分布式机器学习算法自动选择

第一章 绪论

[研究问题的背景]

机器学习是人工智能及模式识别领域的共同研究热点，其理论和方法已被广泛应用于解决工程应用和科学领域的复杂问题。机器学习是研究怎样使用计算机模拟或实现人类学习活动的科学，是人工智能中最具智能特征，最前沿的研究领域之一。自 20 世纪 80 年代以来，机器学习作为实现人工智能的途径，在人工智能界引起了广泛的兴趣，特别是近十几年来，机器学习领域的研究工作发展很快，它已成为人工智能的重要课题之一。机器学习不仅在基于知识的系统中得到应用，而且在自然语言理解、非单调推理、机器视觉、模式识别等许多领域也得到了广泛应用。成为了一个系统是否具有学习能力已成为是否具有“智能”的一个标志。机器学习的研究主要分为两类研究方向：第一类是传统机器学习的研究，该类研究主要是研究学习机制，注重探索模拟人的学习机制；第二类是大数据环境下机器学习的研究，该类研究主要是研究如何有效利用信息，注重从巨量数据中获取隐藏的、有效的、可理解的知识。

[研究问题的挑战]

机器学习模型在许多实际应用问题中表现优异，但是使用起来比较困难。不同的机器学习算法在不同的数据集上往往性能不同，往往需要研究人员，对机器学习模型进行合理选择，才能取得较好的应用效果。这对缺少领域知识的普通用户而言，非常不友好。如何帮助普通用 户自动的从已有解决方案中选择出最佳的机器学习算法，以有效解决实际的机器学习任务，是近几年一个很重要的研究话题。

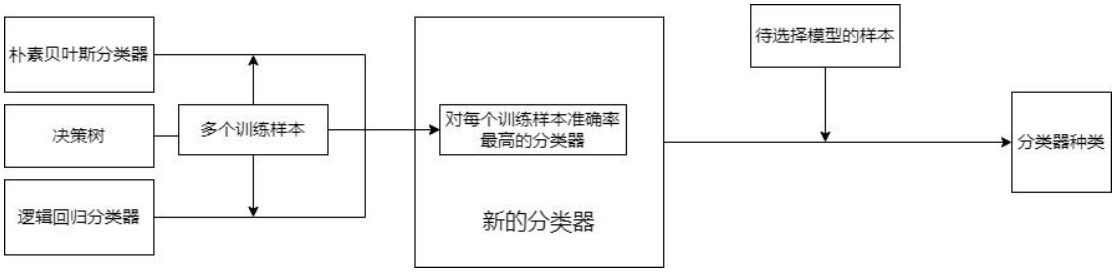
[章节安排]

第二章 系统/方法框架

设计思路

1、根据用户给定的大规模数据集（例如：分类数据集），为用户推荐出最佳的机器学习模型。

以下是我们常用的几种分类模型，给定一组数据，我们将会对这组数据进行分类模型的自动选取，选取大致方法和流程如下图所示：



环境：

本次测试在 Windows 下进行，pyspark 分布式计算平台。

第三章 技术

（1）思路：

首先针对数据集选取一些特征，为训练集的数据集手动测试出最优算法，然后建立一个个代表数据集的特征向量和对应的最优算法（label）的数据集，训练一个分类模型，用这个训练好的分类模型为新任务做算法预测，这个预测的算法作为模型自动选择的结果。

（2）分析：

1、其中数据要在算法集中找到最好的算法，其中算法集里包含以下内容：
逻辑回归分类器：

```
def logistic(train_cv,test_cv):

    lo = LogisticRegression()  ##引入逻辑回归模型
    model1 = lo.fit(train_cv)  ##训练集生成合适的模型
    predictions = model1.transform(test_cv)  ##测试集输出预测值
    print("presiction")
    predictions.show()
    ##统计测试集彼此不同的元素，计算不精确度
    t1 = predictions.select("label").collect()
    t2 = predictions.select("prediction").collect()
    t3 = predictions.count()
    print("t1")
    print(t1)
    t4 = 0
    for i in range(t3):
        if t1[i] == t2[i]:
            t4 += 1
    nbAccuracy = 1.0 * t4 / t3
    print("逻辑回归预测准确值:", nbAccuracy)
    return nbAccuracy
```

朴素贝叶斯分类器：

```
def cal_NaiveBayes(train_cv,test_cv):
    lo = NaiveBayes()  ##引入逻辑回归模型
    model1 = lo.fit(train_cv)  ##训练集生成合适的模型
    predictions = model1.transform(test_cv)  ##测试集输出预测值
    print("presiction")
    predictions.show()
    ##统计测试集彼此不同的元素，计算不精确度
    t1 = predictions.select("label").collect()
    t2 = predictions.select("prediction").collect()
    t3 = predictions.count()
    print("t1")
    print(t1)
    t4 = 0
    for i in range(t3):
        if t1[i] == t2[i]:
            t4 += 1
    nbAccuracy = 1.0 * t4 / t3
    print("朴素贝叶斯回归预测准确值:", nbAccuracy)
    return nbAccuracy
```

3、决策树

```
def cal_Decision_tree(train_cv, test_cv):
    lo = DecisionTreeClassifier()  ##引入逻辑回归模型
    model1 = lo.fit(train_cv)  ##训练集生成合适的模型
    predictions = model1.transform(test_cv)  ##测试集输出预测值
    print("presiction")
    predictions.show()
    ##统计测试集彼此不同的元素，计算不精确度
    t1 = predictions.select("label").collect()
    t2 = predictions.select("prediction").collect()
    t3 = predictions.count()
    print("t1")
    print(t1)
    t4 = 0
    for i in range(t3):
        if t1[i] == t2[i]:
            t4 += 1
    nbAccuracy = 1.0 * t4 / t3
    print("决策树预测准确值:", nbAccuracy)
    return nbAccuracy
```

将所有分类器赋予一个不同的编号，每一个分类器的返回值都为精确度，最后我们会根据精确度的高低确定数据集采用哪种方法进行分类。

2、将每个数据集放入算法集中进行计算，选择准确度最高的算法，将该算法的编号记为自己的 label。

3、提取每个数据集的特征值，形成一个新的向量，向量的最后一元素为该数据集对应的 label，也就是分类方法。因为我们选取了多组数据集，因此会产生多组向量，将生成的不同的向量合并成一个新的数据集，最后一列为每个数据集所选择的最优分类器 label 的集。

代码：（分析：用 pca 提取每个数据的的特征值并合并，写入 write 文件中，再通过 spark.read.csv 进行读取。）

```
    t1 = transformed.select("pca").collect()
    t3 = transformed.count()
    # mkdir()
    with open("write.txt", "a") as f:
        f.write(str(t1[0])[21:][:-3] + ',')
        for i in range(99):
            if (i <= t3 - 2):
                st = str(t1[i + 1])[21:][:-3]
                f.write(st + ',')
            else:
                f.write('null' + ',')
        f.write(str(class_num) + '\n')

    f.close()
test_pca = spark.read.csv('./write.txt', header=True, inferSchema=True)
handle_final = open('write.txt')
```

最后从文件中读取到的数据集如下所示：

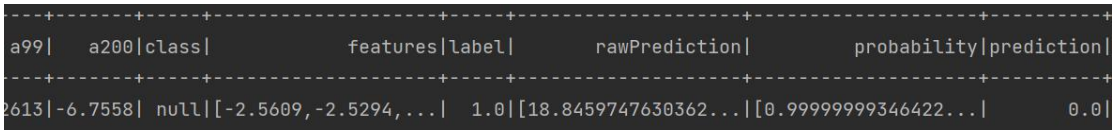
```
--------------------+-----+
            features|label|
--------------------+-----+
[7.7129,4.3325,7....|  1.0|
[-1067.0557,-1051...|  0.0|
[-1067.1561,-1051...|  1.0|
[-523.3912,-683.8...|  1.0|
[-2.532,-2.5037,-...|  0.0|
[-1.9458,-1.9663,...|  0.0|
[-7.47,-7.0581,-7...|  0.0|
```

features 为每个数据集提取特征值之后的属性集，label 为每个适配集所适配的分类器的编号。其中特征值的数量与降维前的数据集规模相同，而每个数据集的规模并不相同，因此这里我们将属性纬度大小统一为 100 维。

4、对新的数据集进行训练，将其作为训练样本，训练一个模型；选择一个新的数据集，放到模型中生成预测值，预测值即为寻找到的最合适的分类模型的编号，事实上这个过程也是一个分类过程。

```
method = logistic_result(test_pca, pca_final)
```

这里随便找了一个数据集，进行机器学习模型的选择，其中 prediction 项为 0，说明编号为 0 的机器学习模型更加适合进行该数据集的分类。



a99	a200	class	features	label	rawPrediction	probability	prediction
2613	-6.7558	null	[-2.5609,-2.5294,...	1.0	[18.8459747630362...	[0.99999999346422...	0.0

第四章 对比分析

论文 1： Auto-WEKA: Combined Selection and Hyperparameter Optimization of Classification Algorithms

现在有许多不同的机器学习算法;考虑到每个算法的超参数，总体上有大量的可能选择。这是同时选择一个学习算法和设置它的超参数的问题，不同于以往模型仅仅是独立解决这两个问题。这个模型可以通过一种完全自动化的方法来解决，利用贝叶斯优化的最新创新。具体来说，我们考虑了广泛的特征选择技术和所有在 WEKA 中实现的分类方法，涵盖了 2 种集成方法、10 种元方法、27 种基分类器，以及每个分类器的超参数设置。在 UCI 库、KDD Cup 09、MNIST 数据集的变体和 CIFAR-10 的 21 个流行数据集上，这种分类方法的分类性能通常比使用标准的选择/超参数优化方法要好得多。

模型选择与超参数优化：

优化给定学习算法的超参数λ∈Λ的问题在概念上类似于模型选择的问题。一些关键的区别是超参数通常是连续的，其空间通常是高维的，我们可以利用不同超参数设置λ1，λ2∈Λ之间的关联结构。给定 n 个超参数λ1，…， λn，域Λ1，…，Λn，超参数空间Λ是这些域的叉积的一个子集：Λ⊂Λ1×…×Λn。这个子集通常是严格的。如果将网络深度设置为 1 或 2，则决定 deep belief network 第三层细节的参数是不相关的。同样地，如果我们使用不同的核，支持向量机的多项式核的参数是无关的。

超参数优化问题可以写成:

$$\boldsymbol{\lambda}^* \in \operatorname*{argmin}_{\boldsymbol{\lambda}\in\boldsymbol{\Lambda}} \frac{1}{k}\sum_{i=1}^{n} \mathcal{L}(A_{\boldsymbol{\lambda}}, \mathcal{D}_{\text{train}}^{(i)}, \mathcal{D}_{\text{valid}}^{(i)}).$$

算法选择与超参数优化组合(cash)：

运行 SMBO 算法选取超参数

Algorithm 1 SMBO

initialise model $\mathcal{M}_L$; $\mathcal{H} \leftarrow \emptyset$
while time budget for optimization has not been exhausted **do**
  $\boldsymbol{\lambda} \leftarrow$ candidate configuration from $\mathcal{M}_L$
  Compute $c = \mathcal{L}(A_{\boldsymbol{\lambda}}, \mathcal{D}^{(i)}_{\text{train}}, \mathcal{D}^{(i)}_{\text{valid}})$
  $\mathcal{H} \leftarrow \mathcal{H} \cup \{(\boldsymbol{\lambda}, c)\}$
  Update $\mathcal{M}_L$ given $\mathcal{H}$
end while
return $\boldsymbol{\lambda}$ from $\mathcal{H}$ with minimal c

AUTO-WEKA：

为了证明一种自动解决 CASH 问题的方法的可行性，我们构建了一个工具 Auto-WEKA，它为 WEKA 包中实现的所有分类算法和特征选择器/评估器解决了这个问题。

论文 2：Auto-Model: Utilizing Research Papers and HPO Techniques to Deal with the CASH problem 分析

本文设计了 auto-model 方法，充分利用相关文献中的已知信息，并引入超参数优化技术，有效地解决 CASH 问题。auto - model 极大地降低了算法实现的成本和超参数配置空间，从而能够高效、简便地处理 CASH 问题。为了说明 auto - model 的优点，我们将其与经典的 Auto-Weka 方法进行了比较。实验结果表明，该方法能在较短时间内获得较好的检测结果。索引术语-算法选择，超参数优化，组合算法选择和超参数优化问题，Auto-weka，分类算法。

算法实现：

1)Knowledge Acquiremet：我们要选择实例特性或找到合适的模型,这需要有效的 knowledge 提取，有效的 knowledge 提取的关键是建立完整的信息网络，设计自己的最优算法的判断标准。

Algorithm 1 KnowledgeAcquisition Approach

Input: Experience obtained from n related papers $InfAll$=$\{(P_i,I,BestA_I^{P_i},OtherAs_I^{P_i})|i$=1,...,$n,I\in IList_P$

Output: Some effective knowledge $CRelations$

$IList \leftarrow$ all instances involved in $InfAll$

$PRank \leftarrow$ rank papers in $InfAll$ in ascending order of their reliability according to strategies in TABLE 1

$CRelations \leftarrow \emptyset$

for $I \in IList$ **do**

  $RInf_I \leftarrow$ tuples related to instance I in $InfAll$

  if $RInf_I$ involves > 5 algorithms **then**

    $OACs \leftarrow \{t[2]|t \in RInf_I\}$

    $Relations \leftarrow \{ (A_i,A_j,Rel_{ij}) \mid A_i,A_j\in OACs, Base_{ij}\neq\emptyset, Rel_{ij}$=max value in $Base_{ij}$ $\}$ # $Base_{ij}$= $\{PRank$.index(t[0])|t$\in RInf_I$&t[2]=A_i&$A_j\in$t[3]$\}$

    $DGraph \leftarrow$ build directed graph according to $Relations$, where $(A_i, A_j, Rel_{ij}) \in Relations$ denotes a directed edge $A_i \rightarrow A_j$ with weight Rel_{ij}

    for each node A_i in $DGraph$, start from it and apply breadth-first search. Record all nodes visited ($BFSA_i$), and the minimum weight in the path from A_i to $A_j \in BFSA_i$ ($NRel_{ij}$)

    $DGraph \leftarrow \{ (A_i,A_j,NRel_{ij}) \mid A_i\in OACs, A_j\in BFSA_i \}$ # update $DGraph$

    $DGraph \leftarrow$ if 2 nodes A_i,A_j in $DGraph$ have conflict relations, only preserve one with bigger weight

    $OACs \leftarrow$ nodes in $DGraph$ with no internal edges

    $OACs \leftarrow \{ (A_i,|ComAs_i|) \mid A_i\in OACs, ComAs_i$=$\{$t[3]|t$\in RInf_I$&t[2]$\in BFSA_i\}$ $\}$

    $OA_I \leftarrow$ an algorithm in $OACs$ with highest score

    $CRelations \leftarrow CRelations \bigcup \{(I, OA_I)\}$

  end if

end for

return $CRelations$

2)实例特征选择：在我们的自动模型方法中，我们根据给定任务实例的特征为其选择合适的算法。一个实例可能有许多可能的特征，但并不是所有特征都与算法性能相关。选择与算法性能相关的特征来表示实例，不仅可以降低特征的计算成本，而且可以帮助算法选择方法更好地区分实例，从而更加有效。

算法 2 给出了实例特征选择方法的伪代码

Algorithm 2 FeatureSelection Approach

Input: Known knowledge $CRelations$ = $\{(I_1, OA_{I_i}), \ldots, (I_t, OA_{I_t})\}$, and candidate set of instance features $Fs = \{f_1, \ldots, f_m\}$

Output: Key features KFs

$D \leftarrow \{ (Fs(I_i),OA_{I_i}) \mid (I_i,OA_{I_i})\in CRelations \}$ # $Fs(I_i)$ represents the feature vector of instance I_i, which contains all features in Fs

$PN \leftarrow$ for each feature $f_i\in Fs$, construct a boolean hyperparameter f_i, where True' ('False') means consider (ignore) feature f_i in the given dataset D.

$A \leftarrow$ a MLP classifier with default architecture and parameter setting

construct a HPO problem $P = (D, A, PN)$ # The k-fold cross-validation accuracy is used to calculate f($\lambda\in\Lambda PN$,A,D)

$OptimalConf \leftarrow$ GA(P) (group size: 50, evolutional epochs: 100)

$KFs \leftarrow \{ f_i \mid f_i\in Fs$&$OptimalConf[f_i]$='True' $\}$

return KFs

3)模型训练:基于关键实例特征 KFs 和 knowledge， DMD 训练将 KFs(I)精确映射到 OAI 的决策模型，从而帮助 UDR 做出合理的决策。

Detail Workflow。算法 3 给出了 MLP 架构搜索方法的伪代码

Algorithm 3 ArchitectureSearch Approach

Input: Known knowledge $CRelations = \{(I_1, OA_{I_i}), \ldots, (I_t, OA_{I_t})\}$, key features KFs, and $Precision$

Output: Suitable neural architecture SNA

1: $D \leftarrow \{ (KFs(I_i), OneHot'(OA_{I_i})) \mid (I_i, OA_{I_i}) \in CRelations \}$
2: $PN \leftarrow$ hyperparameters of MLP (shown in TABLE II)
3: $A \leftarrow$ a MLP regressor
4: construct a HPO problem $P = (D, A, PN)$ # The k-fold cross-validation MSE (mean squared error) is used to calculate $f(\lambda \in \Lambda PN, A, D)$
5: $OptimalConf \leftarrow GA(P)$ (group size: 50) # GA stops when $\lambda \in \Lambda PN$ whose $f(\lambda, A, D) < Precision$ is found
6: $SNA \leftarrow$ an MLP regressor with $OptimalConf$ setting
7: **return** SNA

复杂性分析：将这三个步骤有机地结合起来，得到 DMD 的全局图像，如算法 4 所示。knowledge 获取主要是基于时间复杂性的分析。

Algorithm 4 AutoModelDMD Approach

Input: Experience obtained from n related papers $InfAll = \{(P_i, I, BestA_I^{P_i}, OtherAs_I^{P_i}) | i=1,\ldots,n, I \in IList_P\}$, and candidate set of instance features $Fs = \{f_1, \ldots, f_m\}$

Output: Key features KFs, and suitable neural architecture SNA

1: $CRelations \leftarrow$ CorrespondenceAcquisition($InfAll$)
2: $KFs \leftarrow$ FeatureSelection($CRelations, Fs$)
3: $SNA \leftarrow$ ArchitectureSearch($CRelations, KFs, Precision$ = -0.0015) # we set $Precision$ to -0.0015 by default
4: $D \leftarrow \{ (KFs(I_i), OneHot'(OA_{I_i})) \mid (I_i, OA_{I_i}) \in CRelations \}$
5: $SNA \leftarrow$ train MLP regressor with SNA setting using D
6: **return** KFs, SNA

算法 5 给出了 UDR 的伪代码。自动模型的 UDR 采用

(1)由用户提供的任务实例

(2)关键特性 KFs 和一个训练过的的 MLP，具有一个合适的体系结构 SNA，这是 DMD 的结果。

Algorithm 5 AutoModelUDR Approach

Input: An instance I, key features KFs, and the suitable neural architecture SNA

Output: An optimal algorithm SA and its optimal hyperparameter setting OHS

1: $SA \leftarrow SNA(KFs(I))$ # If SA has not been implemented yet, notify the user to implement it
2: $PN \leftarrow$ the hyperparameters of SA
3: construct a HPO problem P = (I, SA, PN)
4: $OHS \leftarrow$ HPOAlg(P)
HPOAlg is BO or GA. If the calculation of $f(\lambda,SA,I)$ generally costs less than 10 minutes, then we set HPOAlg=GA, else, HPOAlg=BO
User can stop HPOAlg at any time, and OHS is the optimal configuration obtained so far
5: **return** SA, OHS

我们可以发现，论文二中的 Auto-model 方法是基于论文一的方法实现的，对于 Auto-WEKA，并不是所有的分类器都适用于所有的数据集(例如，由于分类器无法处理丢失的数据)。对于给定的数据集，我们的 Auto-WEKA 实现自动地只考虑适用分类器的子集。而 Auto-model 适用性会更加广泛一些，相比于 Auto-WEKA，Auto-model 方法可以较短时间内获得较好的检测结果。因为其引入超参数优化技术，有效地解决 CASH 问题。auto - model 极大地降低了算法实现的成本和超参数配置空间。

而我的代码由于选取的数据集过少，可能结果不够精确，但是运行成本比较二者低。

第五章 相关工作

在此次实验中，搭建 pyspark 分布式平台，学习了 pyspark 分布式计算平台的一些操作，比如数据集的合并，数据集的中某列的提取，之前了解 spark 的途径仅仅是来自课本，没有实践支持，所以也几乎是零基础学习。

熟悉了机器学习中一些常用的处理数据集的方法，比如逻辑回归，朴素贝叶斯分类器，pca 降维处理，加深机器学习的了解。

看了老师推荐的相关论文，明确自己的实验目标，虽然在有限的时间内无法完成的像论文中那么完善。

第六章 相关问题

对于需要进行模型选择的数据集，如果直接提取特征值运行程序会报错。

```
  File "<string>", line 3, in raise_from
pyspark.sql.utils.IllegalArgumentException: requirement failed: The input column stridx_c227e1f5c04d should have at least two distinct values.
```

解决方法：可以随机生成一个行向量，将其与生成的数据集特征值按列合并。

如下，将特征值写入文件后，追加一行即可

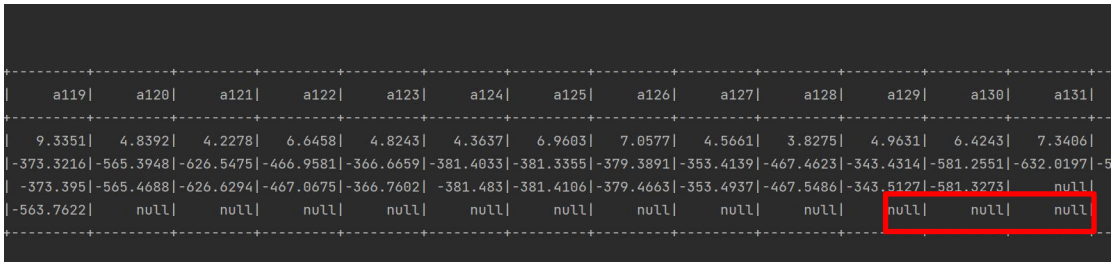
```
with open("write1.txt", "a") as f:
    # for i in range(t3):
    #     f.write(str(i + 1) + ',')
    # f.write("classfication" + '\n')
    f.write(str(t1[0])[21:][:-3])
    for i in range(99):
        if (i <= t3 - 2):
            st = str(t1[i + 1])[21:][:-3]
            f.write(',' + st)
        else:
            f.write(',' + 'null')
    f.write(','+'null'+'\n')
    for i in range(100):
        f.write(str(i)+',')
    f.write('1')
f.close()
```

报错：

```
Traceback (most recent call last):
  File "C:/Users/ASUS---BINXIAN/PycharmProjects/pythonProject4/main.py", line 167, in <module>
    method = logistic_result(test_pca, pca_final)
  File "C:\Users\ASUS---BINXIAN\PycharmProjects\pythonProject4\logistic_judge.py", line 10, in logistic_result
    predictions.show()
  File "D:\Python\Python37\lib\site-packages\pyspark\sql\dataframe.py", line 440, in show
    print(self._jdf.showString(n, 20, vertical))
  File "D:\Python\Python37\lib\site-packages\py4j\java_gateway.py", line 1322, in __call__
    answer, self.gateway_client, self.target_id, self.name)
  File "D:\Python\Python37\lib\site-packages\pyspark\sql\utils.py", line 128, in deco
    return f(*a, **kw)
  File "D:\Python\Python37\lib\site-packages\py4j\protocol.py", line 328, in get_return_value
    format(target_id, ".", name), value)
py4j.protocol.Py4JJavaError: An error occurred while calling o1655.showString.
```

原因：每个数据集提取特征值后，合并生成的新数据集默认维度太大，大于一些原数据集的规模，导致有很多维度值为 null，因此无法正常训练数据集。

如下图所示，开始默认的维度为 200



a119	a120	a121	a122	a123	a124	a125	a126	a127	a128	a129	a130	a131
9.3351	4.8392	4.2278	6.6458	4.8243	4.3637	6.9603	7.0577	4.5661	3.8275	4.9631	6.4243	7.3406
-373.3216	-565.3948	-626.5475	-466.9581	-366.6659	-381.4033	-381.3355	-379.3891	-353.4139	-467.4623	-343.4314	-581.2551	-632.0197
-373.395	-565.4688	-626.6294	-467.0675	-366.7602	-381.483	-381.4106	-379.4663	-353.4937	-467.5486	-343.5127	-581.3273	null
-563.7622	null	null	null	null	null	null	null	null	null	null	null	null

解决方法：

降低生成的新数据集的默认维度。选择所有数据集中规模最小的作为新数据集的维度。

如下图所示：这样就没有过多 null 项了。

a97	a98	a99	a100	class	features	label
578	6.9408	6.5598	8.9983	1	[7.7129,4.3325,7....	1.0
135	-429.4044	-661.4124	-407.4518	1	[-1067.0557,-1051...	1.0
268	-429.4782	-661.4876	-407.5288	1	[-1067.1561,-1051...	1.0
564	-483.5279	-883.4773	-663.7703	1	[-523.3912,-683.8...	1.0

参考文献

[1] Chunnan Wang, Hongzhi Wang, Tianyu Mu, Jianzhong Li, Hong Gao: Auto-Model: Utilizing Research Papers and HPO Techniques to Deal with the CASH problem. ICDE 2020: 1906-1909.

[2] Chris Thornton, Frank Hutter, Holger H. Hoos, Kevin Leyton-Brown: Auto-WEKA: combined selection and hyperparameter optimization of classification algorithms. KDD 2013: 847-855