

华中科技大学计算机科学与技术学院

《机器学习》 结课报告



专 业	<u>计算机科学与技术学院</u>
班 级	<u>CS2206</u>
学 号	<u>U202211111</u>
姓 名	<u>ABC</u>
成 绩	<u></u>
指导教师	<u>何琨</u>
时 间	<u>2024 年 5 月 21 日</u>

目录

1	实验要求	1
2	算法设计与实现	2
2.1	数据预处理	2
2.2	K-Nearest Neighbors	2
2.3	Multilayer Perceptron	3
2.4	Decision Tree	4
3	实验环境与平台	5
4	结果与分析	6
4.1	KNN	6
4.2	MLP	7
4.3	Decision Tree	9
4.4	对比	11
5	个人体会	12
	参考文献	13

1 实验要求

总体要求：

1. 控制报告页数，不要大段大段贴代码
2. 表格和插图请编号并进行交叉引用，表格使用三线表

2 算法设计与实现

数字识别 (digit recognizing) 是机器学习中的一个经典问题, 在本次实验中, 我选取了 **Digit Recognizer** 为任务, 通过使用 K Nearest Neighbors, Multilayer Perceptron 和 Decision Tree 三种模型, 分别对数据集进行训练, 预测, 并通过参数调优等手段, 比较不同模型在 digit recognizer 这个数据集特征下的表现。[1][2][3]

2.1 数据预处理

项目所提供的数据包括训练数据和测试数据, 分别含有 42000 和 28000 个样本。训练样本中第一列为标签 (为 0-9 中的一个数字), 第 2-785 列为数据特征, 是由 28x28 位数字按行主序的方式存储在单行中的。测试集中则无标签列, 其余列含义与训练集相同。

在数据预处理时, 首先通过将训练数据和测试数据的特征全部映射到 $[0,1]$ 值域中, 防止在之后的计算中可能因某个特征处在指数位而导致溢出。随后将训练数据划分为训练集 (training set) 和验证集 (validation set), 方便在训练时能够检测模型的准确率, 并选择较好的超参数。

2.2 K-Nearest Neighbors

K 近邻算法 (K-Nearest Neighbors) 是一个较经典的多分类机器学习算法。[4] 它是一种基于实例的算法, 即没有显式的学习过程 (没有显式的训练阶段)。该算法的核心思想是如果一个样本在特征空间中的 k 个最邻近的样本中的大多数属于一个类别, 则该样本也划分为这个类别。其伪代码如下 Algorithm 1所示。尽管该算法模型易于理解、实现, 且对异常值不敏感, 但是计算复杂度较高, 对于每个测试样本, 都需计算该样本与所有训练样本间的距离。

Algorithm 1: K Nearest Neighbors (KNN)

Input: Training dataset $D = \{(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)\}$, Test instance

x_{test} , Number of neighbors k

Output: Predicted class label for x_{test}

- 1 Compute the Euclidean distance between x_{test} and each x_i in D ;
- 2 Sort the distances in ascending order and select the k nearest neighbors;
- 3 Count the occurrences of each class label among the k nearest neighbors;
- 4 Assign the class label with the highest count as the predicted label for x_{test} ;

K 值的选取不同, 所得模型的准确率也不同。在具体实现 digit recognizer 时, 通过在 ‘训练’ 过程中分别选取不同的 K 值, 并在验证集上进行准确性验证, 最终得到最合适的 K 值。在推理过程中, 则采用该 K 值, 对测试集样本进行推理。

2.3 Multilayer Perceptron

多层感知机 (Multilayer Perceptron) 是一种前馈人工神经网络，它拥有输入层、隐藏层和输出层。[5] 它是对感知机的推广，克服了感知机对线性不可分数据进行识别的弱点。多层感知机首先学习，然后使用权重存储数据，并使用算法来调整权重并减少训练过程中的偏差，即实际值和预测值之间的误差，其主要优势在于快速解决复杂问题的能力。如下 Algorithm 2 为在本次实验中实现的 MLP 训练过程的伪代码。

Algorithm 2: MLP Training

Input: Training data X_{train} , y_{train} , Learning rate η , Number of epochs n_{epochs} ,

Show progress flag *show_progress*

Output: Trained MLP model

```

1  $train\_size \leftarrow$  number of samples in  $X_{train}$ ;
2 for  $epoch \leftarrow 1$  to  $n_{epochs}$  do
3    $y_{pred} \leftarrow []$ ;
4    $running\_loss \leftarrow []$ ;
5   for  $sample \leftarrow 1$  to  $train\_size$  do
6      $x \leftarrow X_{train}[sample]$ ;
7      $ygt\_i \leftarrow y_{train}[sample]$ ;
8      $y\_i \leftarrow self.forward(x)$ ;
9      $loss \leftarrow self.loss\_function(y\_i, ygt\_i)$ ;
10     $loss\_grad \leftarrow self.loss\_function\_grad(y\_i, ygt\_i)$ ;
11     $running\_loss.append(loss)$ ;
12     $self.backward(loss\_grad)$ ;
13     $self.step(\eta)$   $y_{pred}.append(y\_i.argmax())$ ;
14  Update metrics with  $y_{train}$  and  $y_{pred}$ 
15  if show_progress and  $epoch \% 1 == 0$  then
16    Print progress and metrics for  $epoch$ ;
17  $self.trained = True$ ;

```

模型的训练分为多个 epoch，在每个 epoch 中，首先初始化 y 预测值与损失值。然后使每个训练样本先后前向传播，计算损失，梯度，反向传播，更新权重等阶段。在每进行完一个 epoch 之后，都将该 epoch 所得 y 预测值与 y 真实值进行比较，并分别计算 **accuracy score**, **balanced_accuracy_score**, **recall_score**, **precision_score** 以及 **loss** 等指标，比较模型特征随 epoch 增加产生的变化。

2.4 Decision Tree

决策树 (Decision Tree) 是一种用于分类和回归任务的非参数监督学习算法。[6] 决策树学习采用”逐个击破”的策略，执行贪心搜索 (greedy search) 来识别决策树内的最佳分割点。然后以自上而下的回归方式重复此拆分过程，直到所有或者大多数记录都标记为特定的类别标签。决策树的布尔逻辑和可视化表示相较于神经网络等算法模型更加容易理解，但是容易过拟合，且训练成本较高。如下 Algorithm 3为在本次实验中实现的决策树的训练代码：

Algorithm 3: Decision Tree Training

Input: Training data X_{train}, y_{train}

Output: Trained decision tree

```

1  $root \leftarrow \text{TreeNode}(X_{train}, y_{train});$ 
2  $nodes \leftarrow \text{Queue}();$ 
3  $nodes.put(root);$ 
4 while  $nodes.size() > 0$  do
5      $current\_node \leftarrow nodes.get();$ 
6      $threshold, feature\_index, gain \leftarrow$ 
         $\text{find\_best\_gain}(current\_node.features, current\_node.target);$ 
7     if  $gain > 0$  then
8          $current\_node.split\_node(threshold, feature\_index, gain);$ 
9         if  $current\_node.has\_child$  then
10              $nodes.put(current\_node.left);$ 
11              $nodes.put(current\_node.right);$ 
12 return  $self;$ 

```

在训练过程中，需要在当前节点中找到某个特征值，使得当选取该属性为该决策节点时，所获得的信息增益最大。我们选择 CART3 算法来选择特征，以在简化模型的同时不完全丢失熵模型的优点。该算法采用基尼系数 (Gini impurity) 对特征进行评估。基尼系数越小，模型越好。在进行推理时，则根据决策树的每个节点的特征及相应阈值，选择应继续选择哪个子节点，最终得出结果。

3 实验环境与平台

该实验中所许软、硬件环境如下表3.1所示。

OS	Linux 5.15.0-100-generic Ubuntu SMP x86_64
CPU	Intel(R) Xeon(R) Gold 6338 CPU @ 2.00GHz
miniconda	24.5.0
slurm	23.11.4
python	3.9.18

表 3.1: 软、硬件环境

在实现各种训练模型时，所调用的 python 库包括：

- numpy 用来对数据进行代数操作
- pandas 用来读取并写入 csv 文件
- sys, os 用来读取命令行参数，读取日志文件路径
- sklearn.model_selection 用来对训练数据进行随机划分
- sklearn.preprocessing.MinMaxScaler 用来将特征映射到 $[0, 1]$ 区间内
- sklearn.metrics 用来对模型的效果进行评估
- functools, collections, queue, scipy.spatial.distance 一些数据结构及提升计算速度的算法
- matplotlib.pyplot 用来绘制模型的效果图

4 结果与分析

4.1 KNN

在进行推理前，首先依次测试选取 1-10 区间内不同 K 值时模型的准确率，然后选取准确率最高的 K 值进行推理。不同 K 值模型准确率的对比如下图4.1所示。可见当 K=4 时，模型准确率最高。当 K 值过小时，“学习”近似误差会减小，只有与输入实例较近或相似的训练实例才会对预测结果起作用，与此同时带来的问题是“学习”的估计误差会增大，换句话说，K 值的减小就意味着整体模型变得复杂，容易发生过拟合。而当 K 值过大时，可以减少学习的估计误差，但缺点是学习的近似误差会增大。这时候，与输入实例较远（不相似的）训练实例也会对预测器作用，使预测发生错误，且 K 值的增大就意味着整体的模型变得简单。

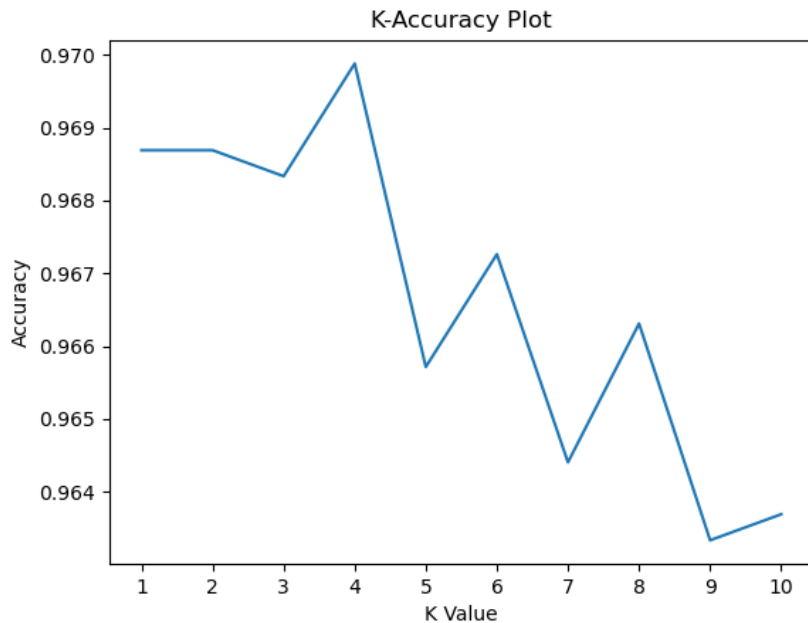


图 4.1: 不同 K 值的预测准确率对比图

于是选择 K=4，并对测试集进行推理。运行时输出的部分截图如下图4.2所示。


```
python3 main.py knn --c 4
Begin preprocessing data
Begin loading train data
X_train shape: (33600, 784)
X_validate shape: (8400, 784)
y_train shape: (33600,)
y_validate shape: (8400,)
Begin loading test data
X_test shape: (28000, 784)
Data preprocessing completed
Begin training
K: 1 Accuracy: 0.9686904761904762
K: 2 Accuracy: 0.9686904761904762
K: 3 Accuracy: 0.9683333333333334
K: 4 Accuracy: 0.9698809523809524
K: 5 Accuracy: 0.9657142857142857
K: 6 Accuracy: 0.9672619047619048
K: 7 Accuracy: 0.9644047619047619
K: 8 Accuracy: 0.9663095238095238
K: 9 Accuracy: 0.9633333333333334
K: 10 Accuracy: 0.9636904761904762
Best K: 4 Accuracy: 0.9698809523809524
Training completed
Begin inference
Inference completed
```

图 4.2: KNN 运行时输出

4.2 MLP

多层感知机在训练时, 在每个 epoch 都会对模型的准确率, 召回率, 精确率等指标进行评估, 在 20 个 epoch 完成后, 模型的训练完成, 各个指标随 epoch 的变化折线图分别如图4.3, 4.4, 4.5, 4.6 和 4.7所示。

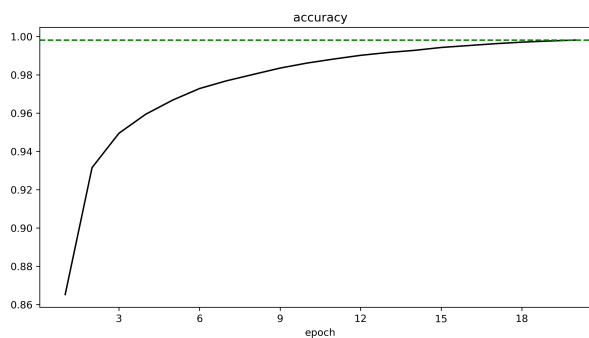


图 4.3: epoch-accuracy graph

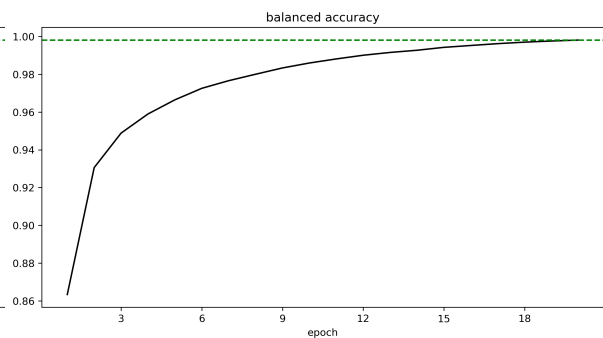


图 4.4: epoch-balanced accuracy graph

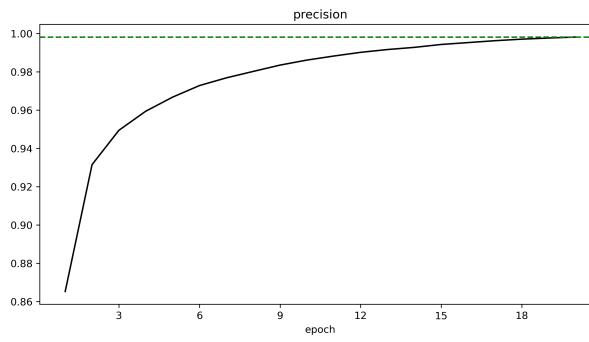


图 4.5: epoch-precision graph

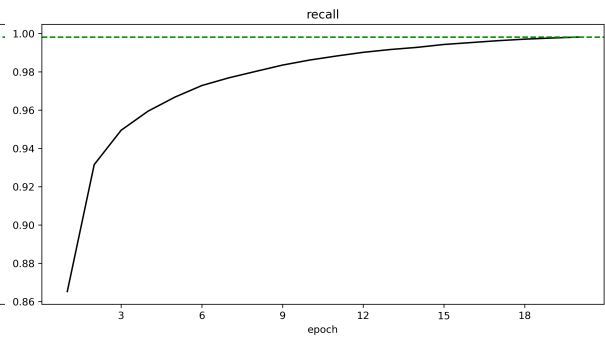


图 4.6: epoch-recall graph

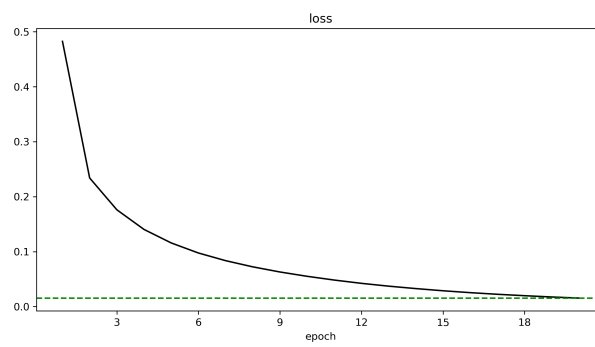


图 4.7: epoch-loss graph

模型训练时的部分输出截图如下图4.8所示（完整输出文件在 log 文件夹中）。

```
Epoch: 4/20 loss: 0.140
balanced accuracy on train: 0.959
accuracy on train: 0.959
class 0 - recall: 0.980, precision: 0.973
class 1 - recall: 0.982, precision: 0.975
class 2 - recall: 0.956, precision: 0.958
class 3 - recall: 0.941, precision: 0.952
class 4 - recall: 0.959, precision: 0.954
class 5 - recall: 0.949, precision: 0.958
class 6 - recall: 0.976, precision: 0.969
class 7 - recall: 0.965, precision: 0.965
class 8 - recall: 0.941, precision: 0.943
class 9 - recall: 0.941, precision: 0.943
Epoch: 5/20 loss: 0.116
balanced accuracy on train: 0.966
accuracy on train: 0.967
class 0 - recall: 0.982, precision: 0.979
class 1 - recall: 0.983, precision: 0.980
class 2 - recall: 0.968, precision: 0.966
class 3 - recall: 0.951, precision: 0.961
class 4 - recall: 0.967, precision: 0.962
class 5 - recall: 0.961, precision: 0.965
class 6 - recall: 0.982, precision: 0.975
class 7 - recall: 0.970, precision: 0.968
class 8 - recall: 0.953, precision: 0.956
class 9 - recall: 0.950, precision: 0.953
Epoch: 6/20 loss: 0.097
```

图 4.8: mlp 训练输出文件

通过观察折线图及输出日志，可以发现随着 epoch 增加，模型在验证集上的准确率、召回率等都迅速增加并达到饱和。

4.3 Decision Tree

根据 chapter2 描述的算法对该模型进行训练，最终得到该模型在验证集上对各个 label 的分类精准率、召回率分别如下图4.9 和 4.10所示。

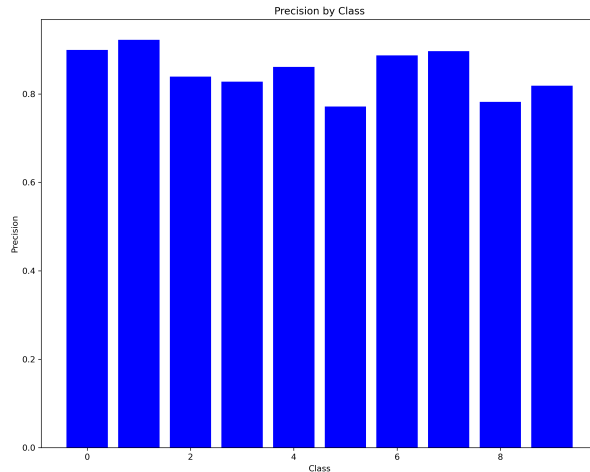


图 4.9: decision tree precision score of different labels

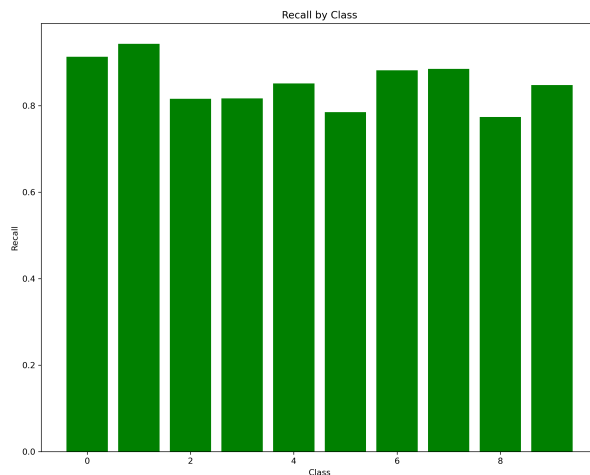


图 4.10: decision tree recall score of different labels

运行时截图如下图4.11所示。

```
Begin preprocessing data
Begin loading train data
X_train shape: (33600, 784)
X_validate shape: (8400, 784)
y_train shape: (33600,)
y_validate shape: (8400,)
Begin loading test data
X_test shape: (28000, 784)
Data preprocessing completed
Accuracy: 0.8523809523809524
Balanced Accuracy: 0.8510831321969679
Precision: [0.89975845 0.92249731 0.83941606 0.82792208 0.86127865 0.77170868
0.88717949 0.89670829 0.78208232 0.8189158 ]
Recall: [0.9129902 0.94279428 0.81560284 0.81643543 0.85101311 0.78490028
0.88152866 0.88465845 0.77365269 0.84725537]
Submission file created successfully
finished
```

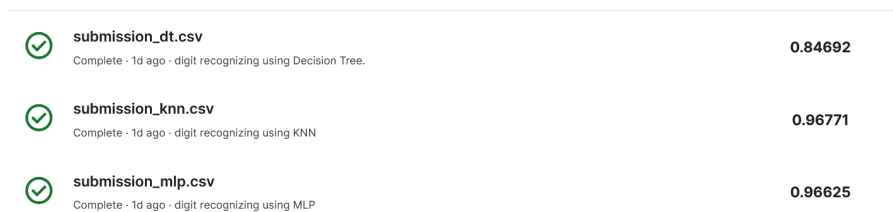
图 4.11: 决策树训练, 推理截图

观察发现, 决策树模型在验证集上的表现整体不如 KNN 和 MLP, 且对 2, 3, 5, 8 等几个 label 的预测正确率显著低于其他几个 label。经过分析, 我认为可能的原因是

- 这几个手写数字的字形较为接近
- 训练数据不平衡，这几个数字的训练样本数量较少
- 过拟合，决策树容易过拟合，记住数据中的噪声和异常点

4.4 对比

将三个模型生成的结果文件提交到 kaggle 中进行测评，结果如下图4.12所示。



✓	submission_dt.csv Complete · 1d ago · digit recognizing using Decision Tree.	0.84692
✓	submission_knn.csv Complete · 1d ago · digit recognizing using KNN	0.96771
✓	submission_mlp.csv Complete · 1d ago · digit recognizing using MLP	0.96625

图 4.12: 不同模型准确率对比

发现 KNN 和 MLP 的准确率相近，而 decision tree 的准确率显著低于二者。可能原因是没有对决策树进行剪枝操作，导致其深度过大，模型过于复杂，造成过拟合。

MLP 还可以继续 fine-tune epoch 和 hidden layers，但是由于时间原因，未进行进一步的尝试。

就训练、推理速度而言，MLP 最快，而 KNN、decision tree 含有大量数学运算，耗时较长，在实际应用中，应充分利用 CPU 多核性能，或者在 GPU 上进行训练、推理。

5 个人体会

在完成机器学习结课大作业的过程中，我使用了 KNN、MLP 和决策树这三种模型进行了 digit recognizer 任务的实现和比较。通过这个项目，我收获了许多经验和体会。

首先，我发现每种模型都有其优缺点。KNN 模型简单易懂，无需训练过程，但在大规模数据集上的计算量较大，预测速度较慢。MLP 模型能够学习到更复杂的非线性关系，具有较强的拟合能力，但需要耗费大量时间进行参数调优和训练。决策树模型易于解释和理解，可以可视化地呈现决策过程，但容易产生过拟合现象。

其次，我意识到了数据预处理的重要性。在这个项目中，我对数据进行了归一化处理，以提高模型的性能和泛化能力。合适的数据预处理可以有效地提高模型的准确率，并加快模型训练的速度。

此外，模型评估和参数调优也是非常关键的步骤。通过分析，我调整了模型的超参数，优化了模型的性能。同时，我还使用准确率、精确率、召回率等指标对模型进行了全面评估，以确保模型的稳定性和可靠性。

最后，我发现在实际应用中，模型的选择并不是一成不变的。不同的任务和数据集可能需要不同的模型来解决。因此，了解各种模型的特点和适用场景，灵活选择合适的模型是非常重要的。

通过这个机器学习大作业，我不仅加深了对 KNN、MLP 和决策树等模型的理解，还提高了数据处理和模型评估的能力，为今后从事机器学习和数据科学领域的工作打下了坚实的基础。

参考文献

- [1] 周志华. 机器学习: 第 3 章. 清华大学出版社, 2016.
- [2] Murphy, K. P. Machine learning : a probabilistic perspective , MIT Press, 2013.
- [3] Murphy, K. P. Probabilistic Machine Learning: An introduction , MIT Press, 2022.
- [4] K. Taunk, S. De, S. Verma and A. Swetapadma, "A Brief Review of Nearest Neighbor Algorithm for Learning and Classification," 2019 International Conference on Intelligent Computing and Control Systems (ICCS), Madurai, India, 2019, pp. 1255-1260, doi: 10.1109/ICCS45141.2019.9065747.
- [5] G. Singh and M. Sachan, "Multi-layer perceptron (MLP) neural network technique for offline handwritten Gurmukhi character recognition," 2014 IEEE International Conference on Computational Intelligence and Computing Research, Coimbatore, India, 2014, pp. 1-5, doi: 10.1109/ICCIC.2014.7238334.
- [6] Song YY, Lu Y. Decision tree methods: applications for classification and prediction. Shanghai Arch Psychiatry. 2015 Apr 25;27(2):130-5. doi: 10.11919/j.issn.1002-0829.215044. PMID: 26120265; PMCID: PMC4466856.