第九次作业讲评

2022-04-11

郭明非

1.1 Dataframe操作

			Alice	Bob
Time	Season	Sport		
day	summer	swim	0.481751	0.983164
	winter	swim	0.988650	0.657761
		soccer	0.160044	0.528269
night	summer	swim	0.636539	0.336371
		soccer	0.067189	0.503048
		sleep	0.018342	0.628909
	winter	swim	0.675093	0.643104
		soccer	0.198039	0.982688

		Mood	Prob
Time	Season		
day	summer	Awesome	1.0
	winter	Bad	0.0
night	summer	Fine	0.2
	winter	Good	0.4

			Alice	Bob	Mood	Prob
Time	Season	Sport				
day	summer	swim	0.309471	0.584258	Awesome	1.0
	winter	swim	0.859634	0.239014	Bad	0.0
		soccer	0.420129	0.548593	Bad	0.0
night	summer	swim	0.125207	0.839748	Fine	0.2
		soccer	0.371357	0.333106	Fine	0.2
		sleep	0.403926	0.262362	Fine	0.2
	winter	swim	0.315635	0.562167	Good	0.4
		soccer	0.029991	0.497958	Good	0.4

```
# 朱芃蔓 2000013079
df2=df2.reindex(index=index)  df2: {DataFrame}(4, 2)  index:
d=pd.merge(df1, df2, on=['Time', 'Season', 'Sport'], how='outer')
print(d)
```

- merge()方法主要用于按列合并(横向合并)，但也可以按行合并(纵向合并)。
- 使用left_on, right_on和how，完成Column-Column Merge，所以需要提前reindex。on里的index名需要在两个DataFrame中都能找到。

1.1 Dataframe操作

```
# 卢红亮 1900011103
df3=pd.merge(df1,df2,left_index=True,right_index=True)
```

- 设置left_index，right_index为True，完成Index-Index Merge，合并不同的列名。此时左DataFrame的索引和右DataFrame的索引同时作为连接键。

1.1 Dataframe操作

```
# 文天宇 1900017823
df3 = pd.merge(df1, df2, left_on=['Time', 'Season'], right_index=True)
print(df3)  df3: {DataFrame}(8, 4)


df3 = pd.merge(df1, df2, right_on=['Time', 'Season'], left_index=True)
print(df3)              df3: {DataFrame}(8, 4)
```

- left和right分别设置一个index和一个on，完成Column-Index Merge。index和on中的值都必须匹配，所以使用right_on和left_index在这里会报错。

```
join_index = self._create_join_index(
    self.left.index,
    self.right.index,
    left_indexer,
    right_indexer,
    how="right",
)
```

```
Create a join index by rearranging one index to match another

Parameters
----------
index: Index being rearranged
other_index: Index used to supply values not found in index
indexer: how to rearrange index
how: replacement is only necessary if indexer based on other_index

Returns
```

1.1 Dataframe操作

```
# 杨奕章 1900012201
df1_1 = df1.join(df2)
```

- join和merge都可以用来组合两个数据帧，但是join根据它们的索引组合两个数据帧，而merge方法更通用，允许指定列来连接两个数据帧。

1.2 Dataframe操作

```
# TODO
# 刘珈征 1900012924
def cal(x):
    x.Alice = x.Alice * x.Prob
    x.Bob = x.Bob * x.Prob
    return x
df3 = df.apply(cal, axis=1)
print(df3)
```

```
# 朱芃蔓 2000013079
d['Alice']=d['Alice']*d['Prob']
d['Bob']=d['Bob']*d['Prob']
print(d)
```

2.1 分词

```
# 陈睿博 1900012203
notation = ['#', '$', ' ']
pattern = '\(.*?\)'
s = 'abcd(ac)cas(sda)'
s = re.sub(pattern, '', s)
print(s)
def cut(string, word_list, max_length):
    string=re.sub(pattern,'',string)
    res=[]
    st=0
    l=len(string)
    new_string=''
    for i in range(l):
        if string[i] not in notation:
            new_string+=string[i]

    new_l=len(new_string)
    while(st<new_l):
        for i in range(max_length,0,-1):
            if new_string[st:st+i] in word_list or i==1:
                res.append(new_string[st:st+i])
                st+=i
                break

    return res
```

- 首先需要求出词表里词语的最大长度max_length
- 每次取长度为max_length的子串：
  - 如果该子串在词表中，将这个子串放入分词结果中
  - 如果不在词表中，则去掉最后一个字符再尝试进行匹配
- 最后只剩一个单字时，如果还匹配不上（out-of-vocabulary），可以保留或删除

2.2 计算TF-IDF值

	计算	作用
TFt	词t在所有文档中出现的次数	做特征选择，有时候可以忽略总词频小于某个阈值的词
TFt,d	词t在文档d中出现的次数	可以作为文档d的特征
IDFt	N：文档总数 DFt：包含词t的文档数 $idf_t = \log\left(\frac{N}{df_t}\right)$ $idf_t = \log\left(\frac{N}{df_t + 1}\right)$	作为词t的权重，包含t的文档数越少，认为t的区分能力越大，权重（IDFt）越大

计算TF-IDF值：$tf - idf_{t,d} = tf_{t,d} \times idf_t$

2.2 计算TF-IDF值

```
# 吴悦欣 1900012946
def count(x):
    cnt = x['word'].value_counts()
    words_sum = x.shape[0]
    d = {'word':cnt.index.values,
         'TF':cnt.values/words_sum}
    cnt_2d = pd.DataFrame(data=d)


    return pd.merge(x, cnt_2d, on='word')

TF_data = new_data.groupby('Poem_id').apply(count)
```

- 返回一个包含所有值及其数量的Series。且为降序输出，即数量最多的第一行输出。在这里，index.values是单词，values是单词在文档里出现的次数。

```
# 吴悦欣 1900012946
doc_num = len(new_data['Poem_id'].value_counts())

def count_poem(x):
    df_t = len(x['Poem_id'].value_counts())
    return np.log(doc_num / df_t)

IDF_data = new_data.groupby('word').apply(count_poem)
```

2.3 计算词向量

```
# 李昊洋 2000012918
def vec(x):
    l = len(x)
    vec = []
    poem = []
    for i in range(l):
        poem.append(x.iloc[i]['word'])
    for i in word_list:
        if i in poem:
            idx = poem.index(i)
            vec.append(x.iloc[idx]['TF-IDF'])
        else:
            vec.append(0.0)
    return vec

vecGroup = new_data.groupby('Poem_id').apply(vec)
vecGroup.name = 'Poem_vec'
new_data = pd.merge(new_data, vecGroup, left_on = 'Poem_id', right_index = True)
```

- 用word_list里的词做特征计算向量，一共3570维。

2.3 计算词向量

```
# 黄倍 2000013083
count=0
dict_word2loc={}
word_tfidf=[]
for i in range(len(new_data2['word'])):
    if(new_data2.loc[i]['word'] not in dict_word2loc):
        dict_word2loc[new_data2.loc[i]['word']]=count
        word_tfidf.append(new_data2.loc[i]['tf_idf'])
        count+=1
```

- 用分出来的新词做特征计算向量。可以使用所有的分出来的新词，也可以自定义过滤低频词。

```
def cal_vec(x):
    vec1=np.zeros(len(word_tfidf))
    t=x.drop_duplicates('word')
    for i in t['word']:
        vec1[dict_word2loc[i]]=word_tfidf[dict_word2loc[i]]
    len_word=len(x['word'])
    list1=[vec1 for _ in range(len_word)]
    x['Poem_vec']=list1
    return x
```

	Poem_vec
0	[0.18420680743952367, 0.16377378248888402, 0.1...
1	[0.18420680743952367, 0.16377378248888402, 0.1...
2	[0.18420680743952367, 0.16377378248888402, 0.1...
3	[0.18420680743952367, 0.16377378248888402, 0.1...
4	[0.18420680743952367, 0.16377378248888402, 0.1...
...	...
6936	[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, ...
6937	[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, ...
6938	[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, ...
6939	[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, ...
6940	[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, ...

3.1 计算similarity

```
# 陈福康 1900013049
def similarity(vec_list):
    # 堆叠成矩阵
    vec_matrix = np.vstack(tuple(vec_list.values))
    # 对矩阵每一行向量归一化
    vec_norm_matrix = vec_matrix / np.linalg.norm(vec_matrix,axis = 1,keepdims=True)
    # 矩阵乘
    return vec_norm_matrix @ vec_norm_matrix.T
```

```
pd.DataFrame(similarity(new_data.groupby('Poem_id').head(1)['Poem_vec']),
             index=new_data['Poem_id'].unique(),
             columns=new_data['Poem_id'].unique())
```

$$sim(X,Y) = cos\theta = \frac{\vec{x} \cdot \vec{y}}{\|x\| \cdot \|y\|}$$

3.2 基于词典的分词

```python
def spfa(graph):
    # TODO
    # 方孖韬 2000012929
    path=[0]*graph.numVertices
    queue=[]
    queue.append(0)
    inq=[0]*graph.numVertices
    inq[0]=1
    dis=[1<<20]*graph.numVertices
    dis[0]=0
    while len(queue):
        u=queue.pop(0)
        inq[u]=0
        for v in range(u+1,graph.numVertices):
            if v in graph.get_vertex(u).connectedTo:
                if dis[v]>dis[u]+graph.get_vertex(u).get_weight(v):
                    dis[v]=dis[u]+graph.get_vertex(u).get_weight(v)
                    path[v]=u
                    if not inq[v]:
                        queue.append(v)
    return path
```

```cpp
vector<double> dis(N, -1000000000);          //这里实际上取了1n 相当于求最长路径
vector<bool> inq(N, 0);                      //判断节点是否处于队列中的标记
vector<int> path(N);                         //记录前驱节点
queue<int> q;
q.push(0);
inq[0] = 1;
dis[0] = 0;

while (!q.empty()) {
    int u = q.front();
    q.pop();
    inq[u] = 0;
    for (int v = u + 1; v < N; v++) {        //松弛操作
        if (dis[v] < dis[u] + adjmat[u][v]) {
            dis[v] = dis[u] + adjmat[u][v];
            path[v] = u;
            if (!inq[v]) q.push(v);
        }
    }
}
```