華中科技大學

课 程 实 验 报 告

课程名称： 物联网数据存储与管理

专业班级： 物联网 1801 班

学　　号： U201814573

姓　　名： 张骞

指导教师： 华宇

报告日期： 2021 年 6 月

计算机科学与技术学院

LSH | python 实现局部敏感随机投影森林

一、随机投影森林

1、随机投影森林理论与实现伪代码

当数据个数比较大的时候，线性搜索寻找 KNN 的时间开销太大，而且需要读取所有的数据在内存中，这是不现实的。因此，实际工程上，使用近似最近邻也就是 ANN 问题。

其中一种方法是利用随机投影树，对所有的数据进行划分，将每次搜索与计算的点的数目减小到一个可接受的范围，然后建立多个随机投影树构成随机投影森林，将森林的综合结果作为最终的结果。

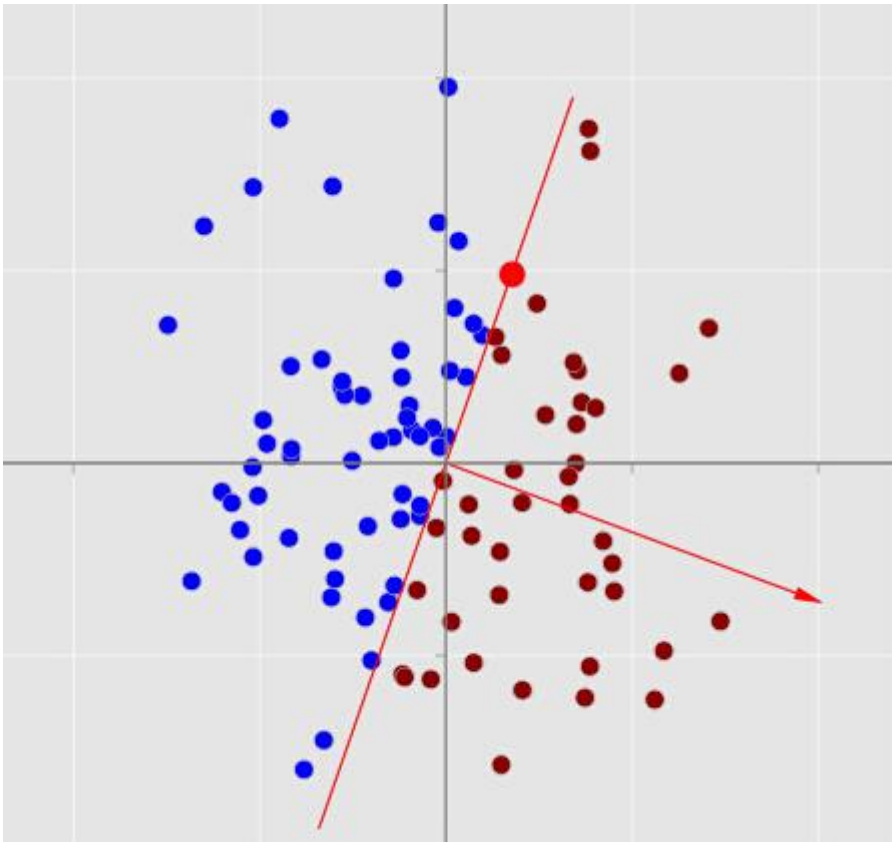
建立一棵随机投影树的过程大致如下（以二维空间为例）：

- 随机选取一个从原点出发的向量
- 与这个向量垂直的直线将平面内的点划分为了两部分
- 将属于这两部分的点分别划分给左子树和右子树

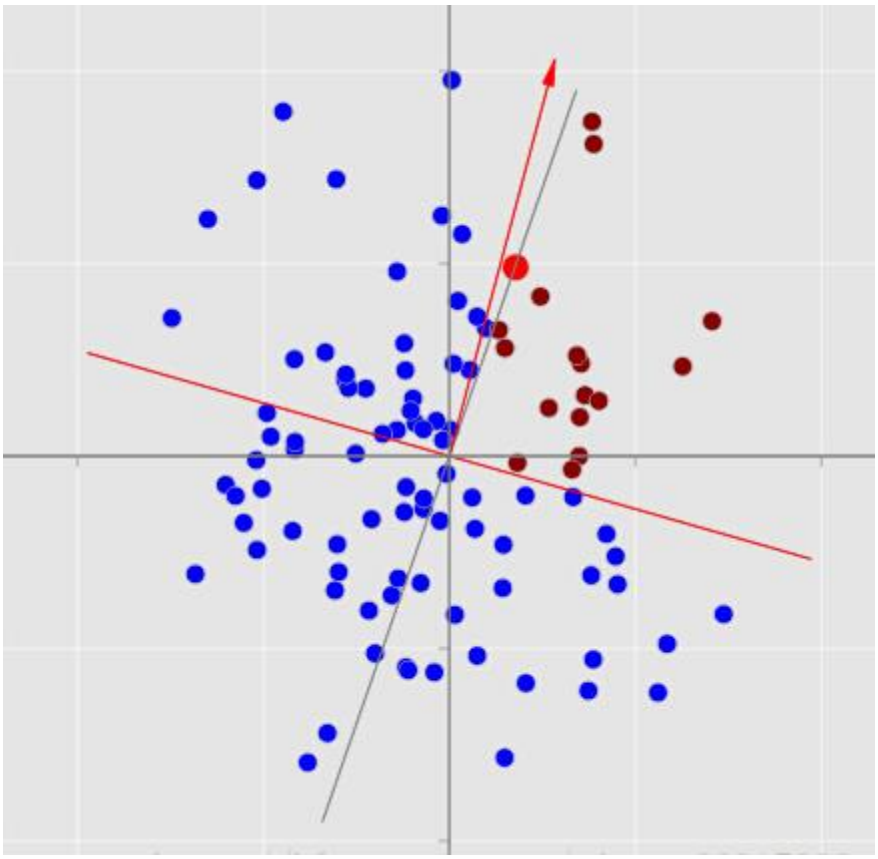
在数学计算上，是通过计算各个点与垂直向量的点积完成这一步骤的，点积大于零的点划分到左子树，点积小于零的点划分到右子树。

注意一点，图中不带箭头的直线是用于划分左右子树的依据，带箭头的向量是用于计算点积的。这样，原有的点就划分为了两部分，

图例如下：



但是此时一个划分结果内的点的数目还是比较多，因此继续划分。再次随机选取一个向量，与该向量垂直的直线将所有点进行了划分。图例如下：



注意一点，此时的划分是在上一次划分的基础上进行的。

也就是说现在图中的点已经被划分成了四部分，对应于一棵深度为2，有四个叶节点的树。以此类推继续划分下去，直到每个叶节点中点的数目都达到一个足够小的数目。注意这棵树并不是完全树。

随机投影森林的建立需要两个参数，即单棵树的深度 + 森林数量。这两个参数决定了数据集的分散程度以及随机投影后得到的向量维数。

利用这棵树对新的点进行最近邻计算时，首先通过计算该点与每次划分所用向量的点积，来找到其所属于的叶节点，然后利用这个叶节点内的这些点进行最近邻算法的计算。

这个过程是一棵随机投影树的计算过程，利用同样的方法，建立多个随机投影树构成随机森林，将森林的总和结果作为最终的结果。

.

2、相应拓展

Wright 等人 已将随机投影的方法应用于视角变化的人脸识别，Nowak 等人 采用随机投影的方法学习视觉词的相似度度量，Freund 等人将随机投影应用于手写体识别上，取得了很好的效果。

.

3、随机投影森林构造向量+聚类

论文《基于随机投影的场景文本图像聚类方法研究》中，将每一个叶子节点当成一维特征，用叶子节点的特征点个数作为叶子节点的描述，最后得到测试图像的特征向量。

有点类似 word2vec 之中的霍夫曼树。

论文中的实验结果：

表 2　微软字符数据集聚类结果

评价指标	deep = 4	deep = 6	deep = 8	deep = 10
P	72.40%	77.61%	86.80%	87.67%
R	72.19%	78.23%	87.16%	85.83%
F	72.29%	77.92%	86.98%	86.69%

表 3　AP 和 K-means 聚类方法对比

算法		deep = 6	deep = 8	deep = 10	deep = 12
AP	*P*	66.60%	79.07%	86.66%	86.02%
	R	66.33%	77.90%	85.77%	79.80%
	F	66.47%	78.48%	86.21%	82.79%
K-means	*P*	62.47%	74.83%	80.71%	81.01%
	R	61.67%	74.69%	79.95%	77.65%
	F	62.57%	74.81%	80.33%	79.29%

其中，森林规模 10 棵。

第一组实验，使用 sift 局部特征描述，在不同的 deep，树深度下识别的准确率。其中 F=（2 * R * P）/（R+P），大致来看深度 deep=8 来说，比较合理。

第二组实验，AP 聚类和 Kmeans 聚类在不同深度的差别，实验数据是 google 图片集，局部特征描述使用 ASIFT 方法，用 AP 和 Kmeans 分别进行聚类。因为 AP 聚类算法的类别数由相似矩阵的对角线元素值决定，所以需要多次测试，最终以相似度矩阵的中值为相似度矩阵对角线上的元素值，用来控制聚类的类别数。得到的 AP 聚类各项评价指标值是多次实验的平均值。而 K-means 聚类是多次实验不同的迭代次数与类别数，以最好的聚类结果作为

最终结果

表 4　ASIFT 和 SIFT 方法对比

算法		deep = 6	deep = 8	deep = 10	deep = 12
ASIFT	*P*	66.60%	79.07%	86.66%	86.02%
	R	66.33%	77.90%	85.77%	79.80%
	F	66.47%	78.48%	86.21%	82.79%
SIFT	*P*	55.04%	62.03%	71.07%	72.27%
	R	57.08%	62.85%	70.32%	65.07%
	F	56.04%	62.44%	70.69%	68.48%

第三组实验实验数据是 google 图片集，聚类算法使用 AP 聚类，用不同的局部特征描述法（ASIFT 与 SIFT）得到的聚类结果 ASIFT 局部特征描述得到的结果比 SIFT 方法在各项指标上都高 10%以上。

由此可见，ASIFT 比 SIFT 对自然场景下的文本区域图像的局部特征描述更好更准确，这是因为 SIFT 只是具有尺度和旋转不变性，对于具有视角变化的相同文字却无法得到匹配描述，而 ASIFT 不仅对图像具有尺度旋转不变性，还具有仿射不变性，这种特性对自然场景下的文本处理有更好的实用性。

详细的 ASIFT 与 SIFT 对比可见论文。

.

二、LSHForest/sklearn

LSHforest=LSH+随机投影树

在 python 的 sklearn 中有 LSHForest 可以实现。官方文档在：

sklearn.neighbors.LSHForest

1、主函数 LSHForest

class sklearn.neighbors.LSHForest(n_estimators=10, radius=1.0,

n_candidates=50, n_neighbors=5, min_hash_match=4,

radius_cutoff_ratio=0.9, random_state=None)

随机投影森林是最近邻搜索方法的一种替代方法。

LSH 森林数据结构使用已排序数组、二进制搜索和 32 位固定长度的哈希表达。随机投影计算距离是使用近似余弦距离。

n_estimators : int (default = 10)

树的数量

min_hash_match : int (default = 4)

最小哈希搜索长度/个数，小于则停止

n_candidates : int (default = 10)

每一颗树评估数量的最小值，反正至少每棵树要评估几次，雨露均沾

n_neighbors : int (default = 5)

检索时，最小近邻个数，就怕你忘记忘了设置检索数量了

radius : float, optinal (default = 1.0)

检索时，近邻个体的距离半径

radius_cutoff_ratio : float, optional (default = 0.9)

检索时，半径的下限，相当于相似性概率小于某阈值时，停止搜索，或者最小哈希搜索长度小于 4 也停止

random_state : int, RandomState instance or None, optional (default=None)

随机数生成器使用种子，默认没有附带属性：

hash_functions_ : list of GaussianRandomProjectionHash objects

哈希函数 g(p,x)，每个样本一个哈希化内容

trees_ : array, shape (n_estimators, n_samples)

Each tree (corresponding to a hash function)

每棵树对应一个哈希散列，且这个哈希散列是经过排序的。显示的是哈希值。n_estimators 棵树，n_samples 个散列。

original_indices_ : array, shape (n_estimators, n_samples)

每棵树对应一个哈希散列，哈希散列是经过排序的，显示的是原数据序号 index.

trees_ 和 original_indices_ 就是两种状态，trees_ 是每棵经过排序树

的散列，original_indices_ 是每棵经过排序树的序号 Index.

.

2、LSHForeast 相关函数

fit(X[, y])

Fit the LSH forest on the data.

数据载入投影树

get_params([deep])

Get parameters for this estimator.

获取树里面的相关参数

kneighbors(X, n_neighbors=None, return_distance=True)

检索函数，n_neighbors 代表所需近邻数， 不设置的话则返回初始化设置的数量，return_distance，是否打印/返回特定 cos 距离的样本。

返回两个 array，一个是距离 array，一个是概率 array

kneighbors_graph([X, n_neighbors, mode])

Computes the (weighted) graph of k-Neighbors for points in X

数量检索图，n_neighbors 代表所需近邻数， 不设置的话则返回初

始化设置的数量，mode=’connectivity’默认

partial_fit(X[, y])

添加数据到树里面，最好是批量导入。

radius_neighbors(X[, radius, return_distance])

Finds the neighbors within a given radius of a point or points.

半径检索，在给定的区间半径内寻找近邻，radius 为半径长度，

return_distance 代表是否打印出内容。

radius_neighbors_graph([X, radius, mode])

Computes the (weighted) graph of Neighbors for points in X

半径检索图

set_params(**params)

Set the parameters of this estimator.

重设部分参数

.

3、案例一则

>>> from sklearn.neighbors import LSHForest

```
>>> X_train = [[5, 5, 2], [21, 5, 5], [1, 1, 1], [8, 9, 1], [6, 10, 2]]
>>> X_test = [[9, 1, 6], [3, 1, 10], [7, 10, 3]]

>>> lshf = LSHForest(random_state=42)
>>> lshf.fit(X_train)
LSHForest(min_hash_match=4, n_candidates=50, n_estimators=10,
          n_neighbors=5, radius=1.0, radius_cutoff_ratio=0.9,
          random_state=42)

>>> distances, indices = lshf.kneighbors(X_test, n_neighbors=2)

>>> distances
array([[ 0.069...,    0.149...],
       [ 0.229...,    0.481...],
       [ 0.004...,    0.014...]])

>>> indices
array([[1, 2],
       [2, 0],
       [4, 0]])
```

LSHForest(random_state=42)树的初始化，

lshf.fit(X_train)开始把数据载入初始化的树；

lshf.kneighbors(X_test, n_neighbors=2)，找出 X_test 每个元素的前 2 个（n_neighbors）相似内容。

其中，这个是 cos 距离，不是相似性，如果要直观，可以被 1 减。

.

4、案例二则

来源于：用 docsim/doc2vec/LSH 比较两个文档之间的相似度

```
# 使用 lsh 来处理
tfidf_vectorizer = TfidfVectorizer(min_df=3, max_features=None,
ngram_range=(1, 2), use_idf=1, smooth_idf=1,sublinear_tf=1)
train_documents = []
for item_text in raw_documents:
    item_str = util_words_cut.get_class_words_with_space(item_text)
    train_documents.append(item_str)
x_train = tfidf_vectorizer.fit_transform(train_documents)

test_data_1 = '你好，我想问一下我想离婚他不想离，孩子他说不要，是六个月就自动生效离婚'
```

```
test_cut_raw_1 =

util_words_cut.get_class_words_with_space(test_data_1)

x_test = tfidf_vectorizer.transform([test_cut_raw_1])


lshf = LSHForest(random_state=42)

lshf.fit(x_train.toarray())


distances, indices = lshf.kneighbors(x_test.toarray(), n_neighbors=3)

print(distances)

print(indices)
```