神经网络

刘德华

2023 年 7 月 18 日

例子 1

MLP 分类器的随机学习策略比较[a]。

[a]实现的程序见例1

例子 2

MLP 分类器的权重可视化[a]。

[a]实现的程序见例1

例子 3

MLP 回归器在糖尿病数据集上的表现[a]。

[a]实现的程序见例1

1 代码

Python 代码 1

```
# 导入操作系统库
import os
# 更改工作目录
os.chdir(r"D:\softwares\applied statistics\pythoncodelearning\chap10\sourcecode")
# 导入警告库
import warnings
# 导入基础计算库
import numpy as np
# 导入绘图库
import matplotlib.pyplot as plt
# 导入数据集工具
from sklearn import datasets
# 导入收敛警告工具
from sklearn.exceptions import ConvergenceWarning
# 导入 MLP 分类器
from sklearn.neural_network import MLPClassifier
# 导入最大最小归一化工具
from sklearn.preprocessing import MinMaxScaler
# 导入绘图库中的字体管理包
from matplotlib import font_manager
# 实现中文字符正常显示
font = font_manager.FontProperties(fname=r"C:\Windows\Fonts\SimKai.ttf")
```

```
# 使用 seaborn 风格绘图
plt.style.use("seaborn-v0_8")
# 设置不同模型下的参数
params = [
    {
        "solver": "sgd",
        "learning_rate": "constant",
        "momentum": 0,
        "learning_rate_init": 0.2,
    },
    {
        "solver": "sgd",
        "learning_rate": "constant",
        "momentum": 0.9,
        "nesterovs_momentum": False,
        "learning_rate_init": 0.2,
    },
    {
        "solver": "sgd",
        "learning_rate": "constant",
        "momentum": 0.9,
        "nesterovs_momentum": True,
        "learning_rate_init": 0.2,
    },
    {
        "solver": "sgd",
        "learning_rate": "invscaling",
        "momentum": 0,
        "learning_rate_init": 0.2,
    },
    {
        "solver": "sgd",
        "learning_rate": "invscaling",
        "momentum": 0.9,
        "nesterovs_momentum": True,
        "learning_rate_init": 0.2,
    },
    {
        "solver": "sgd",
        "learning_rate": "invscaling",
        "momentum": 0.9,
        "nesterovs_momentum": False,
        "learning_rate_init": 0.2,
```

```
    },
    {"solver": "adam", "learning_rate_init": 0.01},
]
# 表爱你
labels = [
    "constant learning-rate",
    "constant with momentum",
    "constant with Nesterov's momentum",
    "inv-scaling learning-rate",
    "inv-scaling with momentum",
    "inv-scaling with Nesterov's momentum",
    "adam",
]
# 绘图参数
plot_args = [
    {"c": "red", "linestyle": "-"},
    {"c": "green", "linestyle": "-"},
    {"c": "blue", "linestyle": "-"},
    {"c": "red", "linestyle": "--"},
    {"c": "green", "linestyle": "--"},
    {"c": "blue", "linestyle": "--"},
    {"c": "black", "linestyle": "-"},
]
# 绘制数据散点
def plot_on_dataset(X, y, ax, name):
    # for each dataset, plot learning for each learning strategy
    print("\nlearning on dataset %s" % name)
    ax.set_title(name)

    X = MinMaxScaler().fit_transform(X)
    mlps = []
    if name == "digits":
        # digits is larger but converges fairly quickly
        max_iter = 15
    else:
        max_iter = 400

    for label, param in zip(labels, params):
        print("training: %s" % label)
        # 构造 MLP 分类器模型
        mlp = MLPClassifier(random_state=0, max_iter=max_iter, **param)
        with warnings.catch_warnings():
            # 忽视警告
```

```
            warnings.filterwarnings(
                "ignore", category=ConvergenceWarning, module="sklearn"
            )
            # 模型拟合
            mlp.fit(X, y)
        # 将模型对象存储到列表中
        mlps.append(mlp)
        # 输出模型的预测准确率
        print("Training set score: %f" % mlp.score(X, y))
        # 模型的损失函数值
        print("Training set loss: %f" % mlp.loss_)
    for mlp, label, args in zip(mlps, labels, plot_args):
        # 绘制模型的损失函数曲线
        ax.plot(mlp.loss_curve_, label=label, **args)

# 开始绘图
fig, axes = plt.subplots(2, 2, figsize=(15, 10))
# 加载鸢尾属数据集
iris = datasets.load_iris()
X_digits, y_digits = datasets.load_digits(return_X_y=True)
data_sets = [
    (iris.data, iris.target),
    (X_digits, y_digits),
    datasets.make_circles(noise=0.2, factor=0.5, random_state=1),
    datasets.make_moons(noise=0.3, random_state=0),
]

for ax, data, name in zip(
    axes.ravel(), data_sets, ["iris", "digits", "circles", "moons"]
):
    # 建模，绘图
    plot_on_dataset(*data, ax=ax, name=name)

fig.legend(ax.get_lines(), labels, ncol=3, loc="upper center")
plt.show()
fig.savefig("../codeimage/code1.pdf")


learning on dataset iris
training: constant learning-rate
Training set score: 0.980000
Training set loss: 0.096950
training: constant with momentum
Training set score: 0.980000
```

```
Training set loss: 0.049530
training: constant with Nesterov's momentum
Training set score: 0.980000
Training set loss: 0.049540
training: inv-scaling learning-rate
Training set score: 0.360000
Training set loss: 0.978444
training: inv-scaling with momentum
Training set score: 0.860000
Training set loss: 0.503452
training: inv-scaling with Nesterov's momentum
Training set score: 0.860000
Training set loss: 0.504185
training: adam
Training set score: 0.980000
Training set loss: 0.045311

learning on dataset digits
training: constant learning-rate
Training set score: 0.956038
Training set loss: 0.243802
training: constant with momentum
Training set score: 0.992766
Training set loss: 0.041297
training: constant with Nesterov's momentum
Training set score: 0.993879
Training set loss: 0.042898
training: inv-scaling learning-rate
Training set score: 0.638843
Training set loss: 1.855465
training: inv-scaling with momentum
Training set score: 0.912632
Training set loss: 0.290584
training: inv-scaling with Nesterov's momentum
Training set score: 0.909293
Training set loss: 0.318387
training: adam
Training set score: 0.991653
Training set loss: 0.045934

learning on dataset circles
training: constant learning-rate
Training set score: 0.840000
```

```
Training set loss: 0.601052
training: constant with momentum
Training set score: 0.940000
Training set loss: 0.157334
training: constant with Nesterov's momentum
Training set score: 0.940000
Training set loss: 0.154453
training: inv-scaling learning-rate
Training set score: 0.500000
Training set loss: 0.692470
training: inv-scaling with momentum
Training set score: 0.500000
Training set loss: 0.689143
training: inv-scaling with Nesterov's momentum
Training set score: 0.500000
Training set loss: 0.689751
training: adam
Training set score: 0.940000
Training set loss: 0.150527

learning on dataset moons
training: constant learning-rate
Training set score: 0.850000
Training set loss: 0.341523
training: constant with momentum
Training set score: 0.850000
Training set loss: 0.336188
training: constant with Nesterov's momentum
Training set score: 0.850000
Training set loss: 0.335919
training: inv-scaling learning-rate
Training set score: 0.500000
Training set loss: 0.689015
training: inv-scaling with momentum
Training set score: 0.830000
Training set loss: 0.512595
training: inv-scaling with Nesterov's momentum
Training set score: 0.830000
Training set loss: 0.513034
training: adam
Training set score: 0.930000
Training set loss: 0.170087
```

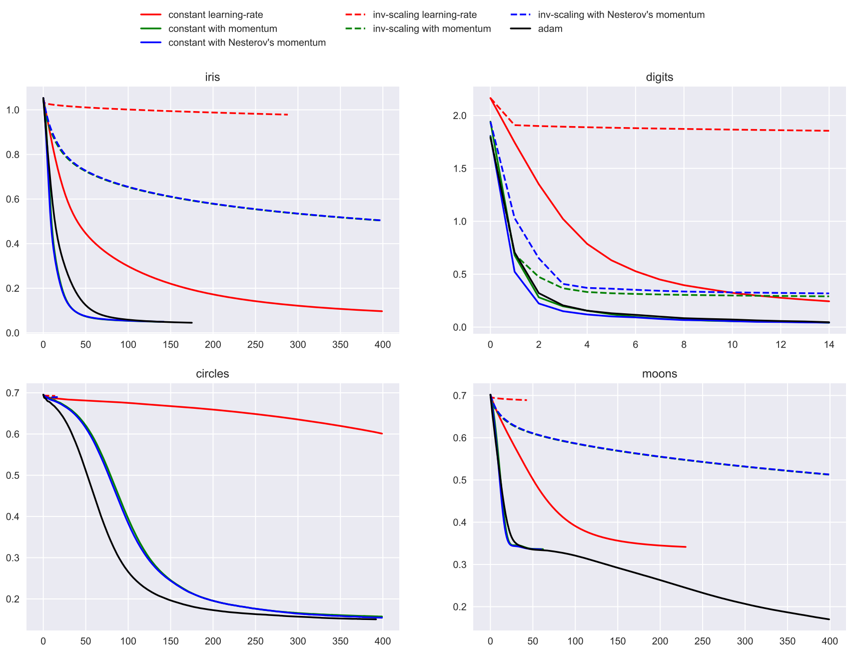

图 1: code1

Python 代码 2

```
# 导入操作系统库
import os
# 更改工作目录
os.chdir(r"D:\softwares\applied statistics\pythoncodelearning\chap10\sourcecode")
# 导入警告库
import warnings
# 导入基础计算库
import numpy as np
# 导入绘图库
import matplotlib.pyplot as plt
# 导入收敛警告工具
from sklearn.exceptions import ConvergenceWarning
# 导入 MLP 分类器
from sklearn.neural_network import MLPClassifier
# 导入数据集工具
from sklearn.datasets import fetch_openml
# 导入数据集划分工具
from sklearn.model_selection import train_test_split
# 导入绘图库中的字体管理包
from matplotlib import font_manager
# 实现中文字符正常显示
font = font_manager.FontProperties(fname=r"C:\Windows\Fonts\SimKai.ttf")
```

```
# 使用 seaborn 风格绘图
plt.style.use("seaborn-v0_8")
# 加载数据
X, y = fetch_openml(
    "mnist_784", version=1, return_X_y=True, as_frame=False, parser="pandas"
)
X = X / 255.0
# 划分数据集
X_train, X_test, y_train, y_test = train_test_split(
    X, y, random_state=0, test_size=0.7
)
# 构建 MLP 分类器
mlp = MLPClassifier(
    hidden_layer_sizes=(40, ), # 隐藏层的个数
    max_iter=8, # 最大迭代次数
    alpha=1e-4,
    solver="sgd",
    verbose=10,
    random_state=1,
    learning_rate_init=0.2,
)
# 忽视警告
with warnings.catch_warnings():
    warnings.filterwarnings("ignore", category=ConvergenceWarning,␣
↪module="sklearn")
    # 拟合模型
    mlp.fit(X_train, y_train)
# 训练集上和测试集上模型的分类准确率
print("Training set score: %f" % mlp.score(X_train, y_train))
print("Test set score: %f" % mlp.score(X_test, y_test))
# 开始绘图
fig, axes = plt.subplots(4, 4)
# 可视化模型的权重系数
vmin, vmax = mlp.coefs_[0].min(), mlp.coefs_[0].max()
for coef, ax in zip(mlp.coefs_[0].T, axes.ravel()):
    ax.matshow(coef.reshape(28, 28), cmap=plt.cm.gray, vmin=0.5 * vmin, vmax=0.5 *␣
↪vmax)
    ax.set_xticks(())
    ax.set_yticks(())

plt.show()
fig.savefig("../codeimage/code2.pdf")

Iteration 1, loss = 0.44139186
```

```
Iteration 2, loss = 0.19174891
Iteration 3, loss = 0.13983521
Iteration 4, loss = 0.11378556
Iteration 5, loss = 0.09443967
Iteration 6, loss = 0.07846529
Iteration 7, loss = 0.06506307
Iteration 8, loss = 0.05534985
Training set score: 0.986429
Test set score: 0.953061
```

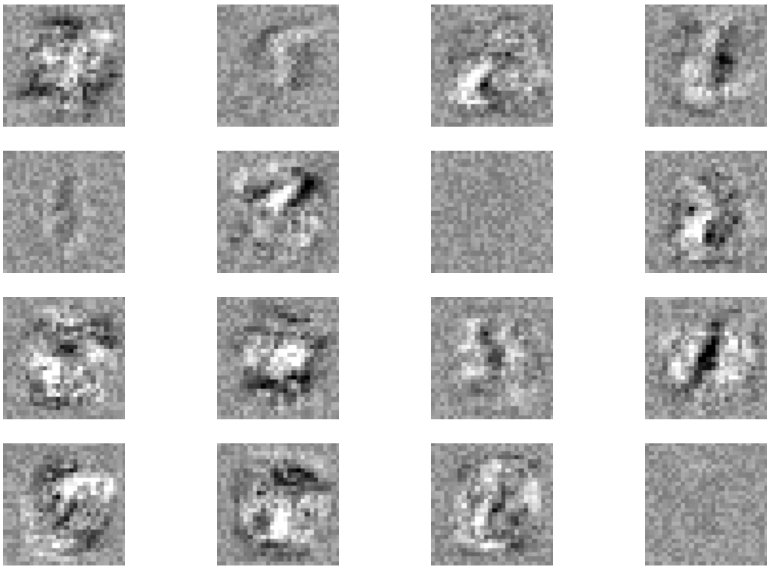

图 2: code2

Python 代码 3

```
# 导入操作系统库
import os
# 更改工作目录
os.chdir(r"D:\softwares\applied statistics\pythoncodelearning\chap10\sourcecode")
# 导入基础计算库
import numpy as np
# 导入绘图库
import matplotlib.pyplot as plt
# 导入 MLP 分类器
from sklearn.neural_network import MLPRegressor
# 导入数据集工具
```

```
from sklearn.datasets import load_diabetes
# 导入数据集划分工具
from sklearn.model_selection import train_test_split
# 导入绘图库中的字体管理包
from matplotlib import font_manager
# 实现中文字符正常显示
font = font_manager.FontProperties(fname=r"C:\Windows\Fonts\SimKai.ttf")
# 使用 seaborn 风格绘图
plt.style.use("seaborn-v0_8")
# 加载数据
X, y = load_diabetes(return_X_y=True)
# 划分数据集
x_train, x_test, y_train, y_test = train_test_split(
    X, y, random_state=0, test_size=0.3
)
# 构建 MLP 回归模型
mlpr = MLPRegressor()
# 模型拟合
mlpr.fit(x_train, y_train)
# 预测
y_pred = mlpr.predict(x_test)
# MSE
mse = np.mean((y_pred-y_test)**2)
print("测试集上的 MSE 为", mse, sep="\n")
# 绘图
fig, ax = plt.subplots(figsize=(6,6))
ax.plot(y_test, y_pred, "ro", alpha=0.4)
plt.show()
fig.savefig("../codeimage/code3.pdf")
```

```
测试集上的 MSE 为
22985.621957134874
```

图 3: code3