无监督学习文本分类作业报告

人工智能82班 刘志成 2183511589

相关代码已上传至我的github仓库：
https://github.com/zchliu/2020-Fall

问题引入

使用无监督学习的方法来对文本数据集进行分类，数据集自选

数学模型

首先对下面公式中出现的数学符号进行说明

- 假设训练集一共有L个词，每一个词用字母W_l表示，$l=0,1,...,L-1$，每一个词的出现次数用字母M_l表示，$M_l=0,1,2,...$
- 假设训练集一共有N篇文档，每一篇文档用字母X_n表示，$n=0,1,...,N-1$

idf权重模型

由于没有预先对文档做上标签，因此无法得到一个类里面的词频，因此使用idf权重模型

一个有N篇文档，L个词的idf权重公式计算如下：

$$M[l]=log(\frac{N+1}{N_l+1})$$

N_l是在所有文档中第l个词出现的文档数

$$N_l=\sum_{\forall n,W_l\in X_n}1$$

k-means聚类

k-means聚类的基本思想是指定每一个类的质心，然后找离质心最近的点，把这几个聚在一起的点看成一个类，然后重新计算质心，重复以上步骤n次进行迭代，最后就可以获得分好的聚类，步骤如下：

1. 将所有文档按词生成idf权重词典
2. 指定类的数目k，将数据集随机分成k类
3. 在每一个类当中计算质心，将这个类当中所有文档特征的tfidf权重做加权和
   $centroid[word]=\frac{1}{doc\_num}\sum_{doc}doc[word]$
4. 遍历所有点，将这个点分到离它最近的质心的那个类，距离公式为点到质心的所有词的tfidf权重绝对值之和：
   $distance=\sum_{word}|doc[word]-centroid[word]|$
5. 回到第三步直到质心不再改变

代码实现

下面这个代码用来生成一个文档的特征，这里选用50个词作为该文档的特征

```
def build_eigen(path):

    if (not os.path.exists("dictionary.pickle")):
        print("dictionaty.pickle doesn't exist!please first build dictionary!")
    else:
        dictionary_file = open("dictionary.pickle", "rb")
        dictionary = pickle.load(dictionary_file)
        dictionary_file.close()

        content = readfile(path)
        content = re.sub(pattern, ' ', content)
        content_seg = jieba.cut(content)
        raw_word_dict = defaultdict(float)
        for word in content_seg:
            if (word in stopwords):
                continue
            raw_word_dict[word] += 1

        for word in raw_word_dict.keys():
            raw_word_dict[word] = raw_word_dict[word] * dictionary[word].idf

        sorted_word_dict = sorted(raw_word_dict.items(), key=lambda item: item[1], reverse=True)
        word_dict = defaultdict(float)

        total = 0
        k = 1
        for word, word_info in sorted_word_dict:
            k += 1
            word_dict[word] = word_info
            total += word_info
            if k > 50:
                break

        for word in word_dict.keys():
            word_dict[word] = word_dict[word] / total

        return word_dict
```

下面这个代码生成文档的idf矩阵

```
def build_matrix(corpus_dir):

    dictionary = defaultdict(item)

    N = 0
    # for i in range(len(clas_list)):

    F = 0
    begin_time = time.time()
    doc_list = os.listdir(corpus_dir)

    for j in range(len(doc_list)):

        N += 1
        content = readfile(corpus_dir + "\\" + doc_list[j])
        content = re.sub(pattern, ' ', content)
        content_seg = jieba.cut(content)

        seen = []
        for word in content_seg:
            if (word in stopwords):
                continue
            if (word not in seen):
                dictionary[word].nk += 1
                seen.append(word)
            F += 1
    end_time = time.time()
    print("The corpus has words: " + str(F) + ".has docs: " + str(len(doc_list)) + ".consuming time: " + str(end_time - begin_time))

    # 计算idf
    for word in dictionary.keys():
        dictionary[word].idf = np.log((N + 1) / (dictionary[word].nk + 1))

    dictionary_file = open("dictionary.pickle", "wb")
    pickle.dump(dictionary, dictionary_file)
    dictionary_file.close()

    print("dictionary.pickle has been created!")
```

下面几行用来将文本初始化随机分为k类

```
def Random_shuffle(k, src_dir, dst_dir):

    # 把所有文档初始化为k个类
    begin_time = time.time()
    for i in range(k):
        if (not os.path.exists(dst_dir + "\\" + str(i))):
            os.mkdir(dst_dir + "\\" + str(i))

    doc_list = os.listdir(src_dir)
    for j in range(len(doc_list)):
        random_num = random.uniform(0, 1)
        for i in range(k):
            if i / k < random_num < (i + 1) / k:
                shutil.copyfile(src_dir + "\\" + doc_list[j], dst_dir + "\\" + str(i) + "\\" + doc_list[j])
    end_time = time.time()
    print("random shuffle time consuming: " + str(end_time - begin_time))
```

下面几行用来显示idf词典的大致内容

```
def view_dictionary():
    if (not os.path.exists("dictionary.pickle")):
        print("dictionaty.pickle doesn't exist!please first build dictionary!")
    else:
        dictionary_file = open("dictionary.pickle", "rb")
        dictionary = pickle.load(dictionary_file)
        dictionary_file.close()

        dic_sorted = sorted(dictionary.items(), key=lambda item: item[1].idf)
        k = 0
        for word, word_info in dic_sorted:
            print("|" + word + "|" + str(word_info.idf) + "|")
            k += 1
            if k > 20:
                break
```

这是计算两个文档距离的函数，如果两个文档当中一个重复的词都没有，那么距离是最大值2

```
def distance(word_dict1, word_dict2):

    dist = 0
    for word in (word_dict1.keys() | word_dict2.keys()):
        dist += abs(word_dict1[word] - word_dict2[word])

    return dist
```

下面是kmeans的主函数，具体步骤在数学模型里面讲到了，这里设置10次迭代

```
def kmeans(dir):

    if (not os.path.exists("dictionary.pickle")):
        print("dictionaty.pickle doesn't exist!please first build dictionary!")
    else:

        iteration = 20
        for i in range(iteration):

            begin_time = time.time()
            clas_list = os.listdir(dir)
            clas_num = len(clas_list)
            # 求每一个类的质心，第i个类的质心为centroid[i]
            centroid_list = []
            for i in range(clas_num):

                doc_list = os.listdir(dir + "\\" + clas_list[i])
                centroid = defaultdict(float)

                doc_num = len(doc_list)
                for j in range(doc_num):

                    doc_path = dir + "\\" + clas_list[i] + "\\" + doc_list[j]
                    doc_eigen = build_eigen(doc_path)
                    for word, word_info in doc_eigen.items():
                        centroid[word] += word_info

                for word in centroid.keys():
                    centroid[word] = centroid[word] / doc_num

                centroid_list.append(centroid)

            clas_list = os.listdir(dir)
            clas_num = len(clas_list)
            for i in range(clas_num):
                doc_list = os.listdir(dir + "\\" + clas_list[i])
                doc_num = len(doc_list)
                for j in range(doc_num):
                    min_dist = 10000000
                    doc_path = dir + "\\" + clas_list[i] + "\\" + doc_list[j]
                    doc_eigen = build_eigen(doc_path)
                    for k in range(len(centroid_list)):
                        dist = distance(doc_eigen, centroid_list[k])
                        if dist < min_dist:
                            min_dist = dist
                            predict_clas = clas_list[k]
                    if predict_clas == clas_list[i]:
                        continue
                    else:
                        shutil.move(dir + "\\" + clas_list[i] + "\\" + doc_list[j], dir + "\\" + predict_clas + "\\" + doc_list[j])
                        # print("move")
            end_time = time.time()
            print("update consuming time: " + str(end_time - begin_time))

            clas_list = os.listdir(dir)
            clas_num = len(clas_list)

            print("|", end="")
            correct_rate_list = [0 for i in range(clas_num + 5)]
            for i in range(clas_num):
                clas_path = dir + "\\" + clas_list[i]
                correct_rate = evaluate(clas_path)
                correct_rate_list[i] = correct_rate
                print(str(correct_rate) + "|", end="")
            print()
```

下面是对一个类聚类的正确性进行评估，利用已知的标签对类别进行评估

```
def evaluate(clas_dir):

    doc_list = os.listdir(clas_dir)
    doc_len = len(doc_list)
    clas_dict = defaultdict(int)

    for i in range(doc_len):
        doc_list[i] = doc_list[i].split("-")[0]
        clas_dict[doc_list[i]] += 1

    correct_rate = max(clas_dict.values()) / doc_len
    return correct_rate
```

模型评估

模型为复旦中文数据库，相应的介绍在homework2《文本分类实验报告》当中已经提到了，由于要进行20次迭代，为了节省时间，每一个类只用100篇文档，所以总共有9个类，每个类100篇，总共900篇文档用来分类。下面是每一次迭代的准确率：

次数	类1	类2	类3	类4
第一次	0.2	0.2	0.26732673267326734	0.2608695652173913
第二次	0.3548387096774194	0.1896551724137931	0.49137931034482757	0.33980582524271846
第三次	0.4942528735632184	0.2765957446808511	0.65	0.5053763440860215
第四次	0.5945945945945946	0.5102040816326531	0.6666666666666666	0.573170731707317
第五次	0.6388888888888888	0.6865671641791045	0.6713286713286714	0.6301369863013698
第六次	0.6428571428571429	0.8125	0.6713286713286714	0.6438356164383562
第七次	0.6521739130434783	0.8426966292134831	0.6713286713286714	0.6438356164383562
第八次	0.6521739130434783	0.8426966292134831	0.676056338028169	0.6438356164383562
第九次	0.6666666666666666	0.8426966292134831	0.676056338028169	0.6438356164383562
第十次	0.6666666666666666	0.8426966292134831	0.676056338028169	0.6438356164383562

次数	类5	类6	类7	类8	类9
第一次	0.18627450980392157	0.21505376344086022	0.4939759036144578	0.18181818181818182	0.20454545454545456
第二次	0.2711864406779661	0.3	0.5405405405405406	0.23333333333333334	0.3218390804597701
第三次	0.38738738738738737	0.4	0.5208333333333334	0.3548387096774194	0.4659090909090909
第四次	0.4262295081967213	0.6025641025641025	0.5263157894736842	0.45161290322580644	0.5555555555555556
第五次	0.47692307692307695	0.735632183908046	0.5524861878453039	0.5	0.6565656565656566
第六次	0.5241935483870968	0.8387096774193549	0.5747126436781609	0.5192307692307693	0.7582417582417582
第七次	0.5555555555555556	0.8514851485148515	0.5780346820809249	0.5192307692307693	0.8313253012048193
第八次	0.5508474576271186	0.8725490196078431	0.5714285714285714	0.5294117647058824	0.8641975308641975
第九次	0.5508474576271186	0.8737864077669902	0.5714285714285714	0.54	0.8641975308641975
第十次	0.5508474576271186	0.8737864077669902	0.5714285714285714	0.54	0.8641975308641975

大约在第八次以后迭代收敛。
对其进行5折交叉验证，平均正确率在71%左右。