实验报告1：KNN分类器

实验内容

1. 预处理文本数据集，并且得到每个文本的VSM表示。
2. 实现KNN分类器，测试其在20Newsgroups上的效果。

实验环境

- Windows 8.1
- Python 3.6(Anaconda)
- CPU-only

实验步骤

VSM模型

1. 采用textblob package进行数据预处理，分别建立每个文件的字典，计算Term Frequency(TF)，并以字典的方式存储到本地。
    - Normalization

```
s = f.read().lower() # All in lower cases
s = s.replace('/', ' ')
s = s.replace('-', ' ') # Delete '/' and '-'
```

    - Tokenization

```
w = TextBlob(s).words
```

    - Stemming

```
word = Word(word)
word = word.lemmatize() # deal with noun words
word = Word(word)
word = word.lemmatize("v") # deal with verb words
```

    - Stopwords

```
ntlk.download("stopwords") # download the stopwords dataset
stop_words = stopwords.words('english')
clean_w_list = []

if(word not in stop_words and (word not in ['\'s', '\'ll', '\'t'])):
    clean_w_list.append(word)   # delete the words in stopwords
```

    - 生成字典，计算TF

```
clean_w_dict = dict(collections.Counter(clean_w_list))
```

2. 生成训练集和测试集。
    - 80% 训练集
    - 20% 测试集

    测试集和训练集均匀的从每个类中随机抽取。

3. 遍历所有训练集文件的字典，建立Global Dictionary，计算Document Frequency(DF)。过滤掉DF小于15的单词后生成新的字典。

```
global_dict={}
for key in tmp_file_dict:
    if(key in global_dict.keys()):
        global_dict[key] += 1
    else:
        global_dict[key] = 1
```

4. 根据步骤1中计算的TF和步骤3中计算的DF，计算TF-IDF Weight，建立Vector。

$$Weight = TF * IDF$$

- Term Frequency(TF) with sub-linear scaling:

$$TF(t,d) = 1 + log(c(t,d))$$
$$c(t,d)\ be\ the\ frequency\ count\ of\ term\ t\ in\ doc\ d$$

- Inverse Document Frequency(IDF) with non-linear scaling:

$$IDF(t) = log(\frac{N}{df(t)})$$
$$df(t)\ Number\ of\ docs\ containing\ term\ t$$
$$N\ Total\ number\ of\ docs\ in\ collection$$

首先将字典中的每一个term映射到index上，然后遍历每个文件的每个term，并计算每个term的weight放到对应的index下来构造vector。（考虑到字典的长度以及文件的个数过大，为提升效率，采用scipy.sparse包中的csr_matrix稀疏矩阵来存储。）

KNN模型

KNN原理：

训练集是m个n-dimension的vector，并且已知相应的label。对测试集中的每个数据，计算训练集中距离其最近的k个neighbors（本实验中选取的数值是5），其中投票最多label即为测试集中该数据的label。

距离的定义：

- Euclidean distance

$$d(p,q) = \sqrt{\sum_{i=1}(p_i - q_i)^2}$$

- Cosine Similarity

$$d(p,q) = -cosine(p,q) = -\frac{{V_p}^T V_q}{|V_P|_2 |V_q|_2}$$

在本次实验中，选择的是Cosine Similarity作为衡量距离的单位。

Vote的权重：

根据距离给k个neighbors的vote赋予不同的权重。

$$weight\ factor = \frac{1}{d^2}$$

赋予权重的意义：

- 可以使得投票结果不仅取决于票数，且同时更趋向于距离更近的点。
- 当存在两个票数相同的label的时候，会选择距离整体距离更近的label，增加准确率。

提高算法的效率：

在计算KNN时，需要计算测试集中每个vector和训练集中每个vector的距离，时间复杂度至少为o(10^7)

参考YangLin的思路，选择了sklearn.neighbors的BallTree来提高算法效率。由于BallTree的内置距离不包含Cosine距离，可用向量归一化后的欧氏距离代替Cosine距离。推导如下：

$$Euclidean\ Distance = \frac{(X-Y)(X-Y)^T}{|X|_2 |Y|_2}$$

将上式展开可得：

$$ED = XX^T + YY^T - XY^T - YX^T$$

把X、Y向量归一化后，上式可简化为：

$$ED = \frac{2 - 2XY^T}{|X|_2 |Y|_2}$$

由上式可知，向量归一化后的欧氏距离和Cosine距离是成反比的。且向量归一化后，向量之间的夹角是不变的。

文件结构

- 主目录

| 20news-18828 存放文件数据

| file_dict_folder 存放每个文件的字典

| result_balltree 存放BallTree的运行结果

| result_original 存放原始方法的运行结果

| VSM_KNN.py 运行代码

- 结果目录（result_balltree/result_original）

| | global_dict_filtered.txt 训练集字典

| | output.txt 输出的log文件（未知原因log后缀不能与github同步，因此修改为txt文件）

|| testing_path.txt 保存测试集路径

|| training_path.txt 保存训练集路径

实验结果

BallTree Result:(with vote weight)

> true_num:3318 total_num:3759 accuracy:0.882682 The Ball Tree Query has finished: it takes 962.904263s

Original Result:(without vote weight)

> true_num:3265 total_num:3759 accuracy:0.868582 The KNN has finished: it takes 19946.281052s

由于Cosine Similarity越大越好，即d值越大两个vector越接近。而vote的weight与d成反比，d越大weight反而越小，与事实冲突。应该是d值越大weight越大才对。因而考虑在原始方法中不使用Vote Weight。

在BallTree方法中，由于最后计算的是欧式距离，所以满足Vote Weight的定义，可以使用。由结果可知，其可以在一定程度上提升预测的准确率。

实验报告2：朴素贝叶斯分类器

实验内容

实现朴素贝叶斯分类器，测试其在20 Newsgroups数据集上的效果。

实验环境

- Windows 8.1
- Python 3.6(Anaconda)
- CPU-only

实验原理

X是一个具有n个分量的特征向量，每个分量的取值为{0,1}

Y是分类结果，共有k个分类，取值范围为{1,2,...,k}

假设共m个样本

$$X \in (x_1, x_2, \ldots, x_n) x_j \in \{0, 1\}$$

$$Y \in \{1, 2, \ldots, k\}$$

$$\underset{y \in Y}{argmax}\, P(Y = y^{(i)} | X = x^{(i)})$$

由贝叶斯定理可得

$$=> \underset{y \in Y}{argmax}\, \frac{P(x|y)P(y)}{P(x)} => \underset{y \in Y}{argmax}\, P(x|y)P(y)$$

$$=> \underset{y \in Y}{argmax}\, P(x_1, x_2, \ldots, x_k|y)P(y)$$

假设X的各个分量之间是相互独立的

$$=> \underset{y \in Y}{argmax} \prod_{j=1}^{k} P(x_j|y)P(y)$$

考虑到P(x_j|y)可能为0，对其采用Laplace平滑。

- 在本实验中

P(y)=类别为y的文件个数 / 所有文件的个数

P(x_j|y)=(类别为y的文件中词语x_j出现的次数+1) / (类别为y的文件中所有词语的次数+被统计的词表中词语个数)

为方便计算，对上述目标函数取log，并采用多项式模型（统计和判断时都关注重复次数）及Laplace平滑（避免P(x_j|y) 为0）。

实验步骤

实验数据

利用Homework1中得到的

- **testing_path.txt** 测试集路径
- **training_path.txt** 训练集路径
- **global_dict_filtered.txt** 训练集全局字典
- 文件夹**file_dict_folder** 每个文件的词频

以及 文件夹**20news-18828** 原始数据

实验步骤

步骤**1:**遍历training_path，计算所有文件的个数total_count，字典 class_file_count保存每一类文件的个数，以及字符串列表保存所有类的名字class_name。

步骤**2:**将global dict中的每个单词映射到下标；将class name中的每一个类名映射到下标。

步骤**3:**遍历training_path对应文件的字典，根据步骤2生成的映射关系，建立一个大小为(class_num, term_num)的ref_matrix保存每一类中每个单词出现的频数

步骤**4:**遍历testing_path，对testing_path的每一个数据，分别计算该数据属于每个类别的概率。

对于每个类别，计算P(y):

```
p_c = class_file_count[class_name]/total_count
```

对testing_path对应文件的字典，遍历其所有包含的单词计算P(x|y):

```
p_t_c += tmp_file_dict[key] * np.log(float(tmp_term + 1) / (total_term + term_count))
```

(其中tmp_file_dict[key]是当前单词在该文件中出现的频数，tmp_term是当前单词在该类别中的频数，total_term是该类别所有单词的个数，term_count是词表中所有单词的个数)

最后得到该类别的概率：

```
p[class_idx]=p_t_c + np.log(p_c)
```

得到每个类的概率后，求最大值。得到的即为当前test数据对应的类别。

目录结构

| 20news-18828 存放文件数据

| file_dict_folder 存放每个文件的字典

global_dict_filtered.txt 训练集字典

testing_path.txt 保存测试集路径

training_path.txt 保存训练集路径

result.log 输出的log文件，保存对于每个类别的预测结果及精确度

NBC.py 朴素贝叶斯分类器实现代码

实验结果

预测的准确率为

> 0.8606012237297154

实验报告3：测试sklearn的聚类算法

实验内容

测试sklearn中以下聚类算法在tweets数据集上的聚类效果。k-means, AffinityPropagation, mean-shift, SpectralClustering, AgglomerativeClustering, DBSCAN, GaussianMixtures。

实验环境

- ubuntu16.04
- python3.7
- sklearn 0.19.2

实验原理

- **K-Means**

  首先随机选择k个聚类质心点，重复以下过程直到收敛：1、对于每个样例，选择距离其最近的聚类质心点作为其类别c。 2、对于每个类别c，计算所有包含样例的平均值作为新的质心。

- **Affinity Propagation**

  s(i,k):样本i和样本k之间的相似度。

  r(i,k):吸引度。样本k适合作为样本i的聚类中心的累积信任度。

  a(i,k):归属度。样本 i 应该选择样本 k 作为其聚类中心的累积信任度。

  damping factor:阻尼系数，为了避免 r(i,k) 和 a(i,k) 在更新时发生数值震荡；

preference:偏好参数，相似度矩阵中横轴纵轴索引相同的点。

算法流程：（1）计算初始的相似度矩阵，将各点之间的吸引度 r(i,k) 和归属度 a(i,k) 初始化为 0；

（2）更新各点之间的吸引度，随之更新各点之间的归属度，公式如下：

$$r(i,k) = s(i,k) - \max_{k' \neq k}(a(i,k') + s(i,k'))$$

$$a(i,k) = \begin{cases} \min\left\{0, r(k,k) + \sum_{i' \notin \{i,k\}} \max\left(0, r(i',k)\right)\right\}, i \neq k \\ \sum_{i' \neq k} \max\left(0, r(i',k)\right), \ i = k \end{cases}$$

按照lamda(阻尼系数)的比例进行更新。

（3）确定当前样本 i 的代表样本(exemplar)点 k，k就是使{a(i,k)+r(i,k)}取得最大值的那个 k；重复步骤 2 和步骤 3，直到所有的样本的所属都不再变化为止；

- **Mean-Shift**

  对于集合中的每一个元素x，把该元素移动到它邻域中所有元素特征值的均值的位置，不断重复直到收敛，把该元素与它的收敛位置的元素标记为同一类。考虑到邻域中每个元素对x的重要程度不一样，距离近的元素对其的影响较大，采用高斯核函数，对各个元素进行加权并取平均值，求得偏移值。

- **Spectral Clustering**

  谱聚类就是先用Laplacian eigenmaps对数据进行降维（简单来说就是先将数据转换成邻接矩阵或相似性矩阵，再转换成Laplacian矩阵，再对Laplacian矩阵进行特征分解，把最小的K个特征向量排列在一起），然后再用k-means聚类。由于降纬的作用，它的效果通常比k-means好，计算复杂度低。

- **Agglomerative Clustering**

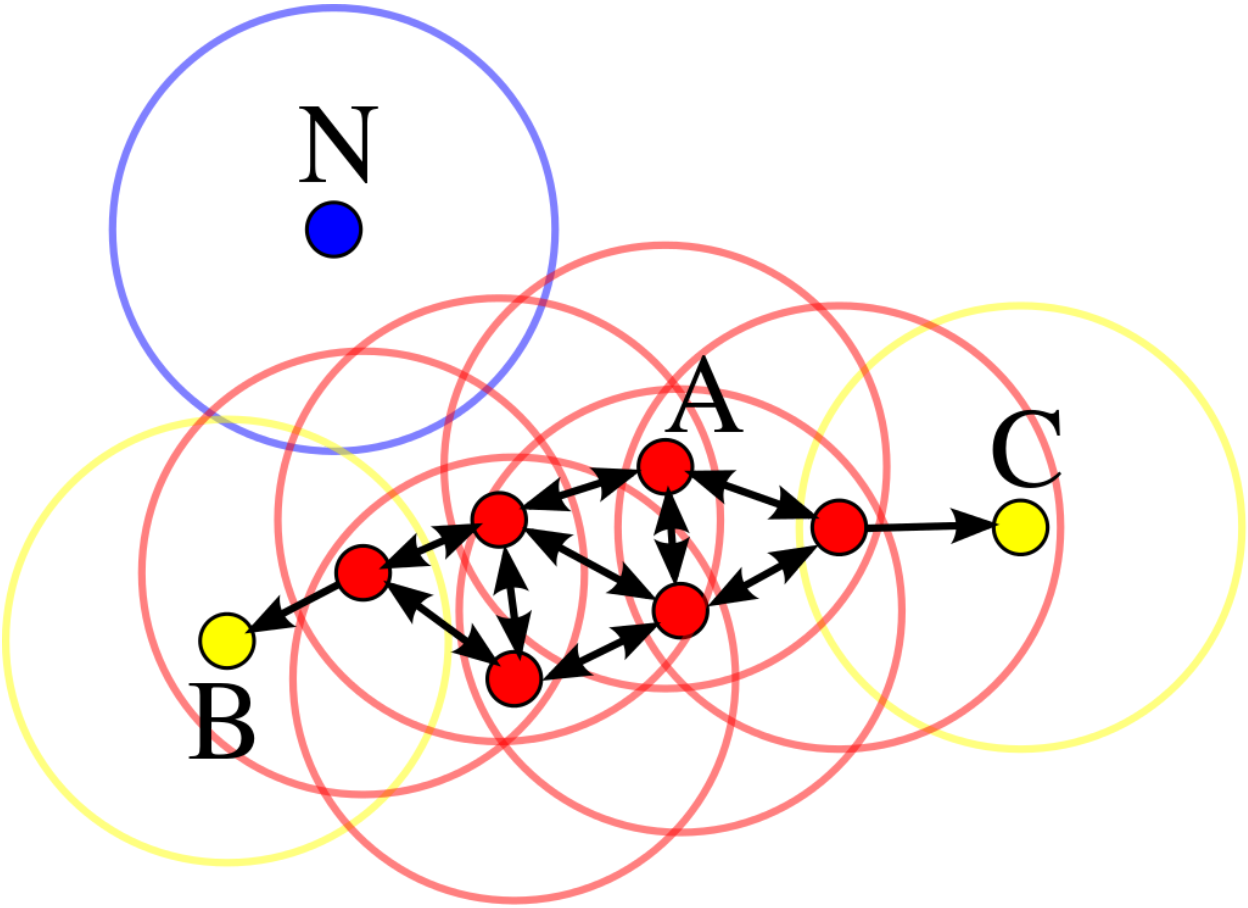
  层次聚类的方法。递归的合并能最小程度增加给定链接距离的一对簇。

  两个簇的邻近度：

  - ward：定义为两个簇合并时导致的平方误差的增量。
  - average：定义为两个簇里所有元素的平均值之间的距离。
  - complete or maximum：将两个簇里距离最远的距离定义为两个簇的距离。

- **DBSCAN**

  首先计算每个点邻域中的点的个数，若该点邻域点的个数大于形成高密度区域所需的最小点数，则讲该点定义为核心点。如果p是核心点，则它与所有由它可达的点（包括核心点和非核心点）形成一个聚类，每个聚类有最少一个核心点，非核心点也可以是聚类的一部分，但它在聚类的[边缘]位置，因为它不能达至更多的点。



- **Gaussian Mixtures**

  EM算法。观察数据x=(x_1, x_2, … , x_m) 隐变量z。

  首先初始化模型参数theta的初值theta_0。

  E-step：

  $$Q_i(z^{(i)}) := P(z^{(i)}|x^{(i)}, \theta))$$

  M-step：

  $$\theta := arg\max_{\theta} \sum_{i=1}^{m} \sum_{z^{(i)}} Q_i(z^{(i)}) logP(x^{(i)}, z^{(i)}|\theta)$$

  重复EM步骤直到theta收敛。

- **Birch**

  首先需要建立CF树。有两个重要的参数：阈值T(判断是否吸收该节点入簇) 分支阈值L(最大分支个数)

  1、从根节点向下寻找和新样本距离最近的叶子节点和叶子节点里最近的CF节点

  2、如果新样本加入后，这个CF节点对应的超球体半径仍然满足小于阈值T，则更新路径上所有的CF三元组，插入结束。否则转入3.

  3、如果当前叶子节点的CF节点个数小于阈值L，则创建一个新的CF节点，放入新样本，将新的CF节点放入这个叶子节点，更新路径上所有的CF三元组，插入结束。否则转入4。

  4、将当前叶子节点划分为两个新叶子节点，选择旧叶子节点中所有CF元组里超球体距离最远的两个CF元组，分布作为两个新叶子节点的第一个CF节点。将其他元组和新样本元组按照距离远近原则放入对应的叶子节点。依次向上检查父节点是否也要分裂，如果需要按和叶子节点分裂方式相同。

所有的叶子节点里的样本点就是一个聚类的簇。

实验步骤

- sklearn.cluster.KMeans => n_clusters 聚类个数 random_state 随机初始化
- sklearn.cluster.AffinityPropagation ==> damping 阻尼系数，防止抖动
- sklearn.cluster.MeanShift ==> bandwidth 高斯核的参数(类似于决定高纬球的半径) bin_seeding 优化加速
- sklearn.cluster.SpectralClustering ==> n_clusters 聚类个数 random_state 随机初始化
- sklearn.cluster.AgglomerativeClustering ==> n_clusters 聚类个数 linkage 确定计算距离的方法(ward, average, complete)
- sklearn.cluster.DBSCAN ==> eps 确定邻域的大小 min_samples 密集区域所需最小样本点
- sklearn.mixture.GaussianMixture ==> n_components 聚类个数
- sklearn.cluster.Birch ==> n_clusters 聚类个数 threshold 高维球半径 branching_factor 分支个数

通过sklearn.metrics.cluster的normalized_mutual_info_score方法，计算得分，判断聚类好坏。

实验结果

> K-Means score:0.779488
>
> Affinity Propogation score:0.794534
>
> Mean-Shift score:0.727855
>
> Spectral Clustering score:0.780187
>
> Agglomerative Clustering-ward score:0.783041
>
> Agglomerative Clustering-complete score:0.755440
>
> Agglomerative Clustering-average score:0.896244
>
> DBSCAN score:0.656964
>
> Gaussian Mixtures score:0.790252
>
> Birch score:0.800036