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　计算机科学与技术

班　　级　CS2202

学　　号　U202211960

姓　　名　殷绍骏

成　　绩

指导教师　何琨

时　　间　2024年5月20日

目录

1 实验背景

1.1 总体要求

1. 模型训练需自己动手实现，严禁直接调用已经封装好的各类机器学习库（包括但不限于sklearn，功能性的可以使用，比如 sklearn.model_selection.train_test_split），但可以使用numpy等数学运算库（实现后，可与已有库进行对比验证）；

2. 使用机器学习及相关知识对数据进行建模和训练，并进行相应参数调优和模型评估；

3. 可以从参考选题中选择，也可以自行选题。

1.2 选题介绍

我选取了数字识别数据集这一题目。根据Kaggle网站上对这一数据集的介绍，这是著名的 MNIST数据集，可用于计算机视觉领域的启蒙。

数据文件 train.csv 和 test.csv 包含手绘数字（从零到九）的灰度图像。每张图像的高度为 28 像素，宽度为 28 像素，总共 784 像素。每个像素都有一个与其关联的像素值，指示该像素的亮度或暗度，数字越大表示越暗。该像素值是 0 到 255 之间的整数（含 0 和 255）。

训练数据集 (train.csv) 有 785 列。第一列称为“标签”，是用户绘制的数字。其余列包含关联图像的像素值。训练集中的每个像素列都有一个类似 Pixelx 的名称，其中 x 是 0 到 783 之间的整数（含 0 和 783）。

测试数据集 (test.csv) 与训练集相同，只是它不包含“标签”列。

2 算法设计与实现

我选取了两种模型，分别是支持向量机（SVM）算法和K近邻（KNN）算法。

2.1 SVM算法

2.1.1 算法设计

SVM算法的最原始版本是处理二分类问题的。我采用一对多策略，将其扩展以满足10分类问题的需要。该数据集一共有0-9十个数字，也就是十个类别。对于每个类别，训练一个二分类器，该分类器将该类别视为正例，其他所有类别视为负例。最终的预测结果将是所有分类器的集合，选择具有最高置信度的类别作为最终预测结果。

2.1.2 算法实现

我定义了一个LinearSVM类，用于手动实现SVM算法。这个类的各个方法陈述如下：

1. 构造函数
    - 参数：learningrate 是学习率，决定了每次参数更新的步长；numepochs 是迭代次数，决定了训练的轮数；C 是正则化参数，用于控制模型的复杂度。
    - 功能：初始化线性支持向量机对象的参数。
2. fit方法
    - 参数：特征矩阵 X 和标签向量 y
    - 功能：使用梯度下降法来更新模型参数，即权重 w 和偏置 b。在每个迭代周期中，它遍历所有样本，并根据当前参数计算梯度并更新参数。
3. train方法
    - 参数：二维np数组 X 和一维np数组 y
    - 功能：训练一个具体的针对某一类别的二分类器。采用了最直接的线性核函数，对两个向量进行内积操作。
4. predict方法
    - 参数：特征矩阵 X
    - 功能：使用训练好的模型参数来进行预测。

2.2 KNN算法

2.2.1 算法设计

kNN算法属于监督学习，它通过计算新实例与训练数据集中所有实例之间的距离，对新实例进行分类或预测。其基本思想是：在特征空间中，如果一个实例的大部分近邻都属于某个类别，则该实例也属于这个类别。

KNN算法的核心步骤在于，对每个测试实例，计算它与所有训练实例的距离，并按照距离的递增关系排序。选择距离最近的K个训练实例。根据K个最近邻的多数投票结果，将测试实例分配给相应的类别。此外，多次实验，选择合适的K值，对算法的性能也至关重要。

2.2.2 算法实现

我定义了一个KNNClassifier类，用于手动实现KNN算法。这个类的各个方法陈述如下：

1. 构造函数
    - 参数：k 用于指定KNN中的邻居数量。
    - 功能：初始化KNN分类器
2. fit方法
    - 参数：X_train 训练数据的特征向量数组，y_train 训练数据的标签数组
    - 功能：用提供的训练数据训练模型，存储训练数据和标签。
3. predict方法
    - 参数：x 待预测的单个样本的特征向量。
    - 功能：对单个样本进行预测，使用KNN算法计算距离，找到最近的K个邻居，并基于它们的标签进行投票以预测该样本的标签。过程中利用了numpy库的数组、排序方法和collection库的Counter类。
4. evalue方法
    - 参数：X_test 测试数据的特征向量数组，y_test 测试数据的真实标签数组。
    - 功能：对测试数据进行预测并评估模型性能，计算准确率，并打印分类报告和混淆矩阵。
5. show方法

- 功能：展示模型的性能评估结果，包括准确率、分类报告和混淆矩阵。
- 调用库：使用了sklearn库中的三个功能性模块
  accuracy_score,classification_report,confusion_matrix

分类报告提供了关于分类器性能的详细信息，给出了准确率、精确率、召回率、F1值、支持度、宏平均以及加权平均指标。

混淆矩阵是一个表格，用于展示分类器在每个类别上的性能。它的行表示实际类别，列表示预测类别。对角线上的元素表示分类正确的样本数，而其他元素表示被错误分类的样本数。通过分析混淆矩阵，我们可以看出分类器在哪些类别上表现良好，哪些类别上存在混淆或错误分类的情况。

3 实验环境与平台

实验平台是Microsoft公司开发的Visual studio code软件，实验环境是Python3.12.2版本，使用Jupyter 扩展，在jupyter notebook上编写代码，代码保存为.ipynb文件。

我引用了一些Python第三方库辅助分析，包括numpy、pandas、matplotlib、sklearn中的功能性模块。

此外，我使用的电脑是华硕天选4，配置为64位AMD处理器和RTX4060显卡。运行的过程中开启了高性能模式，加速计算的运行。

4 结果与分析

4.1 SVM算法运行结果

将训练集划分为为80%的训练集和20%的验证集。设定好正则化参数C为1.0（一个通用的默认值，或许需要进一步调节）。之后开始训练，最后得到的准确率只有60%，运行时间大约为半小时。

```
# 使用测试集进行预测
y_pred = svm.predict(X_test)
```
[60] ✓ 28m 59.3s

```
# 精确度、分类报告和混淆矩阵
from sklearn import metrics
from sklearn.metrics import confusion_matrix,classification_report
print("accuracy:", metrics.accuracy_score(y_true=y_test, y_pred=y_pred), "\n")
print(classification_report(y_test, y_pred))
print(metrics.confusion_matrix(y_true=y_test, y_pred=y_pred))
```
[64] ✓ 0.0s

```
accuracy: 0.6059523809523809

              precision    recall  f1-score   support

           0       0.65      0.72      0.69       796
           1       0.85      0.70      0.77       946
           2       0.63      0.61      0.62       856
           3       0.80      0.24      0.37       856
           4       0.50      0.52      0.51       826
           5       0.41      0.61      0.49       773
           6       0.61      0.82      0.70       787
           7       0.79      0.72      0.75       869
           8       0.51      0.39      0.45       836
           9       0.55      0.72      0.62       855

    accuracy                           0.61      8400
   macro avg       0.63      0.61      0.60      8400
weighted avg       0.64      0.61      0.60      8400
```

图 4.1: 自行设计的线性SVM算法运行结果

使用sklearn库中的svc方法，准确率却能够达到90%，运行速度也很快，只需要80多秒。

```
# 使用sklearn库自带的SVM函数进行训练
model_linear = SVC(kernel='linear')
model_linear.fit(X_train, y_train)
y_pred = model_linear.predict(X_test)
```
✓ 1m 21.1s

```
# 精确度、分类报告和混淆矩阵
from sklearn import metrics
from sklearn.metrics import confusion_matrix,classification_report
print("accuracy:", metrics.accuracy_score(y_true=y_test, y_pred=y_pred), "\n")
print(classification_report(y_test, y_pred))
print(metrics.confusion_matrix(y_true=y_test, y_pred=y_pred))
```
✓ 0.0s

```
accuracy: 0.9188095238095239

              precision    recall  f1-score   support

           0       0.95      0.97      0.96       796
           1       0.95      0.98      0.96       946
           2       0.89      0.91      0.90       856
           3       0.88      0.91      0.89       856
           4       0.91      0.95      0.93       826
           5       0.87      0.87      0.87       773
           6       0.95      0.94      0.95       787
           7       0.94      0.94      0.94       869
           8       0.92      0.83      0.87       836
           9       0.93      0.88      0.90       855

    accuracy                           0.92      8400
   macro avg       0.92      0.92      0.92      8400
weighted avg       0.92      0.92      0.92      8400
```

图 4.2: sklearn库的线性SVM算法运行结果

4.2 KNN算法运行结果

KNN算法的k值选取是非常重要的一个环节。为了提高效率，我首先采用一个更小的数据，取训练集的前5000行数据，运用sklearn.model_selection.train_test_split方法，将其划分为80%的训练集和20%的验证集。取k值范围在3-7之间，分别进行训练和验证，调用KNNClassifier类的show方法展示分析数据，结果如下：

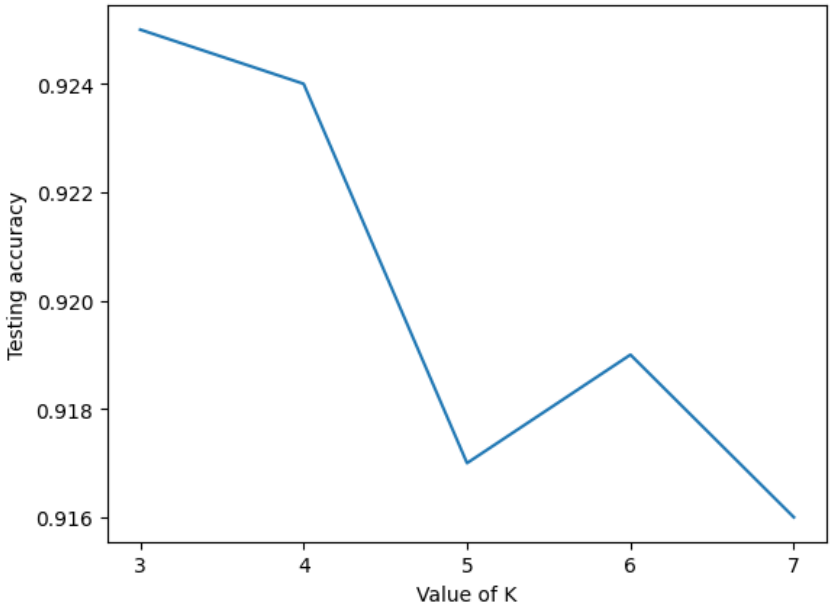

图 4.3: 准确率随k值变化的情况

可以看到，在给定的范围内，所有取值下的准确率均达到了90%以上，k值为3的时候的准确率最高。这给了我继续使用knn算法对该数据集研究的信心。

然后我取训练集的全部行数据，在k值为3的情况下，调用KNN分类器进行判别，结果如下：

```
    k=3
    knn = KNNClassifier(k)
    model = knn.fit(X_train,y_train)
    model.evalue(X_vali,y_vali)
[27] ✓ 30m 24.2s
...  accuracy:0.9707142857142858
                precision    recall  f1-score   support

             0       0.98      0.99      0.98       813
             1       0.97      0.99      0.98       961
             2       0.99      0.97      0.98       860
             3       0.97      0.97      0.97       863
             4       0.98      0.95      0.96       827
             5       0.96      0.98      0.97       756
             6       0.97      0.99      0.98       841
             7       0.96      0.98      0.97       899
             8       0.99      0.93      0.96       768
             9       0.94      0.94      0.94       812

      accuracy                           0.97      8400
     macro avg       0.97      0.97      0.97      8400
  weighted avg       0.97      0.97      0.97      8400

  [[803   0   1   1   1   2   3   0   1   1]
   [  0 956   2   1   0   0   0   1   1   0]
   [  8   4 832   1   1   0   2  10   2   0]
   [  1   0   2 839   0   7   0   5   4   5]
   [  1   6   0   0 788   0   4   1   0  27]
   [  1   0   0   7   0 739   8   0   0   1]
   [  0   1   0   0   2   2 836   0   0   0]
   [  0   8   4   1   2   0   0 877   0   7]
   [  1   7   1  11   2  17   5   0 717   7]
   [  3   2   1   3  11   3   1  21   0 767]]
```

图 4.4: 训练集运行结果

可以看到，准确率高达97%，分类报告中，精确率（Precision）、召回率（Recall）、F1等各项指标也在较为满意的范围。唯一的不足在于运行时间比较漫长。总体来看，这次训练是成功的。

既然在验证集上取得了非常满意的结果，我将其应用于测试集上，计算各个数据的标签值，并保存在submission.csv文件中。将该文件上传至kaggle网站，显示最后的准确率达到了96.7%。

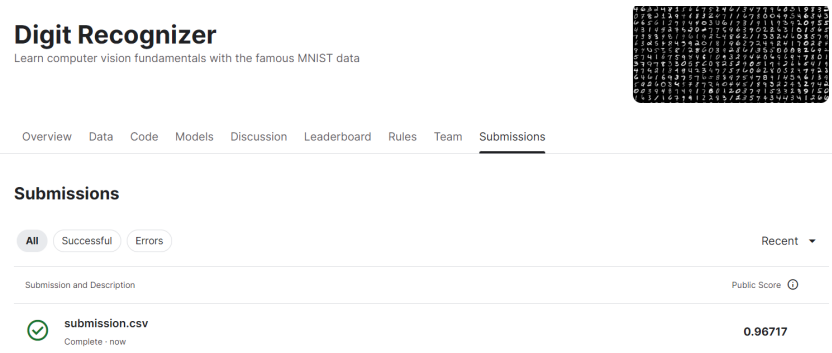

图 4.5: 测试集结果

4.3 结果分析

4.3.1 SVM算法结果分析

对比我自己实现的线性SVM算法和sklearn库中的SVC算法，两者的精确度十分悬殊。

为了分析原因，我查询sklearn的官网，发现SVC方法的C值与我自行实现的SVM算法设置的一样，默认为1.0，但它的核函数默认为RBF。这表明核函数的选择可能是影响准确度的一个重要因素。RBF核函数具有强大的非线性映射能力，能够将原始特征空间映射到高维空间，从而更好地处理非线性可分的数据。这使得它能够更好地处理复杂的实际数据集。此外，我编写的SVM类还有许多其他地方同样比较粗糙，值得改进。

4.3.2 KNN算法结果分析

相比之下，knn算法获得了很高的准确率同时，运行时间也大幅增加。对此我进行了一些原因分析。

运行时间增加的原因有：

1. 高维特征空间：MNIST数据集中的每个图像都是28x28像素的灰度图像，因此每个图像有784个特征（像素）。当特征空间维度很高时，计算距离的复杂度会增加，因为计算每个点之间的距离需要考虑更多的维度。

2. 大规模数据集：MNIST训练集有42000行数据，测试集有28000行数据。由于KNN算法需要在整个训练集上计算每个测试样本与所有训练样本之间的距离，当数据量较大的时候，计算量会很高。

3. 暴力搜索：在我设计的KNN算法中，采用暴力搜索方法来寻找最近邻居，即计算测试样本与所有训练样本之间的距离，然后选择最近的K个邻居。这种暴力搜索

方法的复杂度很高，在高维度和大规模数据集上的计算量非常大，导致运行时间长。

准确度较高的原因有：

1. 简单的特征表示：受益于选用的MNIST数据集的特点，数据集中的每个图像都是相对简单的手写数字，且分辨率较低。这意味着KNN算法可以很好地捕捉到每个数字的局部特征，并且没有太多的噪声或复杂性，使得KNN能够在特征空间中有效地寻找最近邻居。

2. 合理的K值选择：KNN算法中的K值表示用于决定最近邻居的数量。通过事先在较小数据集下验证的方法，选取了合适的k值，将其应用到大规模的整体数据当中，得到了很好的效果。

5 个人体会

这次实验，我有很多感触。首先，模型的选择对训练结果的好坏有着重要的影响。该数字识别数据集的的特征，使得采用简单易懂的K近邻算法，也能取得非常好的效果。而与此同时使用支持向量机算法，想要达到同样的效果，却要花上很大的功夫。

其次，调参是一件烦琐但极为重要的工作。机器学习的任务往往具有很强的实践性。这需要我们对数据有敏锐的察觉力和直观判断力，还要有足够的耐心。

最后，在以往的学习过程中，使用机器学习算法往往直接调用sklearn、tensorflow等现成的库。这样做有利于我们加速机器学习训练，但却缺少了对底层原理的了解与掌握。相反，本次实验的重点，却在于让我们自行动手设计机器学习算法，重复前辈的经验，感受算法设计中的不易。这是一种独特的体验。

参考文献

[1] 周 志 华. 机 器 学 习:第6章.第9章. 清 华 大 学 出 版 社, 2016. [colorlinks,linkcolor=blue]hyperref

[2] kaggle数字识别数据集 https://www.kaggle.com/c/digit-recognizer