更高效的支持向量机算法实现及其在手写数字识别中的应用——00绪论

大数据机器学习课程第二次实验项目

叶璨铭 (2024214500) ycm24@mails.tsinghua.edu.cn

2024年11月11日星期一

目录

## 1 绪论

## 2 代码与文档格式说明

本文档使用Jupyter Notebook编写，所以同时包括了实验文档和实验代码。

本次实验项目采用了 Quarto + nbdev 的系统来发布Jupyter Notebook, 因而我们的实验文档导出为pdf和html格式可以进行阅读，而我们的代码也导出为python模块形式，可以作为代码库被其他项目使用。

我们这样做的好处是，避免单独管理一堆 .py 文件，防止代码冗余和同步混乱，py文件和pdf文件都是从.ipynb文件导出的，可以保证实验文档和代码的一致性。

|  |
| --- |
| 重要 |
| 可以通过以下命令安装我们实验的代码：  pip install git+https://github.com/Open-Book-Studio/THU-Coursework-Machine-Learning-for-Big-Data.git  我们的代码导出为了python模块形式，通过以下命令导入：  from thu\_big\_data\_ml.svm import \* |

https://github.com/Open-Book-Studio/THU-Coursework-Machine-Learning-for-Big-Data.git 是我们本次大数据机器学习课程实验的代码仓库地址，

而这次作业中，我开发的另一个用于图像分类科研的开源项目[有名的分类框架 (NamableClassify)](https://github.com/2catycm/NamableClassify.git)也相应地进行了代码更新，把我们这次作业实现的SVM算法加入到了其中。接下来我们也会用到这个项目中的一些代码（比如分类评测指标）来完成本次作业。我还构建了我们课题组的基础依赖库[ScholarlyInfrastructure](https://github.com/THU-CVML/ScholarlyInfrastructure)，对标fastcore库，对AI科研经常会用到的一些基础性地、和Python语言的表达力有关的代码进行了整理，比如PyTorch模型检查、清晰的日志、实验参数管理、异常处理、argmax自动函数优化等。这里我们用到了实验参数管理功能，把SVM的超参数表达为随机变量，随即使用元参数优化算法进行搜索。

pip install git+https://github.com/2catycm/NamableClassify.git  
pip install git+https://github.com/THU-CVML/ScholarlyInfrastructure.git

from namable\_classify import \*  
from scholarly\_infrastructure import \*

以上代码库开源在github，欢迎各位同学、老师们提出宝贵意见，或者加入我们的开发一起完善，构建更加优质的科研工具。

|  |
| --- |
| 重要 |
| 本文档具有一定的交互性，建议使用浏览器打开html文件，这样比pdf文件阅读体验更佳。  由于这次项目作业内容太多，为了便于管理，我们将项目文档和代码分为了不同几个部分。 |

## 3 实验目的与项目要求

老师给我们的要求是

1. ⼿动实现一个 SVM 分类器，熟悉 SVM 的原理与优化求解 SVM 分类器的算法的过程。
2. MNIST 数据集分类，报告你在测试集上的准确率，与已有SVM库进行对比。
3. 对 SVM 分类器训练的超参数(包括收敛终止条件，学习率等)进行调优
4. 构建使用 kernel 方法的 SVM 分类器
5. 对比不同 SVM 方法

~~作为Top1大学的学生~~，我们不仅需要完成以上内容，还需要进行一些深入的思考和探索。

1. 现有最流行的SVM分类的实现是sklearn以及其背后的libsvm，由于C++编写，确实在CPU上运行很快。但是，如果我们需要在GPU上运行，是否可以考虑用CUDA等方法来加速呢？我们决定尝试一下实现速度更快的SVM分类器。
2. sklearn自带的调参GridSearchCV和RandomizedSearchCV都具有一定的局限性。参考谷歌调参手册，我们使用科学的实验设计来对SVM分类算法的元参数进行搜索，从而实现更高的分类精度，并且获得一些insight，便于我们后续科研中使用SVM。
3. 实现自己的kernel方法。实现之后，不仅像2那样参考谷歌调参手册和假设检验来比较数值上的性能，还使用可视化工具来对不同核函数的效果进行比较。
4. 除了李航《统计学习方法》的介绍逻辑，也补充阅读周志华的《机器学习》西瓜书，加深对SVM的理解。进一步阅读深度学习时代的SVM前沿论文，探索一些更加新的SVM策略，为后续科研提供参考。

事不宜迟，我们开始动手吧！

## 4 实验数据

MNIST 数据库是由 Yann et. al. 提供的⼿写数字数据库⽂件, 官网地址为 http://yann.lecun.com/exdb/mnist/。 主要包含了 60000 张的训练图像和 10000 张的测试图像

类似于上一次Project（KD树实现KNN），我们使用 sklearn CI（持续集成）测试用例的load\_digits数据集，而不只是使用原始的MNIST数据集，来加快实验的效率。并且在划分数据集时，train\_test\_split应当使用stratify参数，以确保每一类样本的比例相同。

from scholarly\_infrastructure.logging.nucleus import logger, print  
from sklearn.datasets import load\_digits, fetch\_openml

# dataset\_dict\_uci\_digits = load\_digits(as\_frame=True)  
dataset\_dict\_uci\_digits = load\_digits(as\_frame=False)  
dataset\_dict\_full\_mnist = fetch\_openml("mnist\_784", as\_frame=True)  
dataset\_dict\_uci\_digits.keys(), dataset\_dict\_full\_mnist.keys()

(  
 dict\_keys(['data', 'target', 'frame', 'feature\_names', 'target\_names', 'images', 'DESCR']),  
 dict\_keys(['data', 'target', 'frame', 'categories', 'feature\_names', 'target\_names', 'DESCR', 'details', 'url'])  
)

dataset\_dict\_uci\_digits.target\_names

array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])

[source](https://github.com/Open-Book-Studio/THU-Coursework-Machine-Learning-for-Big-Data/blob/main/thu_big_data_ml/svm/infra.py#L11)

### 4.1 sklearn\_to\_X\_y\_categories

sklearn\_to\_X\_y\_categories (dataset\_dict)

import pandas as pd  
import numpy as np

def sklearn\_to\_X\_y\_categories(dataset\_dict):  
 X = dataset\_dict['data']  
 y = dataset\_dict['target']  
 if isinstance(X, pd.DataFrame):  
 X:np.array = X.values  
 if isinstance(y, pd.Series):  
 y:np.array = y.values  
 # if y.dtype.name == 'category':  
 # categories = y.dtype.categories  
 # else:  
 X = X.astype(np.float32)  
 y = y.astype(np.int64)  
 categories = np.unique(y)  
 # print(str((X.shape, X.dtype, y.shape, y.dtype, categories)))  
 print(X.shape, X.dtype, y.shape, y.dtype, categories)  
 return X, y, categories

X, y, categories = sklearn\_to\_X\_y\_categories(dataset\_dict\_uci\_digits)  
X\_full, y\_full, categories\_full = sklearn\_to\_X\_y\_categories(dataset\_dict\_full\_mnist)

Sat 2024-11-16 22:11:02.268075

INFO ((1797, 64), dtype('float32'), (1797,), dtype('int64'), array([0, 1, 2, 3, 4, 5, 6, 7, 8, d=855392;file:///home/ye\_canming/repos/novelties/cv/ScholarlyInfrastructure/scholarly\_infrastructure/logging/nucleus.py\nucleus.py;;\:d=96894;file:///home/ye\_canming/repos/novelties/cv/ScholarlyInfrastructure/scholarly\_infrastructure/logging/nucleus.py#55\55;;\  
 9]))

Sat 2024-11-16 22:11:02.537849

INFO ((70000, 784), dtype('float32'), (70000,), dtype('int64'), array([0, 1, 2, 3, 4, 5, 6, 7, 8, d=682506;file:///home/ye\_canming/repos/novelties/cv/ScholarlyInfrastructure/scholarly\_infrastructure/logging/nucleus.py\nucleus.py;;\:d=20885;file:///home/ye\_canming/repos/novelties/cv/ScholarlyInfrastructure/scholarly\_infrastructure/logging/nucleus.py#55\55;;\  
 9]))

划分数据集为训练集和测试集。 注意这里与官方的mnist划分有所不同，但是是合理而且科学的，因为正确使用了stratify参数。

[source](https://github.com/Open-Book-Studio/THU-Coursework-Machine-Learning-for-Big-Data/blob/main/thu_big_data_ml/svm/infra.py#L34)

### 4.2 make\_train\_val\_test

make\_train\_val\_test (X, y, val\_size=0.1, test\_size=0.2, random\_state=42,  
 normalize=True)

from sklearn.model\_selection import train\_test\_split  
from sklearn.preprocessing import StandardScaler

def make\_train\_val\_test(X, y, val\_size=0.1, test\_size=0.2, random\_state=42, normalize=True):  
 X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42,   
 stratify=y)  
 # print(len(X\_train), len(X\_test))  
 if normalize:  
 scaler = StandardScaler()  
 X\_train = scaler.fit\_transform(X\_train)  
 X\_test = scaler.transform(X\_test)  
 # 进一步划分出验证集，用于调参、early stopping等。  
 X\_train, X\_val, y\_train, y\_val = train\_test\_split(X\_train, y\_train, test\_size=0.1, random\_state=42,   
 stratify=y\_train)  
 print(len(X\_train), len(X\_val), len(X\_test))  
 return X\_train, X\_val, X\_test, y\_train, y\_val, y\_test

X\_train, X\_val, X\_test, y\_train, y\_val, y\_test = make\_train\_val\_test(X, y)  
X\_train\_full, X\_val\_full, X\_test\_full, y\_train\_full, y\_val\_full, y\_test\_full = make\_train\_val\_test(X\_full, y\_full)

Sat 2024-11-16 22:11:02.748631

INFO (1293, 144, 360) d=204713;file:///home/ye\_canming/repos/novelties/cv/ScholarlyInfrastructure/scholarly\_infrastructure/logging/nucleus.py\nucleus.py;;\:d=81860;file:///home/ye\_canming/repos/novelties/cv/ScholarlyInfrastructure/scholarly\_infrastructure/logging/nucleus.py#55\55;;\

Sat 2024-11-16 22:11:07.325810

INFO (50400, 5600, 14000) d=196832;file:///home/ye\_canming/repos/novelties/cv/ScholarlyInfrastructure/scholarly\_infrastructure/logging/nucleus.py\nucleus.py;;\:d=371490;file:///home/ye\_canming/repos/novelties/cv/ScholarlyInfrastructure/scholarly\_infrastructure/logging/nucleus.py#55\55;;\

获得 PyTorch 格式 的Dataset， 进一步得到 PyTorch Lightning 的 DataModule

[source](https://github.com/Open-Book-Studio/THU-Coursework-Machine-Learning-for-Big-Data/blob/main/thu_big_data_ml/svm/infra.py#L54)

### 4.3 get\_torch\_dataset

get\_torch\_dataset (X, y)

import torch  
import lightning as L

def get\_torch\_dataset(X, y):  
 X\_tensor = torch.tensor(X, dtype=torch.float32)  
 y\_tensor = torch.tensor(y, dtype=torch.long)  
 dataset = torch.utils.data.TensorDataset(X\_tensor, y\_tensor)  
 return dataset

train\_set = get\_torch\_dataset(X\_train, y\_train)  
val\_set = get\_torch\_dataset(X\_val, y\_val)  
test\_set = get\_torch\_dataset(X\_test, y\_test)  
train\_set\_full = get\_torch\_dataset(X\_train\_full, y\_train\_full)  
val\_set\_full = get\_torch\_dataset(X\_val\_full, y\_val\_full)  
test\_set\_full = get\_torch\_dataset(X\_test\_full, y\_test\_full)

import lightning as L

data\_module = L.LightningDataModule.from\_datasets(  
 train\_dataset=train\_set,   
 val\_dataset=val\_set,   
 test\_dataset=test\_set,   
 predict\_dataset=test\_set,   
 batch\_size=128,   
 num\_workers=4  
)  
data\_module\_full = L.LightningDataModule.from\_datasets(  
 train\_dataset=train\_set\_full,   
 val\_dataset=val\_set\_full,   
 test\_dataset=test\_set\_full,   
 predict\_dataset=test\_set\_full,   
 batch\_size=128,   
 num\_workers=4  
)

[source](https://github.com/Open-Book-Studio/THU-Coursework-Machine-Learning-for-Big-Data/blob/main/thu_big_data_ml/svm/infra.py#L66)

### 4.4 process\_sklearn\_dataset\_dict

process\_sklearn\_dataset\_dict (dataset\_dict:dict,  
 return\_type:Literal['numpy','torch','lightn  
 ing','pandas','all'])

from typing import Literal

ReturnType = Literal['numpy', 'torch', 'lightning', 'pandas', 'all']  
  
def process\_sklearn\_dataset\_dict(dataset\_dict:dict, return\_type:ReturnType):  
 X, y, categories = sklearn\_to\_X\_y\_categories(dataset\_dict)  
 X\_train, X\_val, X\_test, y\_train, y\_val, y\_test = make\_train\_val\_test(X, y)  
 train\_set = get\_torch\_dataset(X\_train, y\_train)  
 val\_set = get\_torch\_dataset(X\_val, y\_val)  
 test\_set = get\_torch\_dataset(X\_test, y\_test)  
 data\_module = L.LightningDataModule.from\_datasets(  
 train\_dataset=train\_set,   
 val\_dataset=val\_set,   
 test\_dataset=test\_set,   
 predict\_dataset=test\_set,   
 batch\_size=128,   
 num\_workers=4  
 )  
 if return\_type == 'numpy':  
 return X\_train, X\_val, X\_test, y\_train, y\_val, y\_test  
 elif return\_type == 'torch':  
 return train\_set, val\_set, test\_set  
 elif return\_type == 'lightning':  
 return data\_module   
 elif return\_type == 'pandas':  
 raise NotImplementedError("Pandas not implemented yet") # 这里可以用 dataset\_dict 的 frame, 但是 train test split 还有预处理。  
 elif return\_type == 'all':  
 return X\_train, X\_val, X\_test, y\_train, y\_val, y\_test, train\_set, val\_set, test\_set, data\_module, categories  
 else:  
 raise ValueError(f"Invalid return\_type: {return\_type}")

## 5 理论回顾

### 5.1 SVM有哪些优化形式？我们选择哪种来实现代码？

参考[论文1](https://www.semanticscholar.org/paper/Deep-Learning-using-Support-Vector-Machines-Tang/96c9f11fd9901f2edeaab8cf6bbff2590cea93c4)和[论文2](https://www.researchsquare.com/article/rs-1200362/v2)，以及sklearn代码库中的分法，支持向量机（SVM）的优化方法最主要地分为两大流派：

1. 使用hinge loss（或者周志华西瓜书SVM章节提到的其他surrogate loss）对线性支持向量机软间隔原始问题进行随机梯度下降SGD（或者西瓜书提到坐标下降也可以）。
2. 使用SMO算法（或者通用的凸二次规划算法）来求解Kernel SVM的对偶问题（也可以求解Linear Kernel）。

这是两个不同形式的问题：

1. 参数保存上，（我个人认为）后者是非参数化方法，需要保存训练集中的支持向量，可能会有很多个，而前者训练成功后就是w和b，可以认为是参数化方法。
2. 前者与现代的深度学习方法非常兼容，可以端到端地优化，作为深度学习方法的一个特殊loss。而后者虽然也有很多文章将CNN预训练的feature加上SVC去做分类器，但是本身的优化问题是不一样的优化问题。前者的优化难度和神经网络类似（不是简单，是很难），后者却对数据量非常敏感，数据量大的时候会很慢（不少论文提到对大数据集使用SVM仍然是研究热点，西瓜书指出其时间复杂度理论上不可能低于O(m^2)）。
3. 后者支持核方法，而前者有论文和博客指出，SGD不可能优化Kernel Method的SVM。
4. 对于前者，[论文3](https://openreview.net/forum?id=gLwzzmh79K)说使用GD来优化的情况下，对线性可分数据集而言，hard margin SVM和Logistic Regression是等价的。

### 5.2 SVM如何实现多分类？

李航书上介绍的SVM是用于二分类问题的，然而本次项目我们需要做MNIST手写数字分类，这需要多分类，因而我们必须了解SVM如何实现多分类。

周志华西瓜书告诉我们认为普通的二分类分类器，可以通过ovr或者ovo策略来实现多分类。那么SVM本身有没有更加独特的技巧去实现多分类呢？

我们参考这个[课件](https://www.mit.edu/~rakhlin/6.883/lectures/lecture05.pdf)。

目前，构造SVM多类分类器的方法主要有两类：

1. 直接法（也就是我说的独特的方法）：直接在目标函数上进行修改，将多个分类面的参数求解合并到一个最优化问题中，通过求解该最优化问题“一次性”实现多类分类。
2. 间接法（所有二分类器都可以这么操作得到多分类器）：主要是通过组合多个二分类器来实现多分类器的构造，常见的方法有one-against-one（OvO）和one-against-all（OvR）两种。具体来说，
   * 一对多法（One-Versus-Rest, OvR）：训练时依次把某个类别的样本归为一类，其他剩余的样本归为另一类，这样k个类别的样本就构造出了k个SVM。分类时将未知样本分类为具有最大分类函数值的那类。
   * 一对一法（One-Versus-One, OvO）：将n个类别两两配对，产生n(n-1)/2个二分类任务，获得n(n-1)/2个分类器，新样本交给这些分类器，得到n(n-1)/2个结果，最终结果投票产生。

### 5.3 SVM如何实现概率输出？

尽管李航书第一章把SVM是归类为非概率模型，实际上SVM经过一定的操作，是可以得到概率输出的。经典机器学习sklearn库的学习器有一个重要的API，predict\_proba()，而sklearn中的SVC同样支持该API的调用。

那么SVM概率输出有哪些方法呢？是否要用到点和分离超平面的几何间隔呢？经过资料查询，我发现SVM的概率输出几种方法：

1. Sigmoid函数转换：可以将实轴上的数值投射到[0,1]上，即将一个输出实值抓化为一个概率值。比如一个分类器的分界线为0，大于0标为+1，小于0标为-1；如果使用sigmoid函数套一下输出值，我们就可以说，输出为0时标为+1的概率为0.5；输出为2时标为+1的概率为0.8等。
2. Platt Scaling：这是libsvm中使用的一种方法，核心思想是把分类的结果作为新的训练集，用logistics回归再训练一个关系，得到具体的概率值。
3. CalibratedClassifierCV：在scikit-learn中，可以使用专门的函数CalibratedClassifierCV对训练好的分类器模型进行校准，校准过程用到了cross-validation。

在使用predict\_proba()函数时，返回的是一个n行k列的数组，第i行第j列上的数值是模型预测第i个预测样本为某个标签的概率，并且每一行的概率和为1。

### 5.4 相关工作

类似于我们这次Project的，实现SVM的代码仓库有哪些？

教育学习目的的库：

* https://github.com/Kaslanarian/libsvm-sc-reading?tab=readme-ov-file
  + 这篇文章写得非常用心，阅读libsvm源码解读地非常详细。
* https://github.com/lzx1019056432/Be-Friendly-To-New-People/tree/master/SVM%E6%94%AF%E6%8C%81%E5%90%91%E9%87%8F%E6%9C%BA
  + 作者历时一个月，终于实现了SMO版本的SVM。但是我们做这个Project只有一周，我们要站在巨人的肩膀上实现地更好。
* https://github.com/Learner0x5a/SVM-SMO
  + 作者推导良久，终于搞懂了SVM以及SMO的公式，实现了符合sklearn接口的SVM。
* https://github.com/shicaiwei123/svm-smo?tab=readme-ov-file
  + 作者使用numpy实现了 SMO，并且使用ovo策略实现了多分类。
* https://github.com/kazuto1011/svm-pytorch
  + 作者使用早期版本的PyTorch实现了SVM。
* https://bytepawn.com/svm-with-pytorch.html
  + 这篇文章实现了二分类情况下，更早期PyTorch版本下实现的SVM。并且开源到github jupyter notebook。

实际部署在工业界的Python库：

* sklearn 本质上调用了 libsvm
  + 注意sklearn很长一段时间不会利用GPU优化SVM！ https://stackoverflow.com/questions/35292741/what-svm-python-modules-use-gpu
* https://pypi.org/project/svmlight/

非Python语言的库

* C++ libsvm https://github.com/cjlin1/libsvm
  + GPU升级版，基于CUDA/C++实现 https://mklab.iti.gr/results/gpu-accelerated-libsvm/
* Rust https://athemathmo.github.io/rusty-machine/doc/rusty\_machine/learning/svm/index.html
* Rust https://docs.rs/linfa-svm/latest/linfa\_svm/

可以看出这次Project的难度不小，我们需要对SVM的原理有深刻的理解，而且需要有较强的工程能力，才能手动实验一个SVM。

## 6 实验内容

### 6.1 分类的评价指标

开尔文爵士曾说，如果你连measure都做不到，谈不上improve。所以我们先measure。

from namable\_classify.metrics import compute\_classification\_metrics

compute\_classification\_metrics?

Signature:  
compute\_classification\_metrics(  
 y\_true: numpy.ndarray,  
 y\_pred\_logits: numpy.ndarray = None,  
 logits\_to\_prob: bool = False,  
 y\_pred: numpy.ndarray = None,  
 labels: list[int | str] | None = None,  
 supress\_warnings: bool = True,  
 y\_pred\_metrics\_only: bool = False,  
)  
Docstring: <no docstring>  
File: ~/repos/novelties/cv/cls/NamableClassify/namable\_classify/metrics.py  
Type: function

这里涵盖了 from sklearn.metrics import roc\_auc\_score, top\_k\_accuracy\_score, matthews\_corrcoef, f1\_score, precision\_score, recall\_score, log\_loss, balanced\_accuracy\_score, cohen\_kappa\_score, hinge\_loss, accuracy\_score 当结果有意义的时候，这些指标都会被计算，用于评价模型的精度。

### 6.2 调库实现SVM

为了给我们后面的实验一个参照，我们调用现有代码库的SVM，关注其精度与速度的情况。 当然如果我们Project在此收尾，只能酌情被扣除分数。 在本节之后，我们将使用 PyTorch 和 numpy 这样的基础科学计算库，来在GPU和CPU上实现SVM及其优化。

|  |
| --- |
| 重要 |
| 本次Project首先展示了几个常用的SVM库的精度与速度，并且对其进行调参；随后本次Project基于基础科学计算库手写实现了SVM及其优化，和前面的库的精度与速度进行了对比。 |

接下来的内容请见文件 [01sv\_use\_lib](./01sv_use_lib.html)

### 6.3 实现 Hinge Loss+SGD 版本的 Soft Margin Linear SVM

我们现在来实现与from sklearn.linear\_model import SGDClassifier等价的 SVM，但是我们基于PyTorch实现，在GPU上面运行，期望能在大型数据集上比sklearn的实现快。

这部分内容请见文件 [02svm\_handy\_crafted\_linear](./02svm_handy_crafted_linear.html)

### 6.4 对手动实现的SVM进行调参

这部分内容请见文件 [02svm\_handy\_crafted\_linear](./02svm_handy_crafted_linear.html)

### 6.5 附加题: 对比不同 kernel 方法下的 SVM 分类器 （对完整SVM进行调参）

这一题本质上是让我们以 kernel 的选择（也包括选择线性Kernel）作为目标元参数，其他参数作为冗余或固定元参数，进行调参实验，发现不同 kernel 方法下的 SVM 分类器的分类效果数值上的区别及其显著性，并且从可视化分析上也作出进一步解释。

这部分内容请见 [03svm\_kernel\_hpo](./03svm_kernel_hpo.html)

### 6.6 附加题: 构建使用 kernel 方法的 SVM 分类器 （手动实现SMO）

这也就是让我们手动实现 SMO 优化算法以及Kernel Method。

这部分内容请见 [04svm\_handy\_crafted\_kernel](./04svm_handy_crafted_kernel.html)