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

指导教师一：朱卫平 副教授

学生学号　：2016302580203

学生姓名　：曹子轩

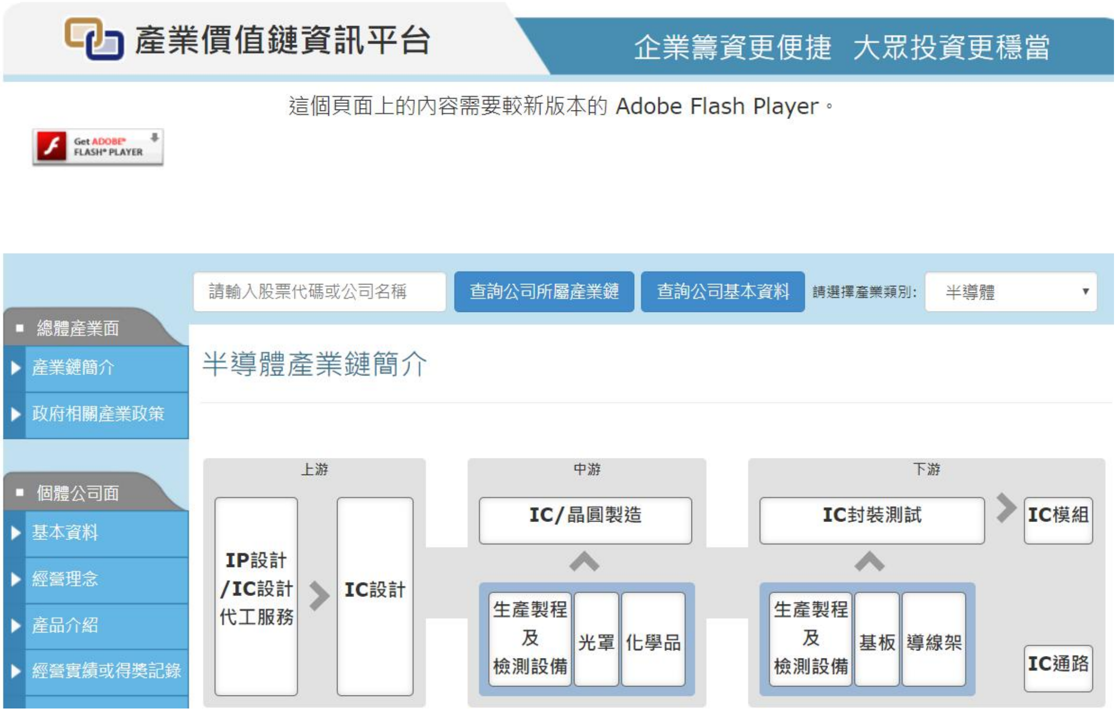
二〇一八年11月

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). 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.



這個頁面上的內容需要較新版本的 Adobe Flash Player。

图 1 产业价值链咨询平台

2. 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.

3. Solution

(1) With a corpus, use word embedding to train and map words into vector space as the basis for subsequent calculations.

(2) Use the support vector machine to train the annotated corpus.

(3) Use LSTM to determine if each vocabulary in a sentence in the document represents an upstream downstream or midstream.

4. 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.

5. 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.

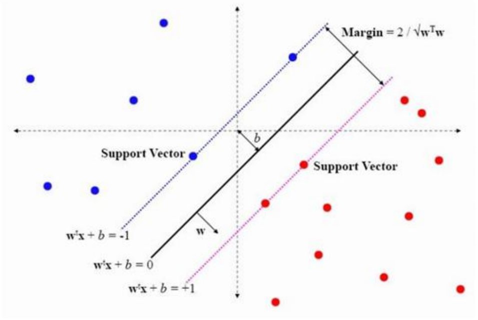


图 2 SVM

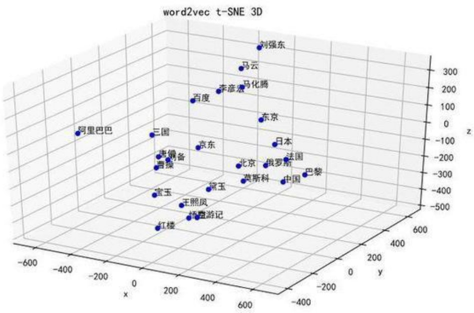


图 3 word2vec

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.

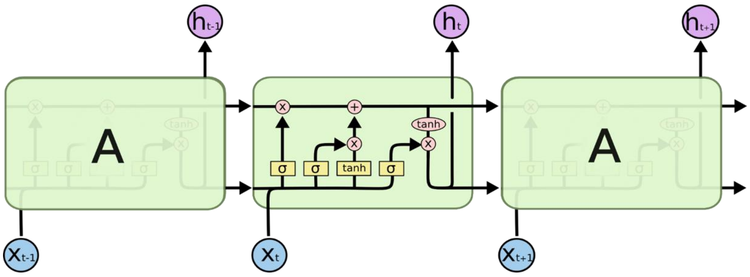


图 4 LSTM cell

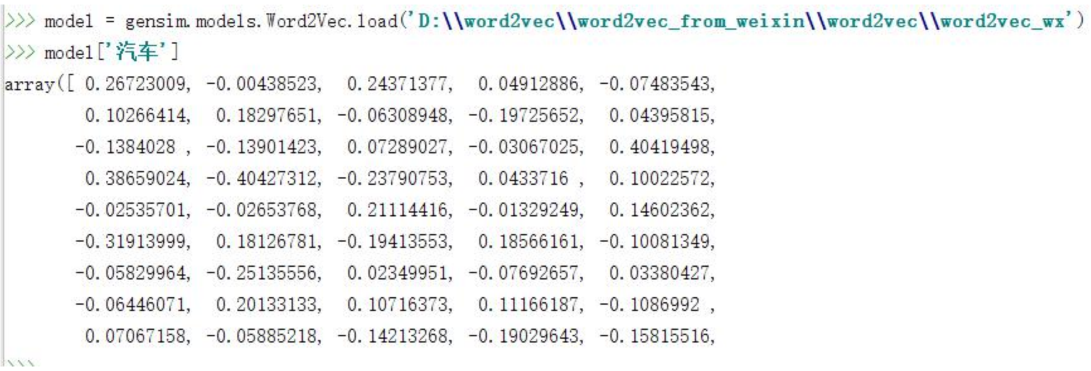
6. Data Set

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

Listed company annual reports: All the reports are collected in EastMoney. This work was done by the seniors of my laboratory and belongs to a branch of Professor Han Bo's research.

7. Experiment

(1) 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.



```
>>> model = gensim.models.Word2Vec.load('D:\\word2vec\\word2vec_from_weixin\\word2vec\\word2vec_wx')
>>> model['汽车']
array([ 0.26723009, -0.00438523,  0.24371377,  0.04912886, -0.07483543,
        0.10266414,  0.18297651, -0.06308948, -0.19725652,  0.04395815,
       -0.1384028 , -0.13901423,  0.07289027, -0.03067025,  0.40419498,
        0.38659024, -0.40427312, -0.23790753,  0.0433716 ,  0.10022572,
       -0.02535701, -0.02653768,  0.21114416, -0.01329249,  0.14602362,
       -0.31913999,  0.18126781, -0.19413553,  0.18566161, -0.10081349,
       -0.05829964, -0.25135556,  0.02349951, -0.07692657,  0.03380427,
       -0.06446071,  0.20133133,  0.10716373,  0.11166187, -0.1086992 ,
        0.07067158, -0.05885218, -0.14213268, -0.19029643, -0.15815516,
>>>
```

(2) 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()

(3) 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().

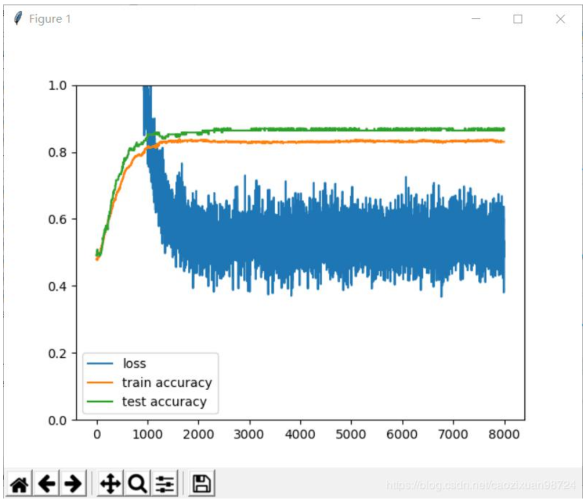
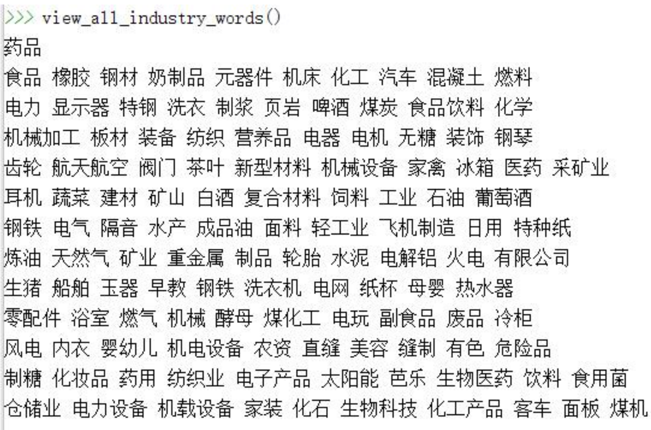


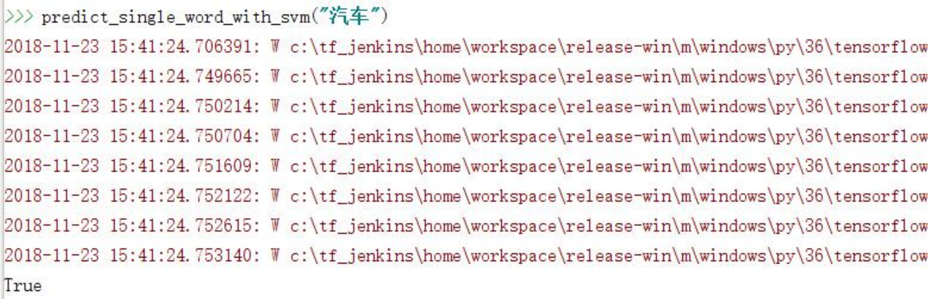
图 5 SVM accuracy and loss

(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.



```
>>> view_all_industry_words()
药品
食品 橡胶 钢材 奶制品 元器件 机床 化工 汽车 混凝土 燃料
电力 显示器 特钢 洗衣 制浆 页岩 啤酒 煤炭 食品饮料 化学
机械加工 板材 装备 纺织 营养品 电器 电机 无糖 装饰 钢琴
齿轮 航天航空 阀门 茶叶 新型材料 机械设备 家禽 冰箱 医药 采矿业
耳机 蔬菜 建材 矿山 白酒 复合材料 饲料 工业 石油 葡萄酒
钢铁 电气 隔音 水产 成品油 面料 轻工业 飞机制造 日用 特种纸
炼油 天然气 矿业 重金属 制品 轮胎 水泥 电解铝 火电 有限公司
生猪 船舶 玉器 早教 钢铁 洗衣机 电网 纸杯 母婴 热水器
零配件 浴室 燃气 机械 酵母 煤化工 电玩 副食品 废品 冷柜
风电 内衣 婴幼儿 机电设备 农资 直缝 美容 缝制 有色 危险品
制糖 化妆品 药用 纺织业 电子产品 太阳能 芭乐 生物医药 饮料 食用菌
仓储业 电力设备 机载设备 家装 化石 生物科技 化工产品 客车 面板 煤机
```

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



```
>>> predict_single_word_with_svm("汽车")
2018-11-23 15:41:24.706391: W c:\tf_jenkins\home\workspace\release-win\m\windows\py\36\tensorflow
2018-11-23 15:41:24.749665: W c:\tf_jenkins\home\workspace\release-win\m\windows\py\36\tensorflow
2018-11-23 15:41:24.750214: W c:\tf_jenkins\home\workspace\release-win\m\windows\py\36\tensorflow
2018-11-23 15:41:24.750704: W c:\tf_jenkins\home\workspace\release-win\m\windows\py\36\tensorflow
2018-11-23 15:41:24.751609: W c:\tf_jenkins\home\workspace\release-win\m\windows\py\36\tensorflow
2018-11-23 15:41:24.752122: W c:\tf_jenkins\home\workspace\release-win\m\windows\py\36\tensorflow
2018-11-23 15:41:24.752615: W c:\tf_jenkins\home\workspace\release-win\m\windows\py\36\tensorflow
2018-11-23 15:41:24.753140: W c:\tf_jenkins\home\workspace\release-win\m\windows\py\36\tensorflow
True
```

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

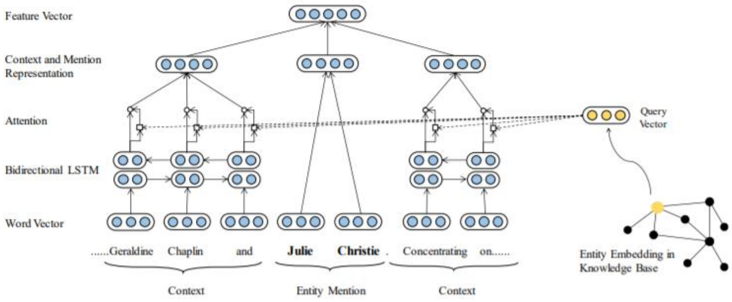


图 8 LSTM framework

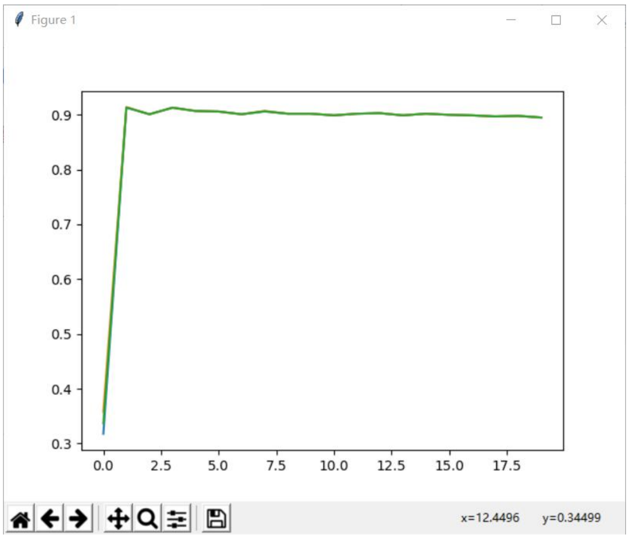
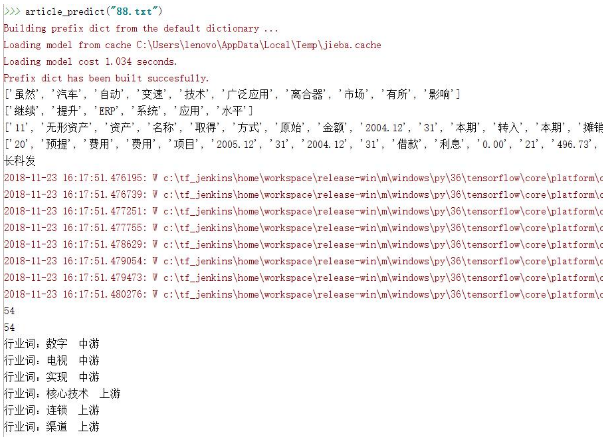


图 9 LSTM train process



```
>>> article_predict(”88.txt”)
Building prefix dict from the default dictionary ...
Loading model from cache C:\Users\lenovo\AppData\Local\Temp\jieba.cache
Loading model cost 1.034 seconds.
Prefix dict has been built succesfully.
['虽然', '汽车', '自动', '变速', '技术', '广泛应用', '离合器', '市场', '有所', '影响']
['继续', '提升', 'ERP', '系统', '应用', '水平']
['11', '无形资产', '资产', '名称', '取得', '方式', '原始', '金额', '2004.12', '31', '本期', '转入', '本期', '摊销
['20', '预提', '费用', '费用', '项目', '2005.12', '31', '2004.12', '31', '借款', '利息', '0.00', '21', '496.73',
长科发
2018-11-23 16:17:51.476195: W c:\tf_jenkins\home\workspace\release-win\m\windows\py\36\tensorflow\core\platform\c
2018-11-23 16:17:51.476739: W c:\tf_jenkins\home\workspace\release-win\m\windows\py\36\tensorflow\core\platform\c
2018-11-23 16:17:51.477251: W c:\tf_jenkins\home\workspace\release-win\m\windows\py\36\tensorflow\core\platform\c
2018-11-23 16:17:51.477755: W c:\tf_jenkins\home\workspace\release-win\m\windows\py\36\tensorflow\core\platform\c
2018-11-23 16:17:51.478629: W c:\tf_jenkins\home\workspace\release-win\m\windows\py\36\tensorflow\core\platform\c
2018-11-23 16:17:51.479054: W c:\tf_jenkins\home\workspace\release-win\m\windows\py\36\tensorflow\core\platform\c
2018-11-23 16:17:51.479473: W c:\tf_jenkins\home\workspace\release-win\m\windows\py\36\tensorflow\core\platform\c
2018-11-23 16:17:51.480276: W c:\tf_jenkins\home\workspace\release-win\m\windows\py\36\tensorflow\core\platform\c
54
54
行业词：数字  中游
行业词：电视  中游
行业词：实现  中游
行业词：核心技术  上游
行业词：连锁  上游
行业词：渠道  上游
```

图 10 analyze a report

8. 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:
    def __init__(self):
        self.input_data = tf.placeholder(tf.float32, [None, sentence_length, word_dim])
        self.output_data = tf.placeholder(tf.float32, [None, sentence_length, class_size])
        fw_cell = tf.nn.rnn_cell.LSTMCell(rnn_size, state_is_tuple=True)
        fw_cell = tf.nn.rnn_cell.DropoutWrapper(fw_cell, output_keep_prob=0.5)
        bw_cell = tf.nn.rnn_cell.LSTMCell(rnn_size, state_is_tuple=True)
        bw_cell = tf.nn.rnn_cell.DropoutWrapper(bw_cell, output_keep_prob=0.5)
        fw_cell = tf.nn.rnn_cell.MultiRNNCell([fw_cell] * num_layers, state_is_tuple=True)
        bw_cell = tf.nn.rnn_cell.MultiRNNCell([bw_cell] * num_layers, state_is_tuple=True)
        words_used_in_sent = tf.sign(tf.reduce_max(tf.abs(self.input_data),
reduction_indices=2))
        self.length = tf.cast(tf.reduce_sum(words_used_in_sent, reduction_indices=1),
tf.int32)
        output, _, _ = tf.nn.static_bidirectional_rnn(fw_cell, bw_cell,

tf.unstack(tf.transpose(self.input_data, perm=[1, 0, 2])),
                                                       dtype=tf.float32,
sequence_length=self.length)
        weight, bias = self.weight_and_bias(2 * rnn_size, class_size)
        output = tf.reshape(tf.transpose(tf.stack(output), perm=[1, 0, 2]), [-1, 2 *
rnn_size])
        prediction = tf.nn.softmax(tf.matmul(output, weight) + bias)
        self.prediction = tf.reshape(prediction, [-1, sentence_length, class_size])
        self.loss = self.cost()
        optimizer = tf.train.AdamOptimizer(learning_rate)
        tvars = tf.trainable_variables()
        grads, _ = tf.clip_by_global_norm(tf.gradients(self.loss, tvars), 10)
        self.train_op = optimizer.apply_gradients(zip(grads, tvars))

    def cost(self):
        cross_entropy = self.output_data * tf.log(self.prediction)
        cross_entropy = -tf.reduce_sum(cross_entropy, reduction_indices=2)
        mask = tf.sign(tf.reduce_max(tf.abs(self.output_data), reduction_indices=2))
        cross_entropy *= mask
        cross_entropy = tf.reduce_sum(cross_entropy, reduction_indices=1)
        cross_entropy /= tf.cast(self.length, tf.float32)
        return tf.reduce_mean(cross_entropy)

    @staticmethod
    def weight_and_bias(in_size, out_size):
        weight = tf.truncated_normal([in_size, out_size], stddev=0.01)
        bias = tf.constant(0.1, shape=[out_size])
        return tf.Variable(weight), tf.Variable(bias)




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
```

9. 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.