**基于分类算法的笑话偏好特征挖掘**

ACM1501 李培昊 U201514616

**摘要**

信息时代中，用户的浏览、评分记录清晰了反映了用户的行为特征和习惯偏好，这对于用户行为的研究具有广泛的社会学意义和商业价值。如何高效的从用户数据中选择具有特征性的数据组，如何根据数据进行准确的行为预测是研究用户行为的一个基本任务。文中使用了Jester在线笑话推荐系统的部分匿名数据集，通过数据挖掘常用的分类算法和神经网络方法，分析用户对特定笑话的偏好问题，即分辨一个笑话是否被指定用户所喜欢。通过多种分类算法进行建模，尝试减少样本空间的特征列数，从不同算法和不同样本空间维度两个方向上分别进行对比

# 数据介绍

## 数据背景

数据来源于从1999年四月到2003年五月收集的，共计73421个用户提供的对于100个笑话的超过410万条连续评分（-10.0~+10.0）记录。

## 数据特点

1. 数据包含73421名匿名用户的评分记录。
2. 数据以Excel(.xls)格式进行存储，以.zip的形式进行压缩。
3. 评分为连续实数，范围在-10.0~+10.0分之间，而值99意味着空值或未评分。
4. 数据每行代表一个用户。
5. 每行第一列代表用户对100个笑话评价的个数，之后100列代表此用户对1~100号笑话的评分。
6. 子列{5, 7, 8, 13, 15, 16, 17, 18, 19, 20}是密集的，几乎所有用户都对这些笑话进行了评分。

## 分析目标

获取多个用户在某些的笑话上的评分，并将其当作一个数据的特征向量，判断用户对一个指定的笑话的偏好程度，进行二分类，即评分大于等于0记为1（喜欢），评分小于0记为-1（不喜欢），通过不同的分类算法实现不同的模型，使用同样的数据集对模型进行训练并验证，分析不同模型的验证准确性。

定义：大众性，为特征列的属性，表现为全部样本在此特征列的取值相同或近似，无法在此特征列上无法差异化样本，从而可能导致样本的分类预测准确度下降。

定义：样本空间特征化，通过高斯训练模型得到的特征的方差，选取其中最小的一部分列将其删除，使得样本降维。

在朴素贝叶斯算法高斯模型中，训练模型得到了全部的特征列的均值和方差，而对于某些大众列（大家都很喜欢的笑话或大家都不喜欢的笑话），其方差较小，这意味着对于不同标签的样本，在此列上的区别不明显，或许有可能因为大众列降低了不同标签的区别度，从而使预测成功的概率降低。因此使用样本空间特征化后的样本，重新训练模型，有可能使得模型的准确度上升，因此分析不同模型对于特征的独特性（区别于大众性）的依赖程度。

## 数据清洗

对于问题研究的共计19763个用户中，存在大量用户未能对全部的100个笑话进行评分。统计对全部100个笑话进行评分的用户数共计2344个，结合问题研究的重点，只选择此类用户的数据作为模型的训练集和验证集。

# 算法描述

## 总体描述

本实验重点测试了四种分类算法和一种神经网络算法实现的分类器模型。

1. 朴素贝叶斯分类器(Naïve Bayes Classifier, NBC)

朴素贝叶斯法是基于贝叶斯定理和特征属性条件独立假设的分类方法，其发源于古典数学理论，有着坚实的数学基础和稳定的分类效率。同时朴素贝叶斯分类器所需估计的参数很少，对缺失数据不太敏感，算法也比较简单。理论上NBC与其他分类方法相比有最小的误差率，但实际并非如此，这是因为NBC模型假设属性之间相互独立，这个假设在实际应用中往往不成立，给NBC的正确分类带来了一定影响。

1. 逻辑回归 (Logistic Regressio, LR)

逻辑回归是一种广义的线性回归分析模型，其结果可以是二分类的也可以是多分类的，但二分类更为常用。其特征变量可以是连续的也可以是离散的，通过逻辑回归得到特征的权重，从而预测某一特征变量下的标签值。

1. 支持向量机 (Support Vector Machine, SVM)

支持向量机被视为一中对感知机模型的扩展的线性分类器模型。感知机找到一个超平面使得数据线性可分，而支持向量机找到使得特征空间上间隔最大化的超平面作为分类器。通过支持向量机模型训练得到的分类器，预测输入的某一特征变量的分类标签。

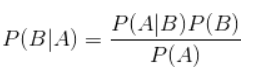
1. K邻居分类器(K Neighbors Classifier)

K邻近分类器是使用了KNN算法，即K最近邻分类法，顾名思意，算法的思想就是每个样本点都可以由它最近的K个邻居代表。通过获取其K个邻居的标签值，预测输入的某一个特征变量的分类标签。由于KNN方法主要靠周围有限的邻近的样本，而不是靠判别类域的方法来确定所属类别的，因此对于类域的交叉或重叠较多的待分样本集来说，KNN方法较其他方法更为适合。

1. 神经网络(Neural Network)

神经网络是一种模仿动物神经网络行为特征，进行分布式并行信息处理的算法数据模型。这种网络依靠系统的复杂程度，通过调整内部大量节点之间相互连接的关系，从而达到处理信息的目的。

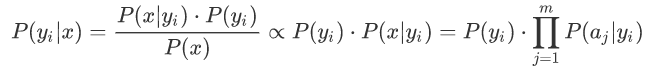
## 朴素贝叶斯分类器

贝叶斯分类法的基础是基于条件概率下的贝叶斯定理也即，通过贝叶斯定义，我们可以通过较为容易求解的P(A|B)获得不易求解的P(B|A)。

朴素贝叶斯分类的正式定义如下：

1. 设为一个待分类的数据项的特征向量，而ai是x的第i个特征属性。
2. 有标签。
3. 计算。
4. 求解k，使得。

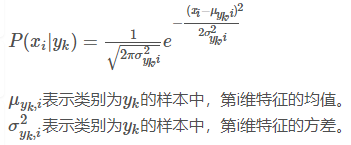
基于朴素贝叶斯的特征属性条件独立的假设，有以下公式推导



至此，朴素贝叶斯分类器的表示被推导如下：



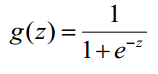
关于P(yi)和P(xj|yi)的求解，因为特征变量是连续性变量，所以选用高斯模型处理连续的特征变量，假设其值服从高斯分布（也称正态分布），即



在python实现中，使用sklearn库中的GaussianNB模型进行实现，模型获取训练样本和训练样本标签，训练得到每一个标签类每一维特征的均值和方差，并据此对输入的测试点进行预测。

|  |
| --- |
| from sklearn.naive\_bayes import GaussianNB |
|  |

## 逻辑回归分类器

逻辑回归就是一种减小预测范围，将预测值限定为[0,1]间的一种回归模型。首先要构造预测函数，即Sigmoid函数，函数形式为，函数图像如图2-1所示。

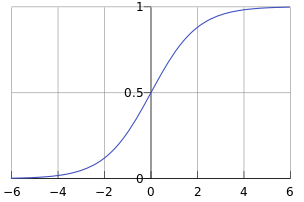
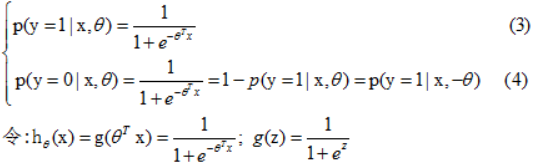
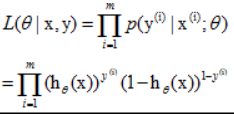


图2-1 Sigmoid 函数图像

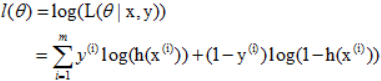
套用Sigmod函数，对于标签为C={0,1}的分类，有公式：



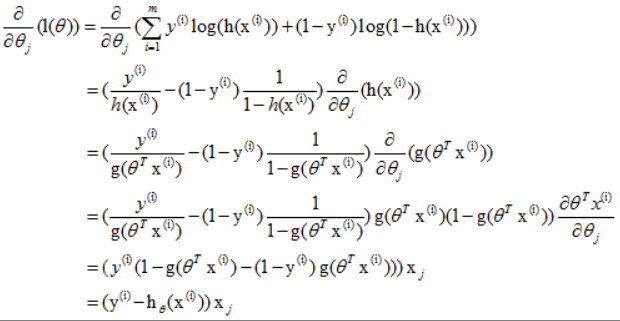
构建逻辑回归模型，对于单个样本，其后验概率为，使用极大似然估计方法得：



对其求log函数，不改变其单调性：



采用梯度下降的方法：



迭代使θ收敛即可 ，其中α为学习速率。

python实现中调用sklearn库中的LogisticRegression模型，使用训练样本和训练样本标签对模型进行训练，训练得到样本的特征系数，并据此对测试样本进行预测。

|  |
| --- |
| from sklearn.linear\_model import LogisticRegression |
|  |

## 支持向量机

和感知机模型一样，SVM求出n维空间中的超平面将数据按标签二分。区分与感知机的是SVM使得两类样本点距离超平面的最近距离最大。

给定一个特征空间上大小为N的训练数据集 ，xi为样本的第i个特征向量，yi为第i个特征向量的标签，目标是找到一个超平面使得正负类分到平面的两侧，当数据集线性可分，这样的超平面存在无数个，SVM要求间隔最大化，解唯一。超平面如图2-2所示。

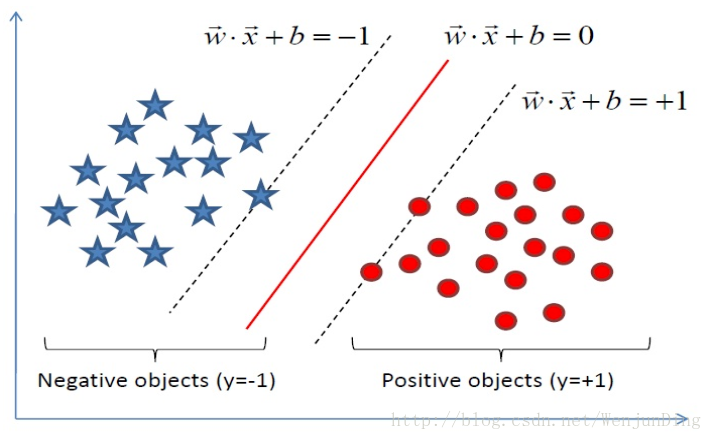
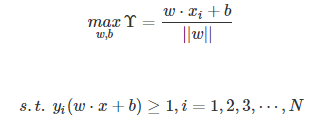
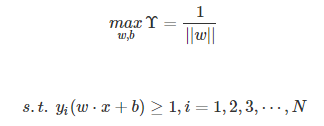


图2-2 支持向量机的超平面示意图

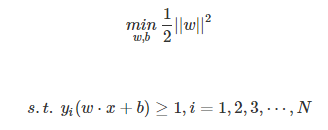
SVM的目的在于最优化几何间隔最大的超平面，在样本是线性可分时，间隔最大化被称为硬间隔，在样本近似可分时，被称为软间隔。SVM学习算法可以被表示为约束最优化问题：



在进行放缩变化(w, b)改变函数间隔的大小，但超平面不会改变，使得问题简化为：



继续进行转化可得：



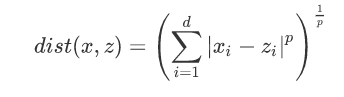
在python实现中，使用sklearn.svm库中的LinearSVC模型进行实现，模型获取训练样本和训练样本标签，训练得到每一特征的权重，并据此对输入的测试点进行预测。

|  |
| --- |
| from sklearn.svm import LinearSVC |
|  |

## K邻居分类器

kNN算法的核心思想是如果一个样本在特征空间中的k个最相邻的样本中的大多数属于某一个类别，则该样本也属于这个类别，并具有这个类别上样本的特性。该方法在确定分类决策上只依据最邻近的一个或者几个样本的类别来决定待分样本所属的类别。 kNN方法在类别决策时，只与极少量的相邻样本有关。

其中，在计算距离矩阵的方法上，一般选用Minkowski Distance方法：



其中d为特征维度，p为可变参数，通常选p=2 欧式距离，或p=∞ 最大向量有效。

算法描述为：

1. 计算测试数据与各个训练样本点之间的距离。
2. 按照距离增大的顺序进行排序。
3. 选取前K个点。
4. 确定前K个点对应标签出现的频率。
5. 将最大频率的标签作为测试数据的预测标签。

在python实现中，使用sklearn.neighbors库中的KNeighborsClassifier模型进行实现，模型获取训练样本和训练样本标签，并据此对输入的测试点进行预测。

|  |
| --- |
| from sklearn.neighbors import KNeighborsClassifier |

## 神经网络

神经网络由多个神经元构成，神经元的模型如图2-3所示。

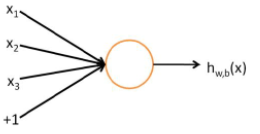


图2-3 神经元模型

这个神经元以x1,x2,x3以及截距+1作为输入值的运算待援，其输出为：



其中f(.)被称为激活函数。

神经网络就是将许多个单一的神经元联结在一起，这样，一个神经元的输出就可以使另一个神经元的输入。列如，图2-4就是一个简单的神经网络：

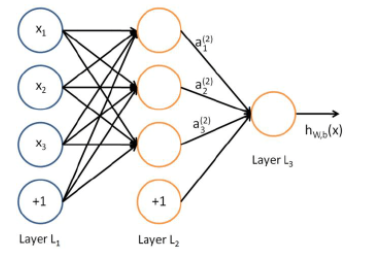
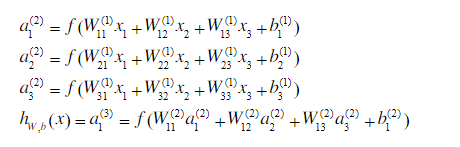
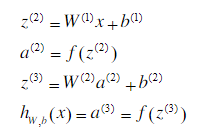


图2-4 神经网络示意图

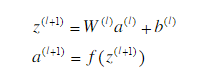
以上述模型为例，计算过程如下：



也可以表示为：



上述计算步骤被称为前向传播，给定第l层的激活值al后，第l+1层的激活值就可以通过下面的计算步骤得到：



在python实现中，使用sklearn.neighbors库中的KNeighborsClassifier模型进行实现，模型获取训练样本和训练样本标签，训练得到每层的权重矩阵和偏置矢量，并据此对输入的测试点进行预测。

|  |
| --- |
| from sklearn.neural\_network import MLPClassifier |
|  |

# 实现过程

## 总体分析

实现过程被分为两个模块，第一个模板是jester\_lib.py文件，第二个文件是run.py。

jester\_lib中主要包括但不限于数据读取，数据预处理，训练模型，预测标签，验证数据，写入文件等功能。

run中主要包括运行测试和通过IO进行模型选择功能。

## 函数设计和实现

为完成3.1小节中描述的功能函数，定义并分析函数设计和具体功能如下：

read\_file(file\_path)：函数获取文件的路径名，打开文件读取数据，将数据转为numpy格式后返回。

spilt\_data(data, verification\_proportion=0.1, label\_col=-1)：函数获取数据，可选参数包括训练集和验证集比例（默认9：1），标签列的选择（默认最后一列）。函数选取全部评价了100个笑话的用户行作为数据集，根据训练集和验证集比例分割，将其余数据作为测试集一起返回。

def get\_model(model, X\_train, Y\_train)：函数获取模型和训练集的数据及标签，返回经过训练的模型。

def prediction(model, X\_point, Y\_point=0, carry\_Point\_func=(lambda x: x.reshape(1, -1)))：函数获取训练好的预测模型、测试点，可选参数为测试点的实际标签（默认为0），对测试点的预处理函数。函数对测试点进行预处理后，通过模型进行预测，获取返回值，判断预测值与实际值是否相同并返回。

def write\_to\_file(data)：函数获取数据，写入指定的xls文件中并保存。

run模块下主要定义运行函数，运行函数循环获取用户输入进行不同模型的验证并输出结果，代码如下：

|  |
| --- |
| model\_dict = {'0': GaussianNB(),  '1': MLPClassifier(),  '2': LogisticRegression(),  '3': LinearSVC(),  '4': KNeighborsClassifier(n\_neighbors=3),  '5': GaussianNB()} model\_string\_dict = {'0': 'GaussianNB() to cutdown 1/4 feature',  '1': 'MLPClassifier()',  '2': 'LogisticRegression()',  '3': 'LinearSVC()',  '4': 'KNeighborsClassifier(n\_neighbors=3)',  '5': 'GaussianNB()'}  if \_\_name\_\_ == "\_\_main\_\_":  # testAll()  data = jester.read\_all\_file()  while 1:  try:  print("please choose model")  print(model\_string\_dict)  k = input()  X\_test, Y\_test, X\_veri, Y\_veri, X\_train, Y\_train = jester.split\_data(data)  print(model\_string\_dict.get(k))  model = jester.get\_model(model\_dict.get(k), X\_train, Y\_train)  if k == '0':  jestersort = jester.sigma(model)  data = jester.futureData(jestersort, data)  jester.verify(model, X\_veri, Y\_veri)  # while 1:  # try:  # print("please choose label col in data")  # ind = int(input())  # X\_test, Y\_test, X\_veri, Y\_veri, X\_train, Y\_train = jester.split\_data(jester.read\_all\_file(), label\_col=ind)  # model = jester.get\_model(model\_dict.get(k), X\_train, Y\_train)  # jester.verify(model, X\_veri, Y\_veri, carry\_point\_func\_dict.get(k))  # except EOFError or ValueError:  # break  except EOFError:  break |

## 功能补充

为了实现对本实验更为细致具体的功能，需在jester\_lib中添加或修改函数，描述如下。

1. 为了进行测试结果的随机性，在对数据进行预处理的过程中，不失一般性的对数据的行列进行随机化。修改read\_file函数和spilt\_data函数代码如下：

|  |
| --- |
| # 行随机化  temp\_martrix = data\_martrix.copy() random\_index = list(range(data\_martrix.shape[0])) random.shuffle(random\_index) for cur in range(data\_martrix.shape[0]):  temp\_martrix[random\_index[cur], :] = data\_martrix[cur, :] data\_martrix = temp\_martrix.copy()  # 列随机化 temp\_martrix = data.copy() random\_index = list(range(data.shape[1])) random.shuffle(random\_index) for cur in range(data.shape[1]):  data[:, random\_index[cur]] = temp\_martrix[:, cur] |

1. 为了便捷的验证全部验证集并返回验证成功率，设计实现函数verify，代码如下：

|  |
| --- |
| def verify(model, X\_veri, Y\_veri, carry\_Point\_func=(lambda x: x.reshape(1, -1))):  """  :param model: 训练好的样本  :param X\_veri: 验证集  :param Y\_veri: 验证标签  :param carry\_Point\_func: 对验证坐标点处理  :return:  """  assert model is not None  # since = time.time()  count = 0  sum = X\_veri.shape[0]  for cur in range(sum):  point = X\_veri[cur, :]  predict\_label, predict\_succ = prediction(model, point, Y\_veri[cur][0], carry\_Point\_func)  if predict\_succ:  count += 1  print("test num : %d, succ num : %d, succ precent: %f\n" % (sum, count, count \* 1.0 / sum))  return sum, count, count \* 1.0 / sum  # time\_elapsed = time.time() - since  # print('Training complete in {:.0f}m {:.0f}s'.format(time\_elapsed // 60, time\_elapsed % 60)) |

1. 在朴素贝叶斯算法高斯模型中，训练模型得到了全部的特征列的均值和方差，选取其中方差最小的一部分列删除，以提高样本特征的独特性。代码如下：

|  |
| --- |
| def sigma(model):  assert model is not None  t = model.sigma\_  t = numpy.sum(t, axis=0)  print(t)  sortInd = t.argsort()  return sortInd[:int(len(sortInd) / 4)]   def futureData(sortInd, data: object) -> object:  sortInd = numpy.sort(sortInd)  sortInd = sortInd[::-1]  # print(sortInd)  for cur in range(len(sortInd)):  data = numpy.delete(data, sortInd[cur], axis=1)  return data |

# 结果展示

## 运行演示

使用python运行run.py，运行结果如图4-1所示。

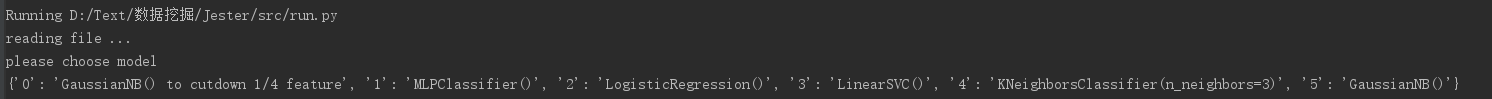


图4-1 运行演示

输入1，选择神经网络模型进行测试，基础的特征列数都是99，测试结果如图4-2所示。

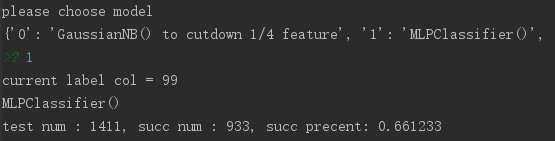


图4-2 神经网络模型99特征列测试

输入2，选择逻辑回归模型进行测试，特征列数为99个，测试结果如图4-3所示。

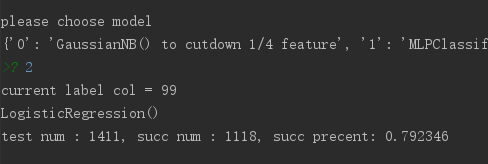


图4-3 逻辑回归模型99特征列测试

输入3，选择SVM模型进行测试，特征列数为99个，测试结果如图4-4所示。

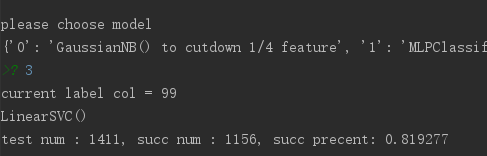


图4-4 SVM模型99特征列数测试

输入4，选择KNN模型进行测试，特征列数为99个，测试结果如图4-5所示。

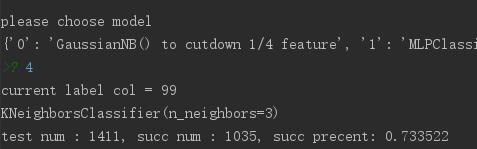


图4-5 KNN模型99特征列数测试

输入5，选择朴素贝叶斯高斯模型进行测试，特征列数为99个，测试结果如图4-6所示。

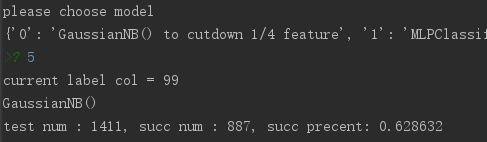


图4-6 朴素贝叶斯高斯模型99特征列数测试

输入0，选择朴素贝叶斯高斯模型进行测试，特征列数为99个，测试结果如图4-7所示，测试后去除最大众性的1/4特征列。

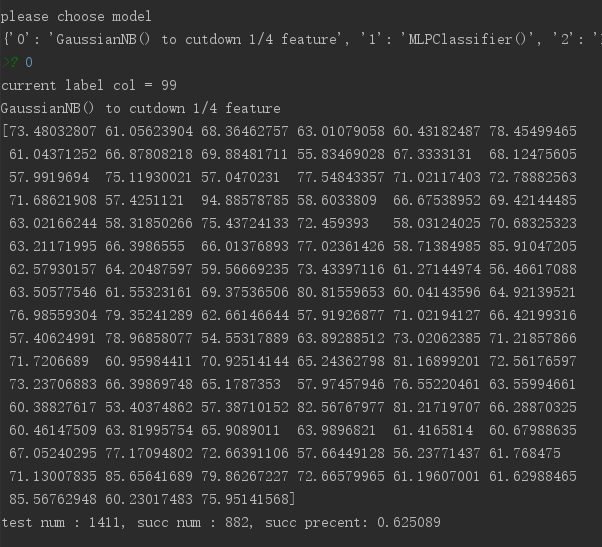


图4-7 朴素贝叶斯高斯模型去除1/4特征列数测试

输入1，选择神经网络模型进行测试，特征列数为75个，测试结果如图4-8所示。

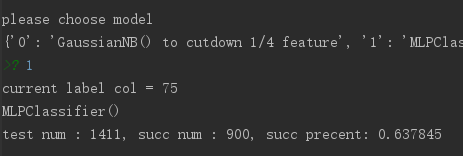


图4-8 神经网络模型75特征列数测试

输入0，选择朴素贝叶斯高斯模型进行测试，特征列数为75个，测试结果如图4-9所示，测试后去除最大众性的1/4特征列。

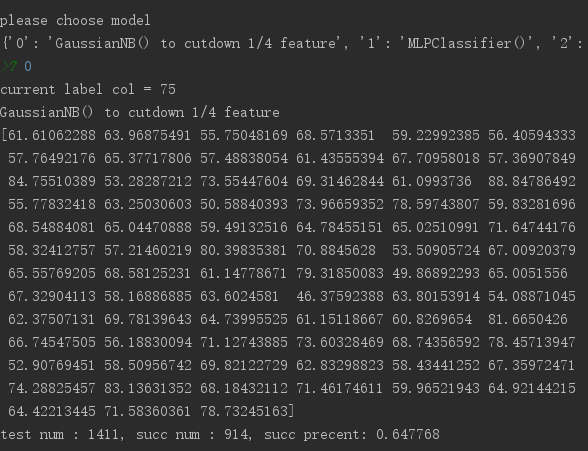


图4-9 朴素贝叶斯高斯模型去除1/4特征列数测试

输入2，选择逻辑回归模型进行测试，特征列数为57个，测试结果如图4-10所示.

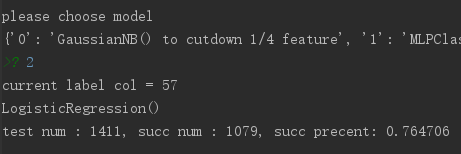


图4-10 逻辑回归模型57特征列数测试

输入0，选择朴素贝叶斯高斯模型进行测试，特征列数为57个，测试结果如图4-11所示，测试后去除最大众性的1/4特征列。

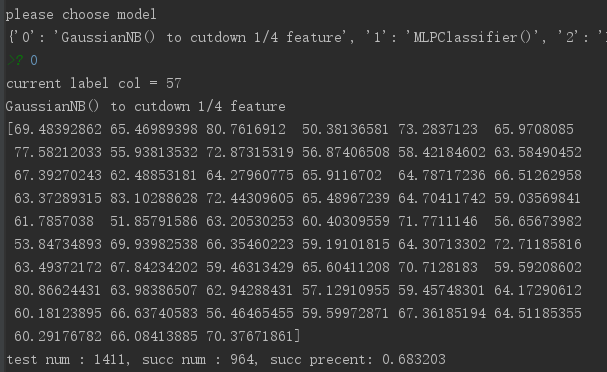


图4-11 朴素贝叶斯高斯模型去除1/4特征列数测试

输入4，选择SVM模型进行测试，特征列数为43个，测试结果如图4-12所示.

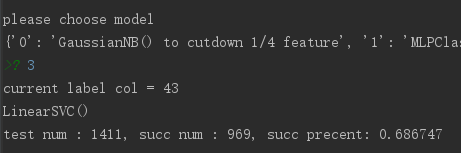


图4-12 SVM模型43特征列数测试

## 结果记录

分别对于5个不同的模型，对于三个不同特征列数取值进行10次测试，计算测试成功率的最值，均值和方差。结果见下表

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 表4-1 测试结果记录表 | | | | | |
|  | feature | 99 | | | |
|  | count | MAX | MIN | 方差 | **均值** |
| Name | Method |  |  |  |  |
| neural\_network | MLPClassifier | 0.771793 | 0.593196 | 0.002731 | **0.71318** |
| linear\_model | LogisticRegression | 0.796598 | 0.686038 | 0.001214 | **0.76258** |
| svm | LinearSVC | 0.809355 | 0.55776 | 0.006436 | **0.70666** |
| neighbors | KNeighborsClassifier | 0.728561 | 0.579731 | 0.003145 | **0.68448** |
| naive\_bayes | GaussianNB | 0.733522 | 0.605245 | 0.001648 | **0.65811** |
|  | feature | 75 | | | |
|  | count | MAX | MIN | 方差 | **均值** |
| Name | Method |  |  |  |  |
| neural\_network | MLPClassifier | 0.716513 | 0.700921 | 0.65769 | **0.66052** |
| linear\_model | LogisticRegression | 0.753366 | 0.808646 | 0.819986 | **0.6995** |
| svm | LinearSVC | 0.743444 | 0.641389 | 0.692417 | **0.81999** |
| neighbors | KNeighborsClassifier | 0.781006 | 0.705882 | 0.595322 | **0.64989** |
| naive\_bayes | GaussianNB | 0.632884 | 0.654855 | 0.717222 | **0.6832** |
|  | feature | 57 | | | |
|  | count | MAX | MIN | 方差 | **均值** |
| Name | Method |  |  |  |  |
| neural\_network | MLPClassifier | 0.822821 | 0.581148 | 0.672573 | **0.61233** |
| linear\_model | LogisticRegression | 0.722183 | 0.785259 | 0.783133 | **0.7888** |
| svm | LinearSVC | 0.557052 | 0.691708 | 0.712261 | **0.65131** |
| neighbors | KNeighborsClassifier | 0.804394 | 0.65769 | 0.731396 | **0.59391** |
| naive\_bayes | GaussianNB | 0.693125 | 0.561304 | 0.664777 | **0.69809** |
|  | feature | 43 | | | |
|  | count | MAX | MIN | 方差 | **均值** |
| Name | Method |  |  |  |  |
| neural\_network | MLPClassifier | 0.716513 | 0.815025 | 0.647059 | **0.59249** |
| linear\_model | LogisticRegression | 0.728561 | 0.734231 | 0.771793 | **0.69738** |
| svm | LinearSVC | 0.623671 | 0.677534 | 0.815734 | **0.63785** |
| neighbors | KNeighborsClassifier | 0.681786 | 0.804394 | 0.635011 | **0.61233** |
| naive\_bayes | GaussianNB | 0.638554 | 0.705882 | 0.67399 | **0.70872** |

## 数据分析

根据上表中的数据，生成验证成功率（此后简称“值”）数据统计图表如图4-13~图4-16所示。对每个图以1.对比不同模型的差距。2.对比不同特征列数（此后简称“列数”）的差距。 3.对比不同列数对于同一模型的影响。 4.对比不同模型对于同一列数的影响。

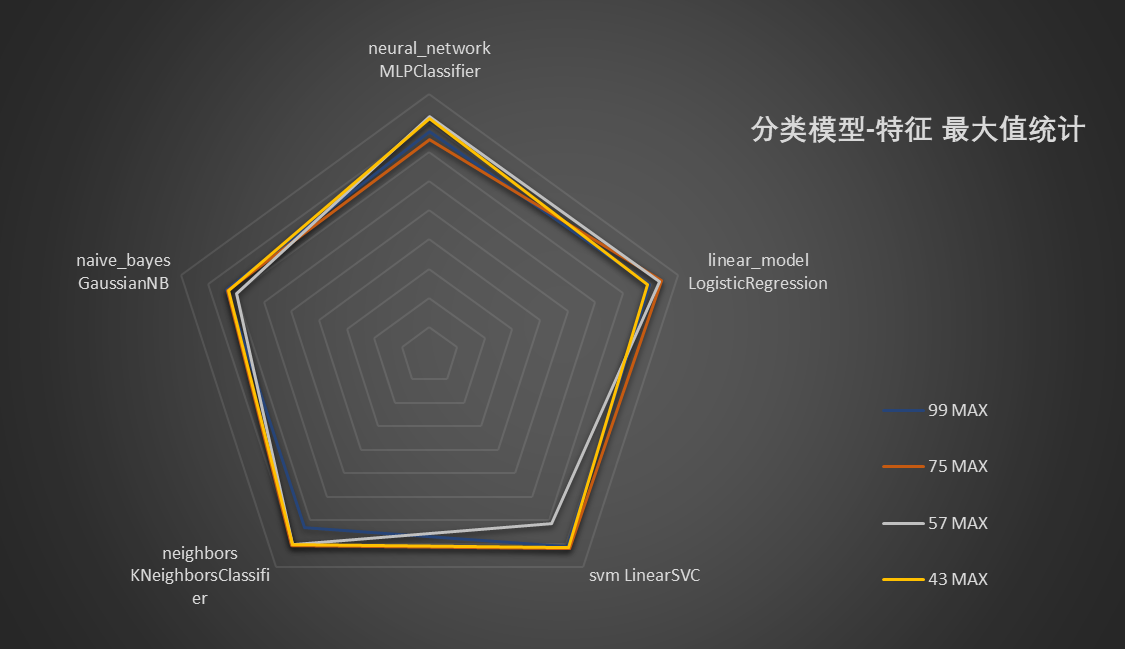


图4-13 分类模型-特征 预测成功率最大值统计图

分析图4-13预测成功率最大值统计图：

1. 对比不同模型，除朴素贝叶斯模型的最大值较小外，其余四种模型的最大值基本相同。
2. 对比不同列数，可以发现去除超过一半的特征列后，特征列数为43时，最大值基本为每个模型不同列数的最大值中最大的。而列数为99、75、57时，都在某一种模型中属于最大值最小的列数值。
3. 对比不同列数对同一模型的影响，发现对于朴素贝叶斯模型和逻辑回归模型中，不同列数对其影响较小，而剩余三者中模型预测成功率的最大值在不同列数上偏差较大。
4. 对比不同模型对于同一列数的影响，可以发现对于列数为43和75的情况下，不同模型的影响较小，而对于列数为99和57的情况下，不同模型影响较大，在SVM模型下，列数为99取得最大值，列数为57取得最小值。

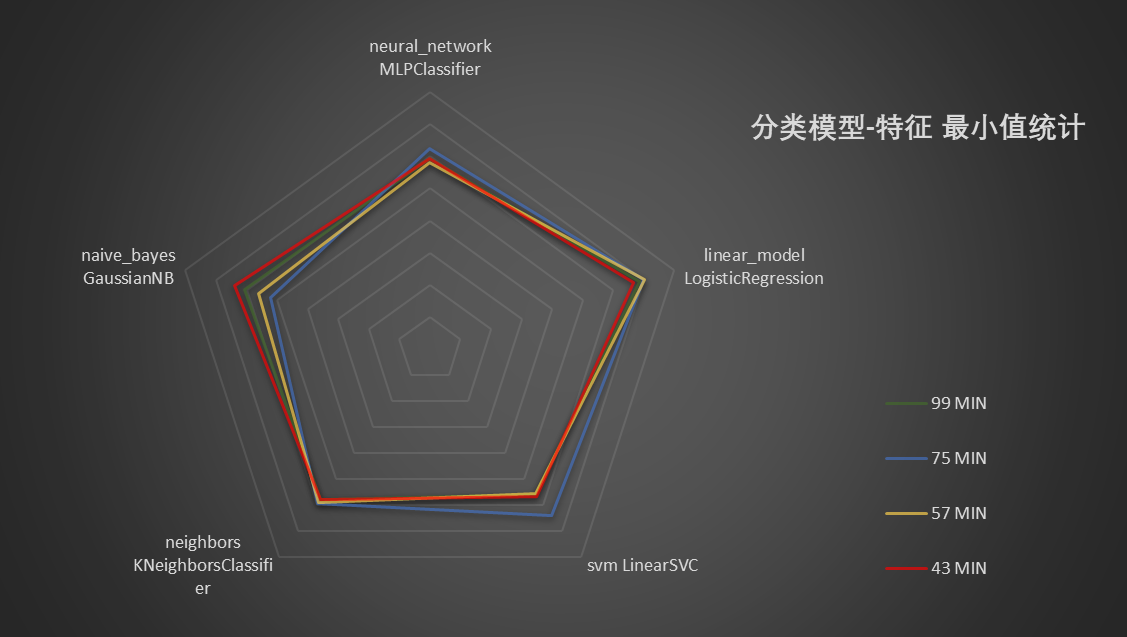


图4-14 分类模型-特征 预测成功率最小值统计图

分析图4-14 分类模型-特征 预测成功率最小值统计图：

1. 对比不同模型，除逻辑回归模型的最小值较大外，其余四种模型的最小值基本相同。
2. 对比不同列数，可以发现列数为99的情况下，最小值较为稳定。而列数为75、57、43的情况下，最小值波动较为严重。
3. 对比不同列数对同一模型的影响，发现对于逻辑回归、KNN模型和神经网络模型，不同列数对模型最小值的影响较小，对于朴素贝叶斯来说，列数为43时，预测成功率最小值最大，列数为75时，最小值最小。而列数为75时，SVM模型最小值最大。
4. 对比不同模型对于同一列数的影响，可以发现逻辑回归模型对在每一个列数的情况下，最小值普遍较大。在朴素贝叶斯模型上，最小值普遍较小。其余三种模型下，不同列数的情况最小值基本一致。

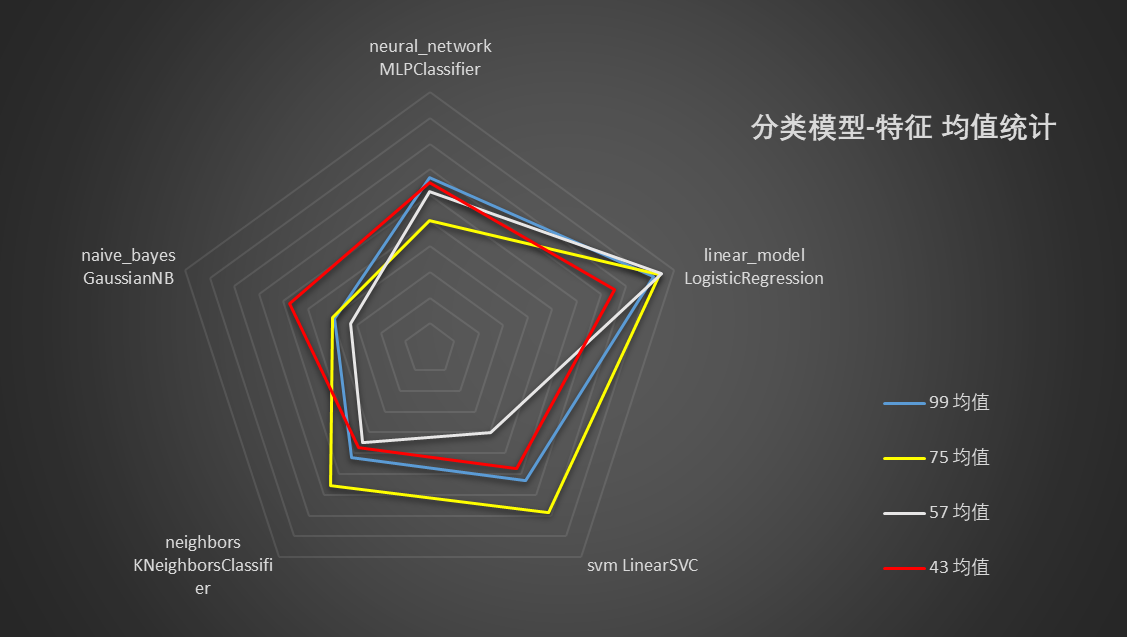


图4-15 分类模型-特征 预测成功率均值统计图

分析图4-15 分类模型-特征 预测成功率均值统计图

1. 对比不同模型，显著观察值逻辑回归模型的均值最大，SVM次之，朴素贝叶斯模型均值最小，其余二者相近。
2. 对比不同列数，可以看到列数为75的模型均值较大，列数为99的模型次之，列数为43的模型较为稳定，列数为57的模型均值最小。
3. 对比不同列数对同一模型的影响，发现对于朴素贝叶斯模型和SVM模型中，不同列数对其均值的影响十分大，对于神经网络和KNN模型，不同列数对其影响较大，不同列数对逻辑回归模型有一定影响。
4. 对比不同模型对于同一列数的影响，可以看到列数为75时，在逻辑回归、SVM、KNN模型下均值都取得最大值。而列数为57时，在朴素贝叶斯、KNN、SVM模型下均值取得最小值。列数为43的模型最为稳定，列数为57的模型最不稳定，列数为99和75的模型稳定性较差。

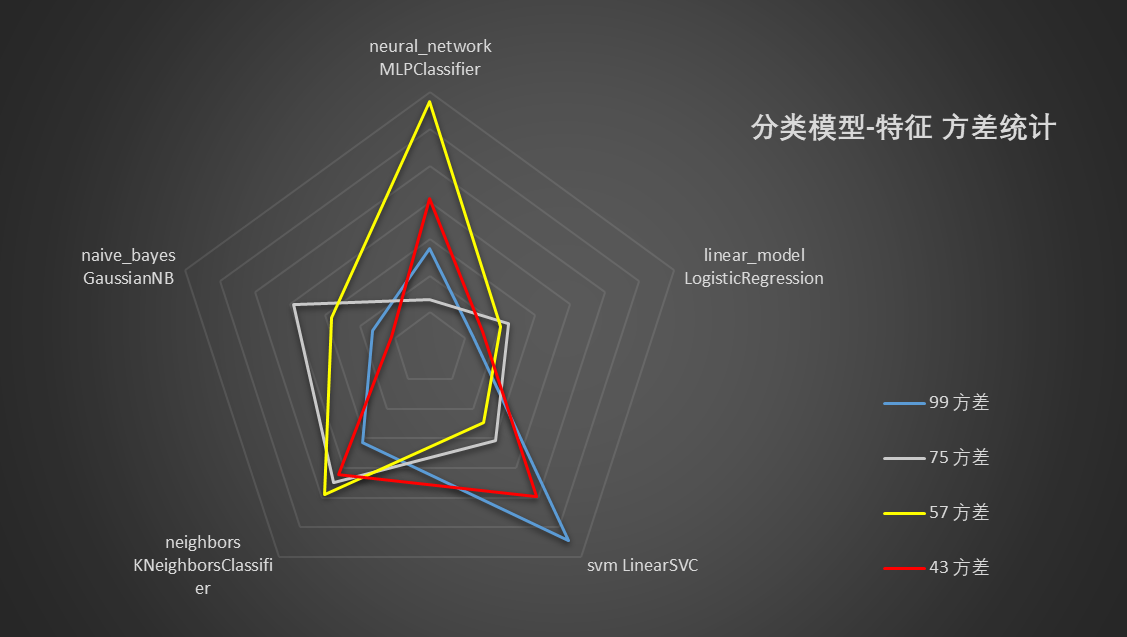


图4-16 分类模型-特征 预测成功率方差统计图

1. 对比不同模型，除朴素贝叶斯模型的方差很小外，其余四种模型的方差波动性十分大，其中神经网络和SVM模型的方差最大。
2. 对比不同列数，可以发现列数为43时方差较小，而列数为57的方差较大，列数为75和99的模型预测结果方差稳定性较差。
3. 对比不同列数对同一模型的影响，发现不同列数对逻辑回归和KNN模型以及朴素贝叶斯模型的方差影响较小，而对SVM和神经网络模型的方差影响十分剧烈。
4. 对比不同模型对于同一列数的影响，可以发现对于列数为75的情况，不同模型对其影响较小，列数为57的情况，不同模型对其影响巨大。而SVM模型对列数为99时偏大，神经网络模型对列数为57时方差影响十分大，其余模型对列数的影响较小。

## 结果评估

总体而言，逻辑回归模型表现最好，其在不同的列数下预测成功率最大值较稳定，最小值较大，均值最高，方差最小，在列数为99、75、57时，均值大概在76%，即使在列数为43时，表现最差为73%，也比第二名神经网络模型的70%的均值大了3%。

不同列数对于SVM模型的影响最大，当列数为75时，均值为73%仅次于逻辑回归，且较第三名高了3%，当列数为57时，均值为65%，当列数为99时，方差十分大，当列数为43时，方差次之。

列数为43的模型表现出了令人难以置信的稳定性，对于同一的模型，列数为43的情况几乎总能取得最大的最大值和最小值，对于不同模型，列数为43时能取得十分稳定且并不小的成功率均值和较为稳定的测试方差。

综上所述，逻辑回归是表现最好的分类模型，但在过小的特征列情况下表现稍差；SVM模型是最不稳定的模型，能够在较大的特征列数情况下取得最好的均值和最值，在较小的特征列数情况下取得最小的方差；KNN模型和神经网络模型有相似之处，二者在不同的列数情况下能够取得近似的最值，当相对而言，不同的列数对KNN模型中均值的影响较大，对神经网络模型中方差的影响较小，因此可以通过调整参数特征列的数目的方法获取二者的最优模型；朴素贝叶斯算法高斯模型在特征列数最小的情况下表现出了十分优秀的性能，相比其余特征列数，特征列数为43时，朴素贝叶斯高斯模型获得最好的最值、均值和方差，因此降低样本空间维度，排除大众性特征列的方法对于朴素贝叶斯模式十分有效。

任何算法都不是一个全能的表现优良的算法，比如从算法的期望来说，本例中逻辑回归算法表现出十分优良的性质，其预测成功率的均值远大于其余算法，但在样本空间维度较小的情况下，其成功率显著下降。因此，在面对不同数据和具体的问题时，我们不应该抱残守缺，而是应该及时尝试不同的算法，这样才能获得更优的结果。

# 参考文献

1. [Ken Goldberg](http://goldberg.berkeley.edu/) ,goldberg at berkeley dot edu ,Prof of IEOR and EECS ,4135 Etcheverry Hall ,University of California ,Berkeley, CA 94720-1777 . <http://eigentaste.berkeley.edu/dataset/>
2. 周志华，《机器学习》，清华大学出版社
3. 数据挖掘概念和技术，Jiawei Han,Micheline Kamber,Jian Pei 著，范明、孟小峰 译 2012.7
4. 人工神经网络：<https://en.wikipedia.org/wiki/Artificial_neural_network>