Web信息处理第二次实验

PB19051035 周佳豪　PB18071496　李昱祁

实验说明

实验环境

使用`python 3.9.6`

代码运行步骤

- 首先执行`entity.py`和`relation.py`两个文件，生成每个实体对应的词向量
- 然后执行`predict.py`即可得到预测的结果，储存在`result.txt`中

实验过程

生成词向量：

用`entity_with_text.txt`与`relation_with_text.txt`做语料库，通过`word2vec`得到每个实体和关系对应的词向量，结果储存在`entity_vec.txt`与`relation_vec.txt`中。

这里以对`entity_with_text.txt`的处理为例：

```
from gensim.models import word2vec
import numpy as np
import json

path_with = './lab2_dataset/entity_with_text.txt'
path_co = './lab2_dataset/entity_co.txt'
path_vec = './lab2_dataset/entity_vec.txt'


with open(path_with, 'r') as fr,open(path_co,'w') as fw:
    for line in fr.readlines():
        fw.write(line.strip().split('\t')[1])
        #print(line.strip().split('\t'))
        fw.write('\n')

sentences = word2vec.LineSentence(path_co);
model = word2vec.Word2Vec(sentences);


with open(path_with,'r') as fr,open(path_vec,'w') as fw:
    for line in fr.readlines():
        vec = 0
        entity_id = line.strip().split('\t')[0]
        words = line.strip().split('\t')[1].split(' ')
        for word in words:
            if word in model.wv:
                vec += model.wv[word]
        vec = vec/np.linalg.norm(vec)
        s = json.dumps(vec.tolist())
        s = s.strip("[]")
```

```
        fw.write(entity_id+'\t'+s+'\n')
```

训练：

模型为：

$$t = \theta_h * h + \theta_r * r$$

利用该模型对`train.txt`进行训练，结合梯度下降法找到最优的θ_h和θ_r

代价函数为：

$$J(\theta_h, \theta_r) = \frac{1}{2m}\sum_{i=1}^{m} distance(\theta_h * h^{(i)} + \theta_r * r^{(i)}, t^{(i)})$$

其中$h^{(i)}$，$r^{(i)}$，$t^{(i)}$表示`train.txt`中的第i个三元组。

梯度下降函数为：

$$\theta_h := \theta_h - \alpha * \frac{\partial J(\theta_h, \theta_r)}{\partial \theta_h}$$
$$\theta_r := \theta_r - \alpha * \frac{\partial J(\theta_h, \theta_r)}{\partial \theta_r}$$

其中α为学习速度，是一个可变的参数，代表梯度下降的速度。

偏导数为：

$$\frac{\partial J(\theta_h, \theta_r)}{\partial \theta_h} = \frac{1}{2m}\sum_{i=1}^{m} \frac{\sum_{j=1}^{m} h_j^{(i)}(\theta_h * h_j^{(i)} + \theta_r * r_j^{(i)} - t_j^{(i)})}{distance(\theta_h * h^{(i)} + \theta_r * r^{(i)}, t^{(i)})}$$
$$\frac{\partial J(\theta_h, \theta_r)}{\partial \theta_r} = \frac{1}{2m}\sum_{i=1}^{m} \frac{\sum_{j=1}^{m} r_j^{(i)}(\theta_h * h_j^{(i)} + \theta_r * r_j^{(i)} - t_j^{(i)})}{distance(\theta_h * h^{(i)} + \theta_r * r^{(i)}, t^{(i)})}$$

代码为：

```
import numpy as np
import math
from dataset import TextVecDataset, TrainTripletDataset, TestTwinsDataset

entity_vec = TextVecDataset('./lab2_dataset/entity_vec.txt')
relation_vec = TextVecDataset('./lab2_dataset/relation_vec.txt')


def distance(a, b):
    #print("begin distance")
    n = 100
    num_out = 0
    for i in range(100):
        num_in = math.pow(a[i]-b[i], 2)
        num_out += num_in
    num_out = math.sqrt(num_out)
   # print("exit distance")
    return num_out


def molecular(theta_h, theta_r, h, r, t):
```

```
    num_h = 0
    num_r = 0
    n = 100
    for j in range(n):
        temp = theta_h*h[j]+theta_r*r[j]-t[j]
        num_h += h[j]*temp
        num_r += r[j]*temp
    return num_h,num_r


def derivative(theta_h, theta_r, h, r, t):
    #print("begin derivative_r")
    global entity_vec, relation_vec
    num_out_h = 0
    num_out_r = 0
    m = 272115
    entity_vec = TextVecDataset('./lab2_dataset/entity_vec.txt')
    relation_vec = TextVecDataset('./lab2_dataset/relation_vec.txt')
    for i in range(m):
        k = 0
        if h[i] not in entity_vec.id_text or r[i] not in relation_vec.id_text or
t[i] not in entity_vec.id_text:
            num_in_h = 0
            num_in_r = 0
            if h[i] not in entity_vec.id_text and r[i] in relation_vec.id_text
and t[i] in entity_vec.id_text:
                entity_vec.id_text[h[i]] = entity_vec.id_text[t[i]
                                                              ]-
theta_r*relation_vec.id_text[r[i]]
            if r[i] not in relation_vec.id_text and h[i] in entity_vec.id_text
and t[i] in entity_vec.id_text:
                relation_vec.id_text[r[i]] = (
                    entity_vec.id_text[t[i]]-
theta_h*entity_vec.id_text[h[i]])/theta_r
            if t[i] not in entity_vec.id_text and h[i] in entity_vec.id_text and
r[i] in relation_vec.id_text:
                entity_vec.id_text[t[i]] = theta_h * entity_vec.id_text[h[i]

]+theta_r*relation_vec.id_text[r[i]]
        else:
            d = distance(theta_h*entity_vec.id_text[h[i]]+theta_r *
                         relation_vec.id_text[r[i]], entity_vec.id_text[t[i]])
            k_h,k_r = molecular(theta_h, theta_r,  entity_vec.id_text[h[i]],
relation_vec.id_text[r[i]], entity_vec.id_text[t[i]])
            num_in_r = k_r / d
            num_in_h = k_h / d
        num_out_r += num_in_r
        num_out_h += num_in_h
    num_out_r = num_out_r/(2*m)
    num_out_h = num_out_h / (2*m)
    print("num_out_r: ", num_out_r)
    print("num_out_h:", num_out_h)
    #print("exit derivative_r")
    return num_out_r, num_out_h


def train():
    #print("begin train")
```

```
    alpha = 0.025
    theta_h = 0.001
    theta_r = 0.001
    epochs = 100
    train_vec = TrainTripletDataset('./lab2_dataset/train.txt')
    for epoch in range(epochs):
        print("第%d次训练" % (epoch+1))
        temp_r, temp_h = derivative(
            theta_h, theta_r,  train_vec.triplet["h"], train_vec.triplet["r"],
train_vec.triplet["t"])
        theta_h = theta_h - alpha*temp_h
        theta_r = theta_r - alpha*temp_r
        print("theta_h=%f,theta_r=%f" % (theta_h, theta_r))
   # print("exit train")
    return theta_h, theta_r
```

预测：

最后用求得的θ_h与θ_r以及模型$t=\theta_h * h+\theta_r * r$求得标准的尾实体，用该实体与所有的实体进行距离比较，并按从小到大排序，即为预测的尾实体，并将预测的结果储存在`result.txt`中

代码为：

```
from dataset import TestTwinsDataset,TextVecDataset,TrainTripletDataset
import math
import numpy as np
import train

theta_h,theta_r = train()

test_vec = TestTwinsDataset('./lab2_dataset/test.txt')
entity_vec = TextVecDataset('./lab2_dataset/entity_vec.txt')
relation_vec = TextVecDataset('./lab2_dataset/relation_vec.txt')

def distance(a,b):
    n=100
    num_out = 0
    for i in range(n):
        num_in = math.pow(a[i]-b[i],2)
        num_out += num_in
    num_out = math.sqrt(num_out)
    return num_out
if __name__ == '__main__':
    with open('./lab2_dataset/result.txt','w')as fw:
        num = test_vec.twins_num
        print('num = %d' % num)
        for i in range(num):
            print('正在写第%d行' %i)
            index_h = test_vec.twins['h'][i]
            index_r = test_vec.twins['r'][i]
            d ={}
            if index_h in entity_vec.id_text and index_r in
relation_vec.id_text:
                t = theta_h * entity_vec.id_text[index_h]+ theta_r *
relation_vec.id_text[index_r]
            else:
                print("error")
```

```
            t = np.zeros(100)
        for j in entity_vec.id_text:
            d[j]=distance(t,entity_vec.id_text[j])
        d=sorted(d.items(),key=lambda x:x[1])
        fw.write(str(d[0][0]))
        for k in range(1,5):
            fw.write(','+str(d[k][0]))
        fw.write('\n')
```

实验结果

Hit@5=0.6%,Hit@1=0.4%，可见这种模型的预测结果很差。

原因有以下几点:

- word2vec生成词向量是根据语义相似度，但使用t = h + r的模型时却和语义相似度没太大关系
- 模型应该改为$t = \theta_h * h + \theta_r * r + \theta$,其中$\theta$是一个未知的n维向量，这样才是真正的线性回归模型，但这种模型训练起来有很大难度。

实验总结

- 通过本次实验，入门了机器学习，掌握了线性回归模型。
- 预测一定要选好模型，不能自己随便想一个，要有一定的科学依据。