首先介绍一下聚类和分类的算法原理：

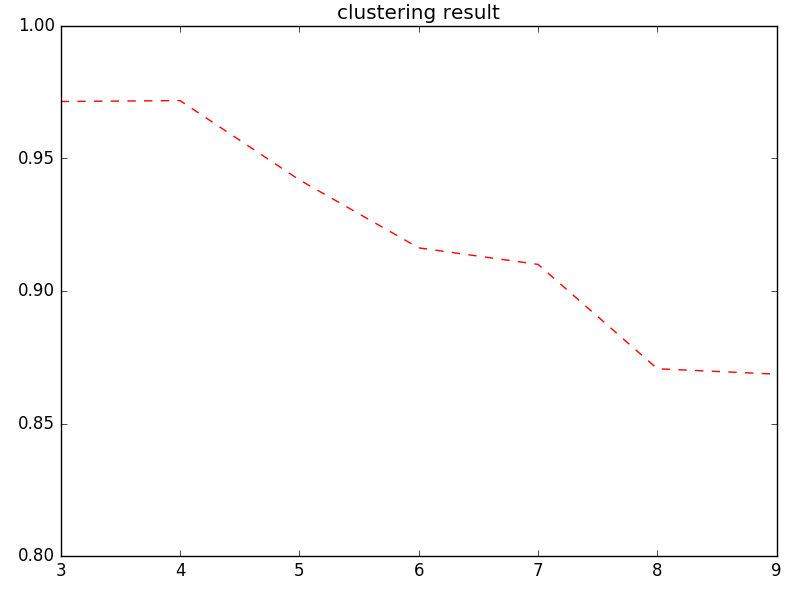
1.聚类：kmeans

|  |
| --- |
| k-means 算法接受参数 k ；然后将事先输入的n个数据对象划分为 k个聚类以便使得所获得的聚类满足：同一聚类中的对象相似度较高；而不同聚类中的对象相似度较小。聚类相似度是利用各[聚类](http://baike.baidu.com/view/31801.htm" \t "_blank)中对象的均值所获得一个“中心对象”（引力中心）来进行计算的。  [K-means](http://baike.baidu.com/view/3066906.htm)算法是最为经典的基于划分的聚类方法，是十大经典[数据挖掘算法](http://baike.baidu.com/view/8358296.htm" \t "_blank)之一。K-means算法的基本思想是：以空间中k个点为中心进行聚类，对最靠近他们的对象归类。通过迭代的方法，逐次更新各聚类中心的值，直至得到最好的聚类结果。  假设要把样本集分为c个类别，算法描述如下：  （1）适当选择c个类的初始中心；  （2）在第k次迭代中，对任意一个样本，求其到c个中心的距离，将该样本归到距离最短的中心所在的类；  （3）利用均值等方法更新该类的中心值；  （4）对于所有的c个[聚类](http://baike.baidu.com/view/31801.htm" \t "_blank)中心，如果利用（2）（3）的[迭代法](http://baike.baidu.com/view/649495.htm" \t "_blank)更新后，值保持不变，则迭代结束，否则继续迭代。  该算法的最大优势在于简洁和快速。算法的关键在于初始中心的选择和距离公式。  来自：  http://baike.baidu.com/link?url=7gqMQAUlsf5fHgGdLWSqqMvRlw3nWHTLxk1lr75NRVcaLT1jV3P1q2QT3mwZFG2kBxauJPf7zYWBo3CjQKx8h\_ |

以你的要做的内容为例，每个用户都有对每个东西点击的数量，每个用户就是一个样本，用户之间如果对诶个东西点击的数量很相似的话，那就认为是一类的。也就是说A用户点a，100次点b，200次，B用户点a，101次点，b，201ci，C用户点a，200次，点击b，100次。那么我们可以看到，明显A和B相似，A和C并不那么相似。

Kmeans定义了要聚多少类，就可以自动举出来。

|  |
| --- |
| Clustering.py |
| #coding:utf-8  import json  from sklearn.cluster import KMeans  from sklearn import metrics  from sklearn.metrics import pairwise\_distances  import numpy as np  import matplotlib.pyplot as plt  import sys  import os  reload(sys)  sys.setdefaultencoding('utf-8')  abspath = os.getcwd()  def writeMatrix(dataset, Path, coding = "utf-8"):  for i in xrange(len(dataset)):  temp = dataset[i]  temp = [str(temp[j]) for j in xrange(len(temp))]  temp = ",".join(temp)  dataset[i] = temp  string = "\n".join(dataset)  f = open(Path, "a+")  line = f.write(string+"\n")  f.close()  def getData():#整理数据的方法  file = open("data.json",'r')  data = json.loads(file.read())  file.close()  data = data['data']  # print type(data)  y = ["C000008","C000022","C000010","C000016","C000007","C000014","C000024",  "C000013","C000020","C000023"]#things\_tag  # print y  X = []#to cluster  Tag = []  # print data[0],type(data[0])  for i in range(len(data)):  temp = []  for j in range(len(y)):  if data[i].has\_key(y[j]):  temp.append(data[i][y[j]])  else:  temp.append(0)  X.append(temp)  Tag.append(data[i]["user\_crc"])  return np.array(X),Tag#将每个数据整理成能够用kmeans计算的类型  def clustering\_drawing():  X,Tag = getData()  n = 3  kmeans\_model = KMeans(n\_clusters = n).fit(X)  labels = kmeans\_model.labels\_  score = metrics.silhouette\_score(X, labels, metric='euclidean')  scoreList = [score]  nList = [3,4,5,6,7,8,9]  for i in range(4,10):# 聚类4-10类循环  # print i  kmeans\_model\_temp = KMeans(n\_clusters=i).fit(X)  labels\_temp = kmeans\_model\_temp.labels\_  score\_temp = metrics.silhouette\_score(X, labels\_temp, metric='euclidean')  print i,score\_temp  scoreList.append(float(score\_temp))  if float(score\_temp) > score:  kmeans\_model = kmeans\_model\_temp  labels = labels\_temp  n = i  print n,labels  plt.axis([3,9,0.8,1.0])  plt.plot(nList, scoreList, 'r--')  plt.show()  def clustering\_result():  X,Tag = getData()  n = 4# 四类  kmeans\_model = KMeans(n\_clusters = n).fit(X)  labels = kmeans\_model.labels\_  # print labels[0]==0,type(labels[0])  # print len(labels),len(Tag)  cluster\_n = [0,1,2,3]  for i in range(len(cluster\_n)):  temp = [cluster\_n[i]]  for j in range(len(labels)):  if cluster\_n[i] == labels[j]:  temp.append(Tag[j])  writeMatrix([temp],"cluster.txt")  if \_\_name\_\_ == '\_\_main\_\_':  clustering\_result() |



具体要聚成多少类，要么就根据问题自己随便定一个，要么根据一些方法，比如上图的

silhouette score，横坐标是要聚类的类数，纵坐标是silhouette score。可以参考维基百科

<https://en.wikipedia.org/wiki/Silhouette_(clustering)>

是一个衡量类似的一个对象是它自己的集群（凝聚力）相比，其他集群（分离）。范围从- 1到1，其中一个高的值表明，该对象是很好的匹配到自己的群集和差匹配到相邻的集群。如果大多数对象都有很高的值，那么集群配置是适当的。如果多个点有一个低或负值的值，那么集群配置可能有太多或太少的集群。

也就是说越高越高，从上图可以看到在3-9类中聚成4类，可能是相对比较好的选择。

得到结果如cluster.txt（外面）

简单地说：

|  |  |
| --- | --- |
| 类 | 用户编号 |
| 0 | 240563119,576703585,3714917228,1165373836,173546485,649106432,38120888,3910202730,3229541777,2636056694,3326299469,1870978388… |
| 1 | 1083244360 |
| 2 | 0 |
| 3 | 3767700102,1322209043,738993209,3999864637,3615280953,3425482112,3586325606,1882035991,4243279210,1925520782,1662821153,174941935… |

表中只给出了少部分例子，具体的还在txt文档中。

2.分类 朴素贝叶斯

分类和聚类有着很大的区别

|  |
| --- |
| 简单地说，分类(Categorization or Classification)就是按照某种标准给对象贴标签(label)，再根据标签来区分归类。 简单地说，聚类是指事先没有“标签”而通过某种成团分析找出事物之间存在聚集性原因的过程。   区别是，分类是事先定义好类别 ，类别数不变 。分类器需要由人工标注的分类训练语料训练得到，属于有指导学习范畴。聚类则没有事先预定的类别，类别数不确定。 聚类不需要人工标注和预先训练分类器，类别在聚类过程中自动生成 。分类适合类别或分类体系已经确定的场合，比如按照国图分类法分类图书；聚类则适合不存在分类体系、类别数不确定的场合，一般作为某些应用的前端，比如多文档文摘、搜索引擎结果后聚类(元搜索)等。  分类的目的是学会一个分类函数或分类模型(也常常称作分类器 ),该模型能把数据库中的数据项映射到给定类别中的某一个类中。 要构造分类器，需要有一个训练样本数据集作为输入。训练集由一组数据库记录或元组构成，每个元组是一个由有关字段(又称属性或特征)值组成的特征向量，此外，训练样本还有一个类别标记。一个具体样本的形式可表示为：(v1,v2,...,vn; c)；其中vi表示字段值，c表示类别。分类器的构造方法有统计方法、机器学习方法、神经网络方法等等。  聚类(clustering)是指根据“物以类聚”原理，将本身没有类别的样本聚集成不同的组，这样的一组数据对象的集合叫做簇，并且对每一个这样的簇进行描述的过程。它的目的是使得属于同一个簇的样本之间应该彼此相似，而不同簇的样本应该足够不相似。与分类规则不同，进行聚类前并不知道将要划分成几个组和什么样的组，也不知道根据哪些空间区分规则来定义组。其目的旨在发现空间实体的属性间的函数关系，挖掘的知识用以属性名为变量的数学方程来表示。聚类技术正在蓬勃发展，涉及范围包括数据挖掘、统计学、机器学习、空间数据库技术、生物学以及市场营销等领域，聚类分析已经成为数据挖掘研究领域中一个非常活跃的研究课题。常见的聚类算法包括：K-均值聚类算法、K-中心点聚类算法、CLARANS、 BIRCH、CLIQUE、DBSCAN等。 |

简单的说聚类就是说要把你们班的同学按照数学语文两门成绩，自动聚成成绩好的同学和成绩一般的同学。分类就是先按照你们班同学数学语文成绩以及他是成绩好还是成绩一般这个标签生成一个分类的标准，用术语讲就是分类模型，那么新转来一个同学，那就可以根据他的数学和语文成绩来分出他是成绩好的同学还是成绩差的同学。在实际生活中，像支付宝的打分、银行该不该给你发信用卡都是分类的实际例子，都是根据历史数据，历史的用户行为以及用户是诚信的还是不诚信这个标签，生成模型，然后对后面要发卡的人进行分类。

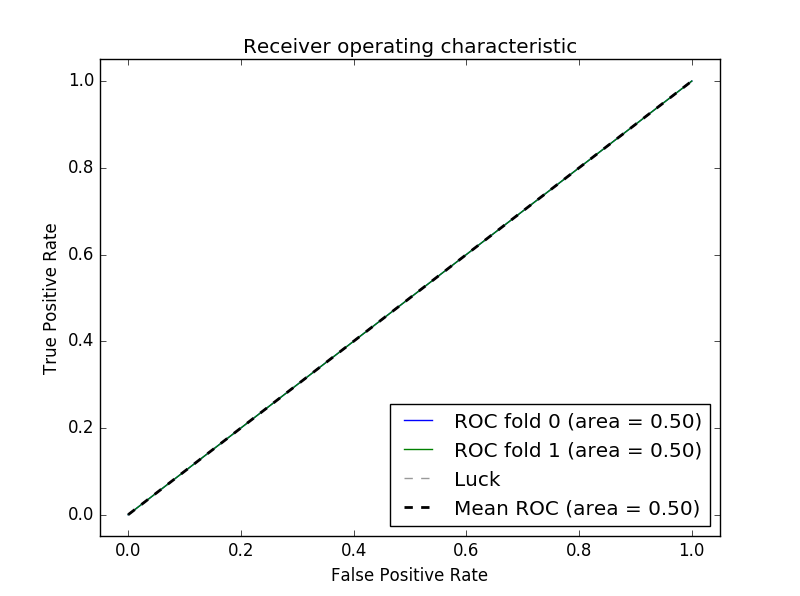
关于朴素贝叶斯

<http://www.cnblogs.com/leoo2sk/archive/2010/09/17/naive-bayesian-classifier.html>

一般都是分两类

|  |
| --- |
| Classification.py |
| #coding:utf-8  import json  import numpy as np  import matplotlib.pyplot as plt  from sklearn.cluster import KMeans  from sklearn.naive\_bayes import GaussianNB  from sklearn.metrics import roc\_curve, auc  from sklearn.cross\_validation import StratifiedKFold  from sklearn import cross\_validation  import sys  import os  from scipy import interp  reload(sys)  sys.setdefaultencoding('utf-8')  abspath = os.getcwd()  def writeMatrix(dataset, Path, coding = "utf-8"):  for i in xrange(len(dataset)):  temp = dataset[i]  temp = [str(temp[j]) for j in xrange(len(temp))]  temp = ",".join(temp)  dataset[i] = temp  string = "\n".join(dataset)  f = open(Path, "a+")  line = f.write(string+"\n")  f.close()  def getData():  file = open("data.json",'r')  data = json.loads(file.read())  file.close()  data = data['data']  # print type(data)  y = ["C000008","C000022","C000010","C000016","C000007","C000014","C000024",  "C000013","C000020","C000023"]#things\_tag  # print y  X = []#to cluster  Tag = []  # print data[0],type(data[0])  for i in range(len(data)):  temp = []  for j in range(len(y)):  if data[i].has\_key(y[j]):  temp.append(data[i][y[j]])  else:  temp.append(0)  X.append(temp)  Tag.append(data[i]["user\_crc"])  return np.array(X),Tag  if \_\_name\_\_ == '\_\_main\_\_':  # X = np.random.randint(5, size=(6, 100))  # y = np.array([1, 2, 3, 4, 5, 6])  # print X,y  # clf = MultinomialNB()  # clf.fit(X, y)  # print(clf.predict(X[2:3]))  # print X[2:3]  X,Tag = getData()  kmeans\_model = KMeans(n\_clusters = 2).fit(X)  y = kmeans\_model.labels\_  # print X,y  # print list(y),len(y)  cv = StratifiedKFold(y, n\_folds=2)  clf = GaussianNB()  mean\_tpr = 0.0  mean\_fpr = np.linspace(0, 1, 100)  all\_tpr = []  for i, (train, test) in enumerate(cv):  probas\_ = clf.fit(X[train], y[train]).predict\_proba(X[test])  # print probas\_  # print "train:", y[train],len(y[train]), "test:", y[test],len(y[test])  fpr, tpr, thresholds = roc\_curve(y[test], probas\_[:, 1])  print fpr,tpr  mean\_tpr += interp(mean\_fpr, fpr, tpr)  mean\_tpr[0] = 0.0  roc\_auc = auc(fpr, tpr)  # print roc\_auc  plt.plot(fpr, tpr, lw=1, label='ROC fold %d (area = %0.2f)' % (i, roc\_auc))  plt.plot([0, 1], [0, 1], '--', color=(0.6, 0.6, 0.6), label='Luck')  mean\_tpr /= len(cv)  mean\_tpr[-1] = 1.0  mean\_auc = auc(mean\_fpr, mean\_tpr)  plt.plot(mean\_fpr, mean\_tpr, 'k--',  label='Mean ROC (area = %0.2f)' % mean\_auc, lw=2)  plt.xlim([-0.05, 1.05])  plt.ylim([-0.05, 1.05])  plt.xlabel('False Positive Rate')  plt.ylabel('True Positive Rate')  plt.title('Receiver operating characteristic')  plt.legend(loc="lower right")  plt.show() |

得到的



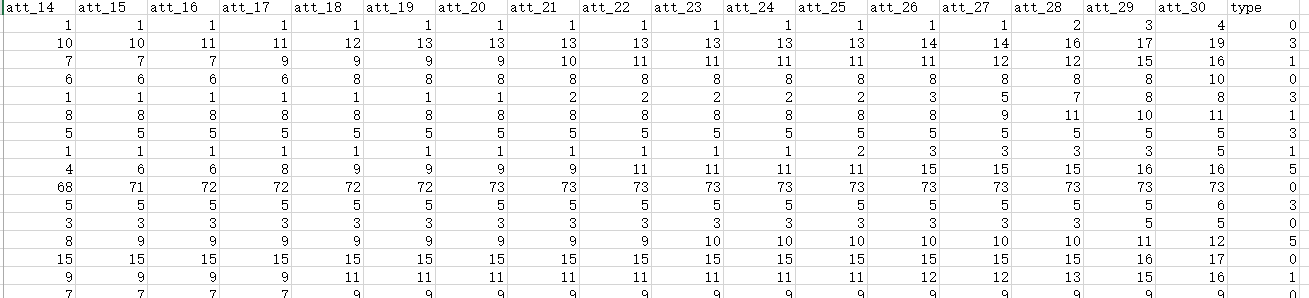
ROC曲线图，具体ROC可以参考<http://blog.csdn.net/xiahouzuoxin/article/details/43165253>

实际上这个效果并不理想，也就是说和猜测或者投掷硬币判断正反面差不多。

总结了以下原因：

1. 因为本身数据并没有类别标签，那就只能先进行聚两类（因为朴素贝叶斯一般都是分两类）然后给数据打上标签，然后进行分类。聚成两类的结果就是5141个用户，5140个用户第一类，1一个用户第二类，所以就算是新来一个人，猜他是第一类的准确率是99%以上，也就是说我这个分类器确实是由99%的准确率。这是数据本身的问题，可能有数据出了问题。
2. 另外一个是分类器选择问题，对于这种部队称的数据分类，最好效果的是SVM。也就是分类器选择问题。

PS给出了一个分类的例子



数据集就是前面很多个标签，就像你的用户对每个东西的点击一样，最后一列是type也就是分类标签，这个例子是众筹一系列数据，然后最后是否成功还是进行中还是超额完成的分类标签，用的决策树的分类模型，这个模型区别于贝叶斯分类器能够进行多类划分