



图 1 产业价值链咨询平台

1. **Requirement Analysis**

The entire system contains two functions. For the input vocabulary, the system must be able to determine whether the vocabulary can be considered to represent an industry. For the input document, the system outputs the upstream, downstream and midstream industries involved in the company's annual report.

First, we need to collect relevant data. The corpus includes a news corpus (which helps us collect industry vocabulary) and a corporate annual corpus.

After that, we need to manually label the data, whether the data is an industry vocabulary, and whether it represents the upstream downstream or midstream vocabulary in the company annual report.

Then, we train through machine learning algorithms and neural networks to automatically label all the data.

Finally, visualize the results.

1. **Solution**
2. With a corpus, use word embedding to train and map words into vector space as the basis for subsequent calculations.
3. Use the support vector machine to train the annotated corpus.
4. Use LSTM to determine if each vocabulary in a sentence in the document represents an upstream downstream or midstream.
5. **Running Environment And Dependencies**

**Python 3.6**: Python is a popular computer language in machine learning and NLP for its easy grammar and rich packages. All the code are written via python.

**Django**: Django is a free and open-source web framework, written in Python, which follows the model-view-template (MVT) architectural pattern. It is maintained by the Django Software Foundation (DSF), an independent organization established as a non-profit. Actually, this project does not necessarily use Django. But considering Django provides us very convenient api to operate database and build a web site, we use it to improve our efficiency.

**Numpy**: Numpy is the fundamental package for scientific computing with Python.

jieba:

**Tensorflow**: TensorFlow is an open-source software library for dataflow programming across a range of tasks. It is a symbolic math library, and is also used for machine learning applications such as neural networks.

**Jieba**: It is a professional Chinese word segmentation tool.

1. **Technology Used**

**Word2vec**: Word2vec is a group of related models that are used to produce word embeddings. These models are shallow, two-layer neural networks that are trained to reconstruct linguistic contexts of words. Word2vec takes as its input a large corpus of text and produces a vector space, typically of several hundred dimensions, with each unique word in the corpus being assigned a corresponding vector in the space. Word vectors are positioned in the vector space such that words that share common contexts in the corpus are located in close proximity to one another in the space.

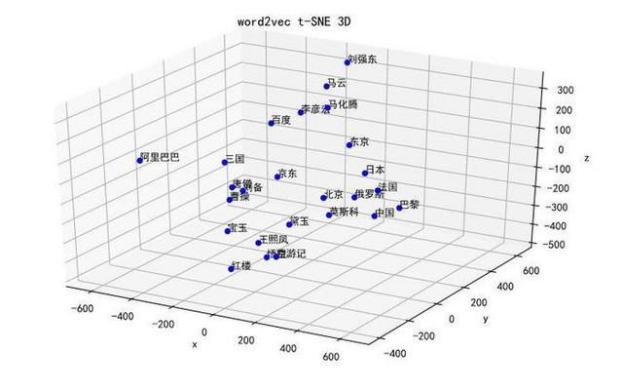
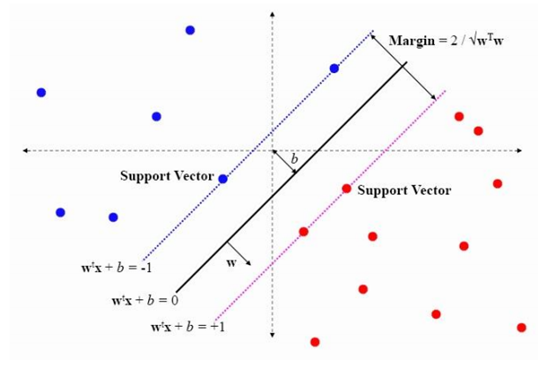


图 2 SVM 图 3 word2vec

**SVM**: In machine learning, support vector machines (SVMs, also support vector networks) are supervised learning models with associated learning algorithms that analyze data used for classification and regression analysis. Given a set of training examples, each marked as belonging to one or the other of two categories, an SVM training algorithm builds a model that assigns new examples to one category or the other, making it a non-probabilistic binary linear classifier (although methods such as Platt scaling exist to use SVM in a probabilistic classification setting). An SVM model is a representation of the examples as points in space, mapped so that the examples of the separate categories are divided by a clear gap that is as wide as possible. New examples are then mapped into that same space and predicted to belong to a category based on which side of the gap they fall.

**LSTM**: Long short-term memory (LSTM) units are units of a recurrent neural network (RNN). An RNN composed of LSTM units is often called an LSTM network. A common LSTM unit is composed of a cell, an input gate, an output gate and a forget gate. The cell remembers values over arbitrary time intervals and the three gates regulate the flow of information into and out of the cell.LSTM networks are well-suited to classifying, processing and making predictions based on time series data, since there can be lags of unknown duration between important events in a time series.

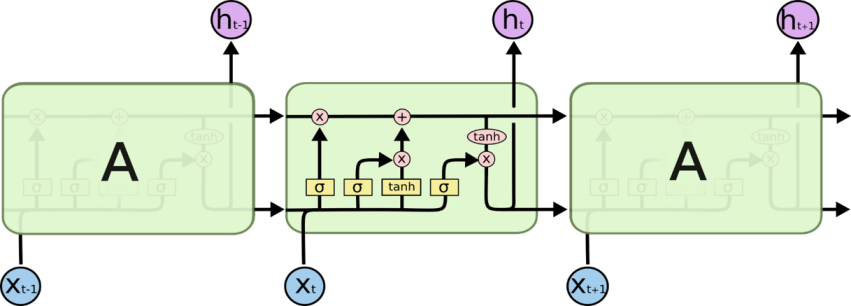
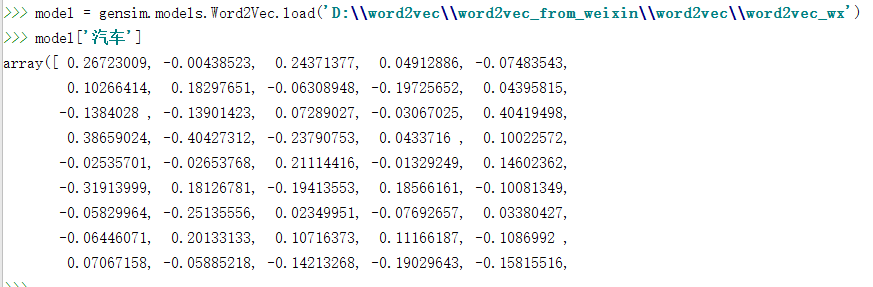


图 4 LSTM cell

1. **Data Set**

**Sohu news data**: The data can be downloaded in this website

**Listed company annual reports**: All the reports are collected in [EastMoney](http://www.eastmoney.com/). This work was done by the seniors of my laboratory and belongs to a branch of Professor Han Bo's research.

1. **Experiment**
2. First, we need to train word to vector. The code can be seen in function train\_data\_build() and train\_data(). Here are some examples of our vector. 
3. Select “行业” as the key word and extract the words in the sentence before "行业". Because we want to build an industry dictionary, we ignore the words that are not used for nouns. After that, we manually label all the industry vocabulary, and the size of the labeled data set is 1000. The code can be seen in function buildData()
4. Using the word vectors and annotations of different words as input, use the support vector machine for training and save the training results. All data is annotated using the results of the training. The following figure is a graphical representation of the accuracy and loss of the support vector machine. It can be seen that as the training progresses, the accuracy rate continues to increase and the loss continues to decrease. The code can be seen in SVM().

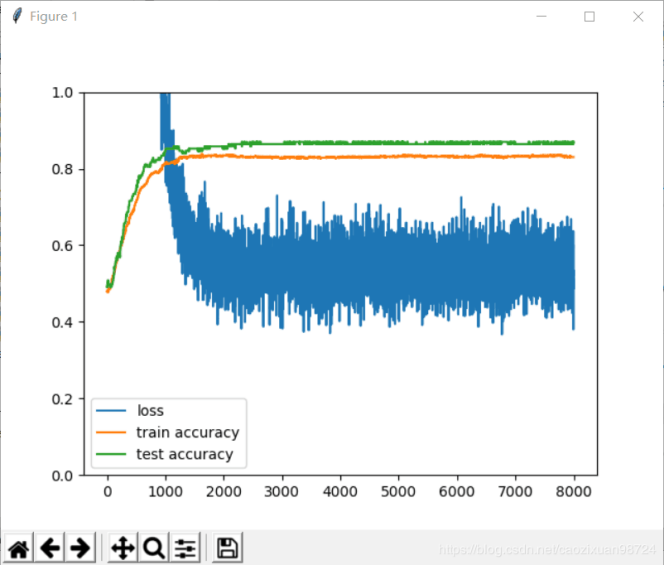


图 5 SVM accuracy and loss

(4) Now we can look at all the industry vocabulary we have collected and we can judge whether

a new word is an industry vocabulary. The process can be seen below.

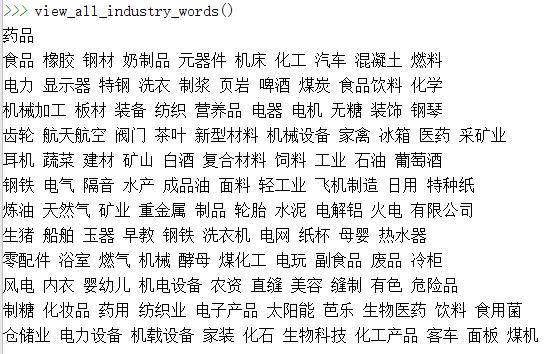


图 6view all words(can't display completely)

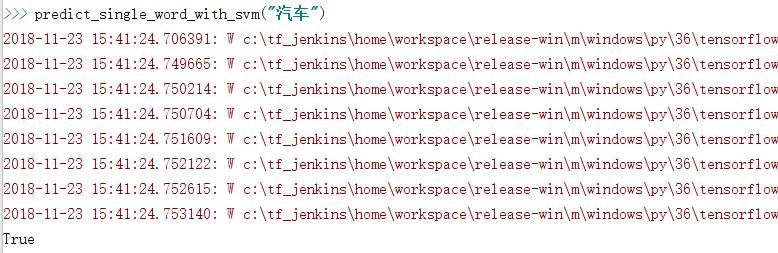


图 7 judge a word

(5) Although we have an industry dictionary, we can't simply match words with an industry dictionary. Because even if we use the keywords such as upstream and downstream to determine the statement containing information, it is difficult to determine whether the industry vocabulary that appears in it actually represents the upstream or downstream of the enterprise. Therefore, in order to improve the accuracy, we use LSTM for training and hope to improve the accuracy by combining context information. We extract sentences containing keywords from the company's annual report, manually annotate each sentence, mark about 1000 sentences, then train and save the training results. Use the training results to label all sentences in the database. This part of the work refers to some of the code on github, especially the paper and code of the intellectual recognition of Ms. Liu Zhiyuan from Tsinghua University. The training process and the result can be seen below. The code can also be seen in model Model and SA

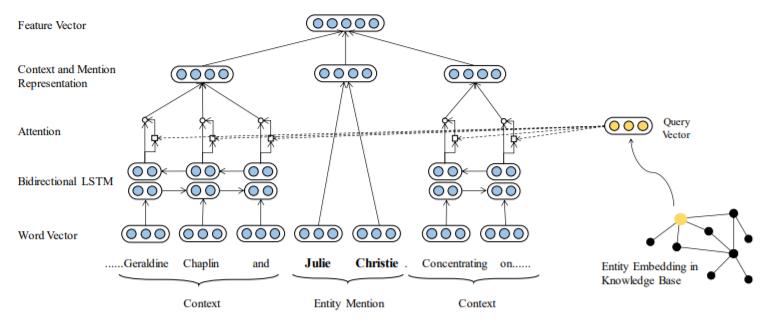


图 8 Entity Recognition Framework

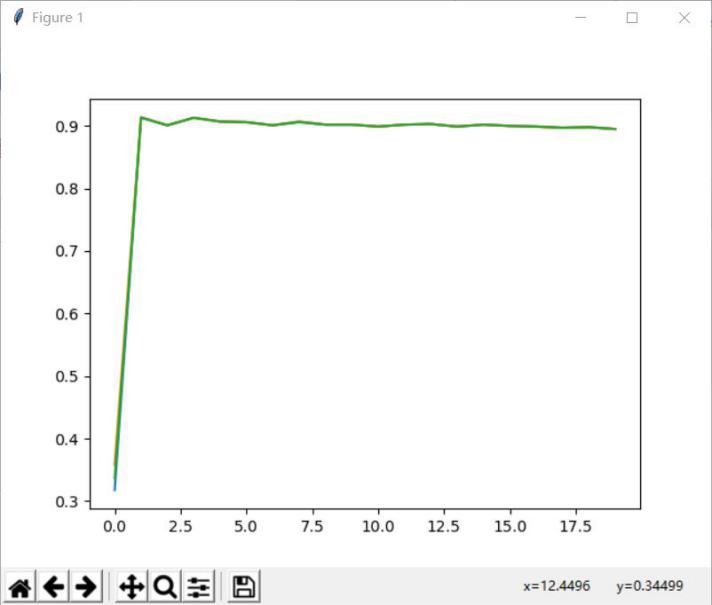


图 9 LSTM training process

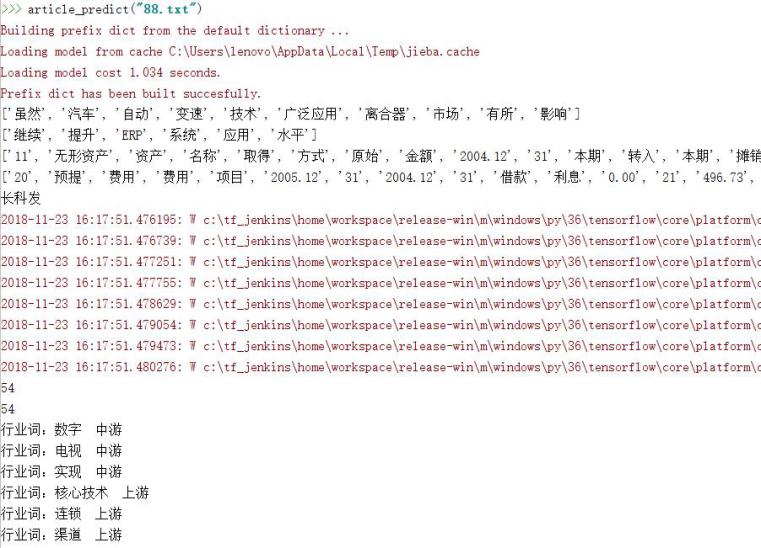


图 10 analyze a report

1. **Key Codes**

**def** train\_data\_build():  
 file = **r'F:\train\_data.txt'** names = file\_name(**'F:\\data'**)  
 **for** name **in** names:  
 f = open(name, errors=**'ignore'**)  
 st = f.read()  
 **with** open(file, **'a+'**) **as** f:  
 seg\_list = jieba.cut(st, cut\_all=**False**)  
 f.write(**" "**.join(seg\_list))  
 f.write(**'\n'**)  
 f.close()  
  
  
**def** train\_data():  
 **from** gensim.models **import** word2vec  
 sentences = word2vec.Text8Corpus(**'F:\\train\_data.txt'**)  
 model = word2vec.Word2Vec(sentences, size=50)  
 model.save(**'word2vec\_model'**)

**def** buildData():  
 rule = **r'(.\*)行业'** compile\_name = re.compile(rule, re.M)  
 names = file\_name(**'E:\\temp\_data\\tmp'**)  
 **for** name **in** names:  
 f = open(name, errors=**'ignore'**)  
 st = f.read()  
 res\_name = compile\_name.findall(st)  
 **for** sentence **in** res\_name:  
 seg\_list = jieba.lcut(sentence, cut\_all=**False**)  
 word = seg\_list[len(seg\_list) - 2]  
 **if** len(word) <= 1:  
 **continue** values = pseg.cut(word)  
 flag\_word = **True  
 for** value, flag **in** values:  
 **if** flag == **'n'**:  
 **continue  
 else**:  
 flag\_word = **False  
 if** flag\_word:  
 Dictionary.objects.get\_or\_create(name=word)

**def** SVM():  
 sess = tf.Session()  
 words = Divided.objects.all()  
 model = gensim.models.Word2Vec.load(**'D:\\word2vec\\word2vec\_from\_weixin\\word2vec\\word2vec\_wx'**)  
 x\_vals = np.array([model[word.name].tolist() **for** word **in** words])  
 y\_vals = np.array([1 **if** word.is\_industry **else** -1 **for** word **in** words])  
 train\_indices = np.random.choice(len(x\_vals), round(len(x\_vals) \* 0.8), replace=**False**)  
 test\_indices = np.array(list(set(range(len(x\_vals))) - set(train\_indices)))  
 x\_vals\_train = x\_vals[train\_indices]  
 x\_vals\_test = x\_vals[test\_indices]  
 y\_vals\_train = y\_vals[train\_indices]  
 y\_vals\_test = y\_vals[test\_indices]  
 *# 批训练中批的大小* batch\_size = 100  
 x\_data = tf.placeholder(shape=[**None**, 256], dtype=tf.float32)  
 y\_target = tf.placeholder(shape=[**None**, 1], dtype=tf.float32)  
 W = tf.Variable(tf.random\_normal(shape=[256, 1]))  
 b = tf.Variable(tf.random\_normal(shape=[1, 1]))  
 *# 定义损失函数* model\_output = tf.matmul(x\_data, W) + b  
 l2\_norm = tf.reduce\_sum(tf.square(W))  
 *# 软正则化参数* alpha = tf.constant([0.1])  
 *# 定义损失函数* classification\_term = tf.reduce\_mean(tf.maximum(0., 1. - model\_output \* y\_target))  
 loss = classification\_term + alpha \* l2\_norm  
 *# 输出* prediction = tf.sign(model\_output)  
 accuracy = tf.reduce\_mean(tf.cast(tf.equal(prediction, y\_target), tf.float32))  
 train\_step = tf.train.GradientDescentOptimizer(0.01).minimize(loss)  
 *# saver = tf.train.Saver()  
 # 开始训练* sess.run(tf.global\_variables\_initializer())  
 loss\_vec = []  
 train\_accuracy = []  
 test\_accuracy = []  
 **for** i **in** range(8000):  
 rand\_index = np.random.choice(len(x\_vals\_train), size=batch\_size)  
 rand\_x = x\_vals\_train[rand\_index]  
 rand\_y = np.transpose([y\_vals\_train[rand\_index]])  
 sess.run(train\_step, feed\_dict={x\_data: rand\_x, y\_target: rand\_y})  
 temp\_loss = sess.run(loss, feed\_dict={x\_data: rand\_x, y\_target: rand\_y})  
 loss\_vec.append(temp\_loss)  
 train\_acc\_temp = sess.run(accuracy, feed\_dict={x\_data: x\_vals\_train, y\_target: np.transpose([y\_vals\_train])})  
 train\_accuracy.append(train\_acc\_temp)  
 test\_acc\_temp = sess.run(accuracy, feed\_dict={x\_data: x\_vals\_test, y\_target: np.transpose([y\_vals\_test])})  
 test\_accuracy.append(test\_acc\_temp)  
 **if** (i + 1) % 100 == 0:  
 print(**'Step #'** + str(i + 1) + **' W = '** + str(sess.run(W)) + **'b = '** + str(sess.run(b)))  
 print(**'Loss = '** + str(test\_acc\_temp))  
 *# saver.save(sess, "./model/model.ckpt")* print(train\_accuracy)  
 print(test\_accuracy)  
 plt.plot(loss\_vec)  
 plt.plot(train\_accuracy)  
 plt.plot(test\_accuracy)  
 plt.legend([**'loss'**, **'train accuracy'**, **'test accuracy'**])  
 plt.ylim(0., 1.)  
 plt.show()

**class** Model(object):  
 **def** \_\_init\_\_(self, name):  
 self.type\_size = 2  
 self.word\_size = 256  
 self.lstm\_size = 100  
 *# self.transe\_size = 100* self.dev = 0.01  
 self.hidden\_layer = 50  
 self.window = 5  
 self.scope = **"root\_train\_second" if** name == **"KA+D" else "root"** self.predict()  
 self.saver = tf.train.Saver(max\_to\_keep=100)  
 self.initializer = tf.global\_variables\_initializer()  
  
 **def** entity(self):  
 self.entity\_in = tf.placeholder(tf.float32, [**None**, self.word\_size])  
 self.batch\_size = tf.shape(self.entity\_in)[0]  
 self.kprob = tf.placeholder(tf.float32)  
 entity\_drop = tf.nn.dropout(self.entity\_in, self.kprob)  
 **return** entity\_drop  
  
 **def** attention(self):  
 *# this method will be overrided by derived classes* **pass  
  
 def** context(self):  
 *# from middle to side* self.left\_in = [tf.placeholder(tf.float32, [**None**, self.word\_size]) \  
 **for** \_ **in** range(self.window)]  
 self.right\_in = [tf.placeholder(tf.float32, [**None**, self.word\_size]) \  
 **for** \_ **in** range(self.window)]  
  
 *# from side to middle* self.left\_in\_rev = [self.left\_in[self.window - 1 - i] **for** i **in** range(self.window)]  
 self.right\_in\_rev = [self.right\_in[self.window - 1 - i] **for** i **in** range(self.window)]  
  
 left\_middle\_lstm = tf.nn.rnn\_cell.LSTMCell(self.lstm\_size)  
 right\_middle\_lstm = tf.nn.rnn\_cell.LSTMCell(self.lstm\_size)  
 left\_side\_lstm = tf.nn.rnn\_cell.LSTMCell(self.lstm\_size)  
 right\_side\_lstm = tf.nn.rnn\_cell.LSTMCell(self.lstm\_size)  
  
 **with** tf.variable\_scope(self.scope):  
 **with** tf.variable\_scope(**'lstm'**):  
 *# from side to middle* left\_out\_rev, \_ = tf.nn.static\_rnn(left\_middle\_lstm, self.left\_in\_rev, dtype=tf.float32)  
 **with** tf.variable\_scope(**'lstm'**, reuse=**True**):  
 *# from side to middle* right\_out\_rev, \_ = tf.nn.static\_rnn(right\_middle\_lstm, self.right\_in\_rev, dtype=tf.float32)  
  
 *# from middle to side* left\_out, \_ = tf.nn.static\_rnn(left\_side\_lstm, self.left\_in, dtype=tf.float32)  
 right\_out, \_ = tf.nn.static\_rnn(right\_side\_lstm, self.right\_in, dtype=tf.float32)  
  
 self.left\_att\_in = [tf.concat([left\_out[i], left\_out\_rev[self.window - 1 - i]], 1) \  
 **for** i **in** range(self.window)]  
 self.right\_att\_in = [tf.concat([right\_out[i], right\_out\_rev[self.window - 1 - i]], 1) \  
 **for** i **in** range(self.window)]  
  
 left\_att, right\_att = self.attention()  
  
 left\_weighted = reduce(tf.add,  
 [self.left\_att\_in[i] \* left\_att[i] **for** i **in** range(self.window)])  
 right\_weighted = reduce(tf.add,  
 [self.right\_att\_in[i] \* right\_att[i] **for** i **in** range(self.window)])  
  
 left\_all = reduce(tf.add, [left\_att[i] **for** i **in** range(self.window)])  
 right\_all = reduce(tf.add, [right\_att[i] **for** i **in** range(self.window)])  
  
 **return** tf.concat([left\_weighted / left\_all, right\_weighted / right\_all], 1)  
  
 **def** predict(self):  
 *# this method will be overrided by derived classes* **pass  
  
 def** fdict(self, now, size, interval, \_entity, \_context, \_label):  
 *# this method will be overrided by derived classes* **pass  
  
 def** mag(self, matrix):  
 **return** tf.reduce\_sum(tf.pow(matrix, 2))  
  
 **def** cross\_entropy(self, predicted, truth):  
 **return** -tf.reduce\_sum(truth \* tf.log(predicted + 1e-10)) \  
 - tf.reduce\_sum((1 - truth) \* tf.log(1 - predicted + 1e-10))  
  
  
**class** SA(Model):  
 **def** attention(self):  
 W1 = tf.Variable(tf.random\_normal([self.lstm\_size \* 2, self.hidden\_layer], stddev=self.dev))  
 W2 = tf.Variable(tf.random\_normal([self.hidden\_layer, 1], stddev=self.dev))  
  
 left\_att = [tf.exp(tf.matmul(tf.tanh(tf.matmul(self.left\_att\_in[i], W1)), W2)) \  
 **for** i **in** range(self.window)]  
 right\_att = [tf.exp(tf.matmul(tf.tanh(tf.matmul(self.right\_att\_in[i], W1)), W2)) \  
 **for** i **in** range(self.window)]  
  
 **return** (left\_att, right\_att)  
  
 **def** predict(self):  
 x = tf.concat([self.entity(), self.context()], 1)  
  
 W = tf.Variable(tf.random\_normal([self.word\_size + self.lstm\_size \* 4, self.type\_size],  
 stddev=self.dev))  
 self.t = tf.nn.sigmoid(tf.matmul(x, W))  
 self.t\_ = tf.placeholder(tf.float32, [**None**, self.type\_size])  
  
 self.loss = self.cross\_entropy(self.t, self.t\_)  
 self.train = tf.train.AdamOptimizer(0.005).minimize(self.loss)  
  
 **def** fdict(self, now, size, interval, \_entity, \_context, \_label):  
 fd = {}  
 new\_size = int(size / interval)  
  
 ent = np.zeros([new\_size, self.word\_size])  
 lab = np.zeros([new\_size, self.type\_size])  
 **for** i **in** range(new\_size):  
 vec = \_entity[now + i \* interval]  
 ent[i] = vec  
 lab[i] = \_label[now + i \* interval]  
 fd[self.entity\_in] = ent  
 fd[self.t\_] = lab  
  
 **for** j **in** range(self.window):  
 left\_con = np.zeros([new\_size, self.word\_size])  
 right\_con = np.zeros([new\_size, self.word\_size])  
 **for** i **in** range(new\_size):  
 left\_con[i, :] = \_context[now + i \* interval][2 \* j]  
 right\_con[i, :] = \_context[now + i \* interval][2 \* j + 1]  
 fd[self.left\_in[j]] = left\_con  
 fd[self.right\_in[j]] = right\_con  
  
 **return** fd

1. **Details And Problems**

In the process of preprocessing the data, we removed the stop words that have no practical

meaning. At the same time, in the process of establishing the industry dictionary, we removed

the words with the part of speech as adjectives or verbs.

In the process of training the word vector, I encountered a lot of problems. The vocabulary that

needs to be trained is too large, the personal computer can't run the program well, and the

computer often gets stuck. However, the results obtained using small samples are not well

classified. Therefore, in subsequent experiments, the word vector training results I used are the

results of open training on the Internet. Detailed information about the results of this training

can be found in the source code ReadMe.

Specific methods for collecting data from corporate annual reports: use keywords such as “上

游” and “下游” to locate key statements. Then take 5 words in the upper and lower parts of

the keyword. (The specific quantity can be changed according to the situation)

When using the support vector machine for classification, the initial effect is not very satisfactory.

The reason is that the difference between the positive and negative examples is too large (most

words are not industry vocabulary). After deleting some counterexamples, the prediction effect is

greatly improved.

When I first started using neural network training, I wanted to divide the industry vocabulary into

two categories, industry terminology and industry non-terminal word. For example, the original

text wants to express "汽车制造", I don't want to regard it as "汽车" and "制造" two industry

vocabulary. But it turns out that this effect is ideally distributed, so I think of all the industry

vocabulary.

In this experiment, the industry dictionary obtained by using the support vector machine has

better effect and higher accuracy. But the results of using neural network training are not ideal.

On the one hand, my training samples are relatively small, on the other hand, there are still many

flaws in the preprocessing of data, and there is too little valuable data. The specific reasons need

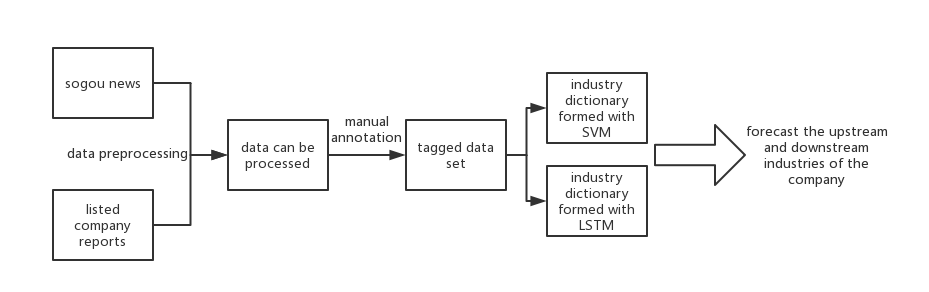
to be explored later.

Because of the rush of time, there are many normative problems in the specific implementation of the code. The coupling of functions is relatively high, and there are some redundant codes. As you can see, I have defined some duplicate tables in the database, mainly because I can distinguish between training data and test data.

When I write the code, I only think about implementing the function, and I don't pay attention to the optimization problem of efficiency. The current training, loading is very time consuming and not practical enough.

1. **How to see results**

Here is the whole idea of the experiment.

****

The running environment of the whole project is relatively complicated. The name and version of the specific required library are described in detail in the **requirement.txt** file in the source code. In order to facilitate the operation, I deployed a project that can be run on the server side.

The username is **root**, password is **Caozixuan98724** and host is **47.107.70.71**

You can use **Xshell** to login my server.

Then cd /BusinessInteligenceFinalWork/IndustryLink/

Input: python3.6 manage.py shell

Input:**from industry.views import SVM, train, predict\_single\_word\_with\_svm, article\_predict, view\_all\_industry\_words\_SVM, view\_all\_industry\_words\_lstm, print\_data\_SVM**

Then, you can try these methods.

These methods can be divided into two categories. One is based on SVM and the other is based on LSTM.

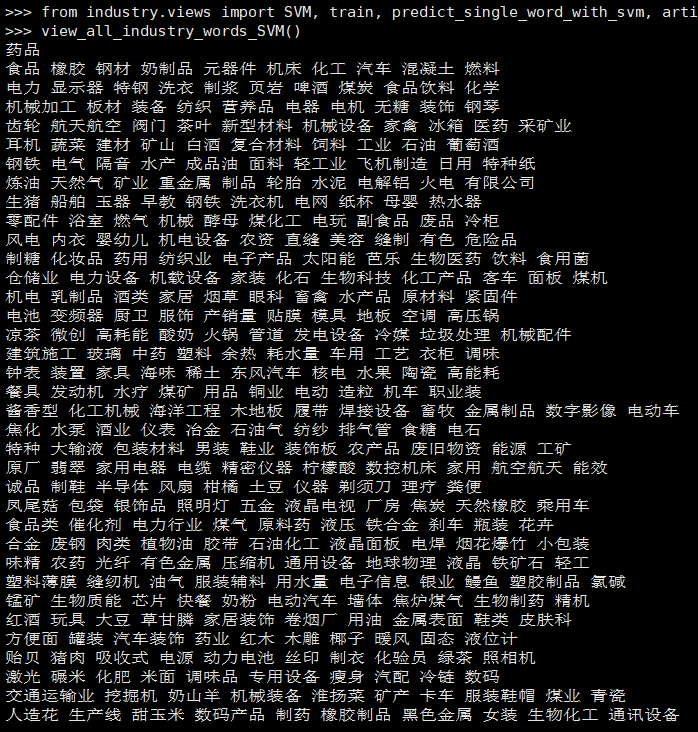
First, SVM based methods.

**SVM() # the method to train SVM model**

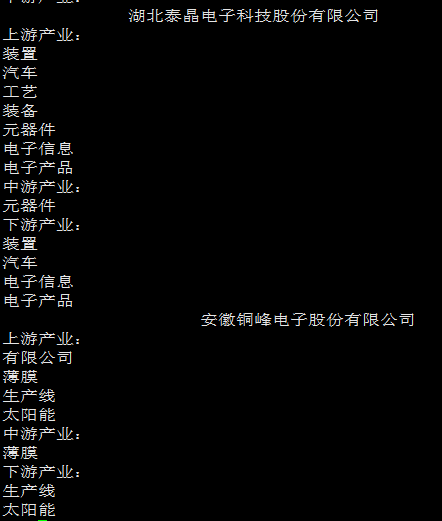
**predict\_single\_word\_with\_svm(“汽车”) # the method to predict single word using SVM**

These methods run well on my personal computer, but I meet difficulty running them on my server. Because the model is over 1G and the memory can not load it.

**view\_all\_industry\_SVM()**



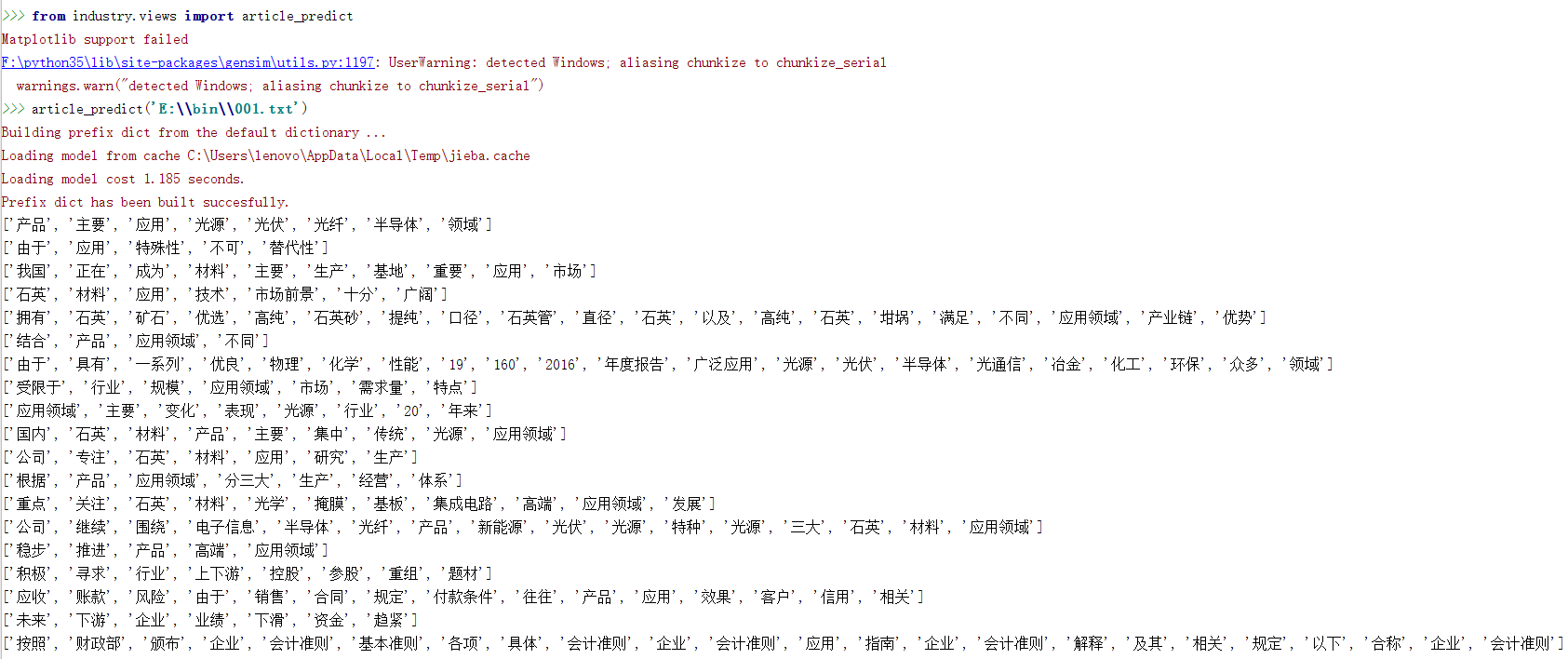
**print\_data\_SVM() # print data that collected with SVM**

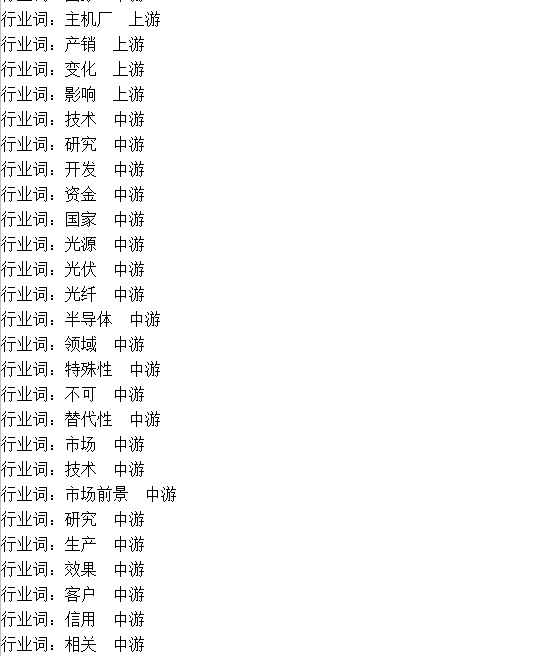


Then, LSTM based methods.

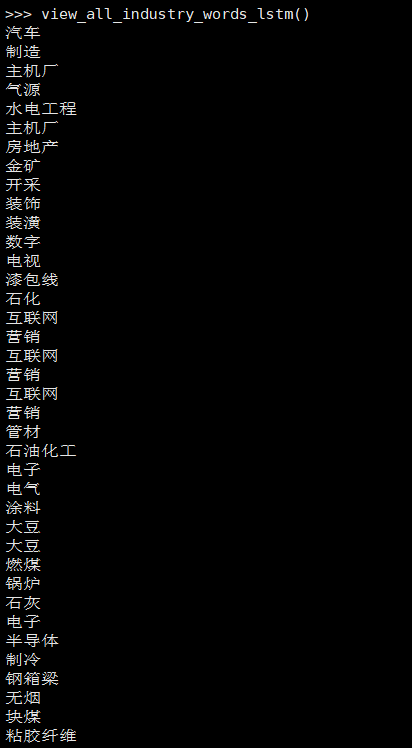
train() **# the method to train SVM model**

article\_predict(“1.txt”)





**view\_all\_industry\_words\_lstm()**



1. **Whole Process**

The whole process of the experiment.(You should change data path to your own path.

1. SVM

**getXMLContent() # get news content from sogou news data set**

**buildData() # build data set and then tag it**

**delete\_word\_SVM() # delete some nonsense word. There are also some other similar methods**

**SVM() # train SVM model**

**predict\_word() # use SVM model to predict all words**

1. LSTM

**collectSentence() # choose useful sentences that will be used in training.**

**train() # train LSTM model**