Experiement 3

孟昶-2021214431

November 11, 2021

Task:

Task1: Visualize the data and try to use an ensemble model to classify.

Task2: Image segmentation via clustering.

Goal:

We hope you can better know about the tSNE or other methods to project the high dimension data to a 2D plane and know better about the ensemble model.

Data:

For CS background students, you should use the attached dataset file.

For Non-CS background students, you can use the sklearn package dataset to load the MNIST dataset.

For image segmentation task, check the data file brainimage.txt

Experiment Step:

- Use the tSNE or PCA to visualize the data.
- Do some EDA (Explore Data Analyze) work to analyze the data. There is an example.
- Choose one ensemble model (like XGboost/Adaboost/Random Forest) to train and test on the dataset.

Try to change the related parameters to see the change of metric you use. Record the result and try to explain it.

- Compare the result with experiment 1 and experiment 2. Give your conclusion about model choosing and parameter setting.
- (Optional, extra score for bonus) There is an imbalance between the data of different labels. Try to solve this problem by using some tricks. (Hint: you can refer to focal loss)
- There is an image of a brain in the data file brainimage.txt. There are 151 rows in the data file and 171 columns, with each data point an integer value. The image is shown below. Your task is to use EM to classify the pixels into three classes (outside brain, gray matter, white matter).

Start by normalizing the pixel values from 0 to 1. Start your search with the following parameters (for the variance parameter, set it to the sampled variance of the entire data set). Your system should converge in about 20-30 steps. Show the log-likelihood of your model. Classify each pixel based on your posterior probability after the final step and plot three images to show each class.

$$P(Z_1 = 1) = P(Z_2 = 1) = P(Z_3 = 1) = \frac{1}{3}$$

$$\mu_1 = 0.3, \mu_2 = 0.5, \mu_3 = 0.7$$

$$\sum_1 = \sum_2 = \sum_3$$

Answer:

1 PCA

1.1 理论基础

主成分分析 (PCA) 是一种统计技术，用于通过选择最重要的特征来减少数据的维数（减少数据集中的特征数量），这些特征捕获了有关数据集的大部分信息。

PCA 通过查找数据的主成分（数据中方差最大的方向）并将其投影到较小维度的子空间中，同时保留大部分信息来进行操作。通过将我们的数据投影到一个更小的空间，我们减少了特征空间的维度。引起最大方差的特征是第一个主成分。负责第二大方差的特征被认为是第二主成分，依此类推。

简而言之，PCA 是一种从数据集中可用的大量特征中提取重要特征（以组件的形式）的方法。

PCA 的具体步骤如下：

1. 预处理数据集，即标准化数据集。
2. 求数据集的协方差，假设其为 S。
3. 求协方差矩阵的特征值和特征向量。
4. 求特征向量的点积和协方差矩阵。

1.2 可视化

如图1所示为 MNIST 数据集经过 PCA 降维后的结果分布图。

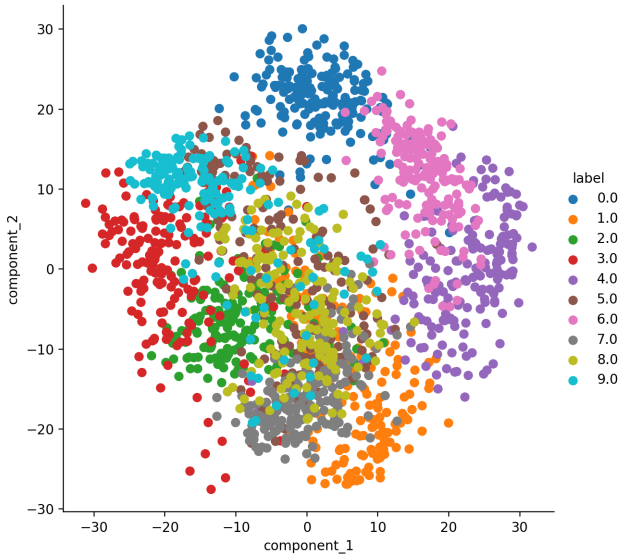

Figure 1: PCA 可视化

如图2所示为 MNIST 数据集的累计方差贡献值曲线。

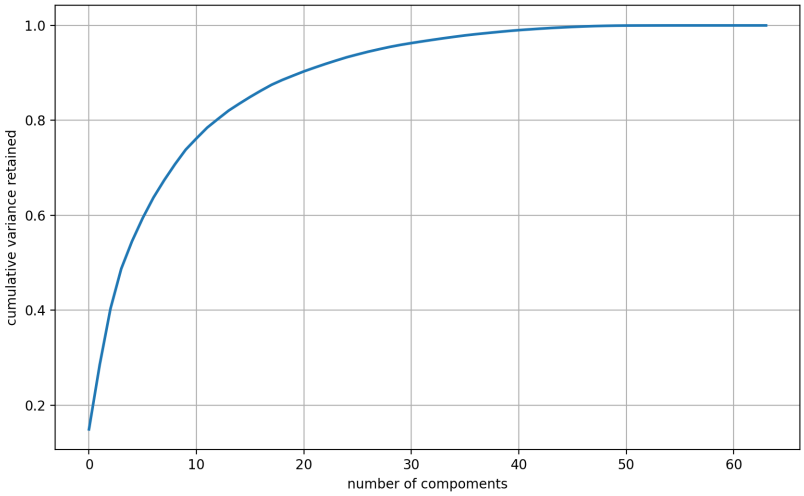

Figure 2: PCA 累计方差贡献值曲线

1.3 代码:

```
digits = load_digits()

target = {"outside brain": 0, "gray matter": 1, "white matter": 2}
pca = PCA()
pca.n_components = 2
pca_data = pca.fit_transform(digits.data)
pca_data = np.vstack((pca_data.T, digits.target)).T
pca_df = pd.DataFrame(data=pca_data, columns=("component_1", "component_2", "label"))
```

```
sns.FacetGrid(pca_df, hue="label", size=6).map(
    plt.scatter, "component_1", "component_2"
).add_legend()
plt.show()
```

2 EDA

首先，利用 sklearn 载入数据后，对数据进行预处理。

```
digits = load_digits()
head = []
for i in range(digits.data.shape[1]):
    head.append("pixel" + str(i))
head.append("label")
data = pd.DataFrame(digits.data)
tar = pd.Series(digits.target)
res = pd.concat([data, tar], axis=1)
res.columns = head
```

打印查看数据的详细信息：

```
print(res.head())
print(res.info())
print("\n SHape of the dataset:", res.shape)
```

判断是否有 nan 存在：

```
nan = res.isnull().sum()
print(nan[nan != 0])
```

结果如图3所示。

```
   pix0  pix1  pix2  pix3  pix4  pix5  pix6  ...  pix58  pix59  pix60  pix61  pix62  pix63  label
0   0.0   0.0   5.0  13.0   9.0   1.0   0.0  ...    6.0   13.0   10.0    0.0    0.0    0.0      0
1   0.0   0.0   0.0  12.0  13.0   5.0   0.0  ...    0.0   11.0   16.0   10.0    0.0    0.0      1
2   0.0   0.0   0.0   4.0  15.0  12.0   0.0  ...    0.0    3.0   11.0   16.0    9.0    0.0      2
3   0.0   0.0   7.0  15.0  13.0   1.0   0.0  ...    7.0   13.0   13.0    9.0    0.0    0.0      3
4   0.0   0.0   0.0   1.0  11.0   0.0   0.0  ...    0.0    2.0   16.0    4.0    0.0    0.0      4

[5 rows x 65 columns]
```

Figure 3: 前五行具体分布数值

统计每种数字的分布情况：

```
_ = res["label"].value_counts().plot(kind="bar")
plt.show()
```

结果如图4所示。

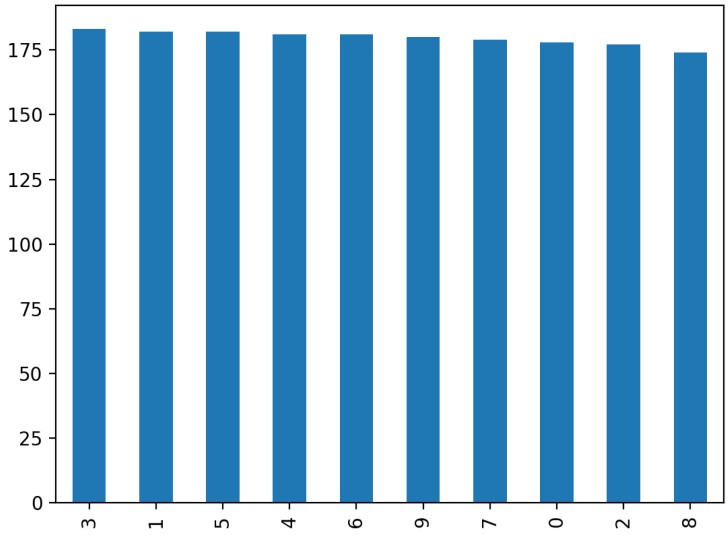

Figure 4: 每种数字的分布数

之后随机取其中 40 个样本进行可视化:

```
N = 40
images = np.random.randint(low=0, high=1000, size=N).tolist()
subset_images = res.iloc[images, :]
subset_images.index = range(1, N + 1)
print("Handwritten picked-up digits: ", subset_images["label"].values)
subset_images.drop(columns=["label"], inplace=True)

for i, row in subset_images.iterrows():
    plt.subplot((N // 8) + 1, 8, i)
    pixels = row.values.reshape((8, 8))
    plt.imshow(pixels, cmap="gray")
    plt.xticks([])
    plt.yticks([])
plt.show()
```

结果如图5所示。

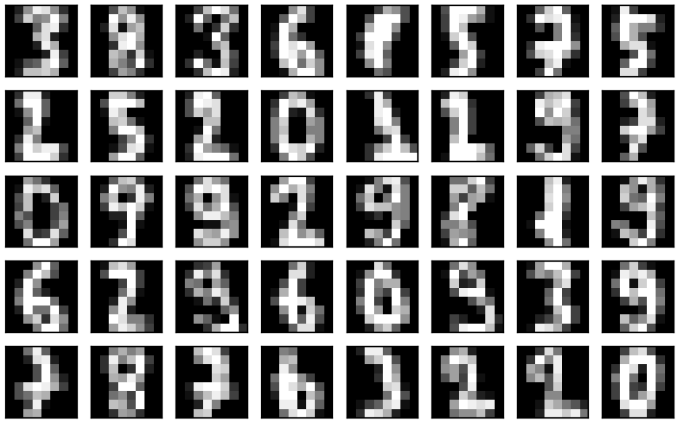

Figure 5: 随机 40 个样本可视化

查看特征之间的相关系数矩阵：

```
for i in range(4):
    df_corr = res.iloc[:, 16 * i : 16 * (i + 1)].corr()
    sns.heatmap(df_corr, center=0, annot=True, cmap="YlGnBu")
    plt.show()
```

结果如图6、7、8、9所示。

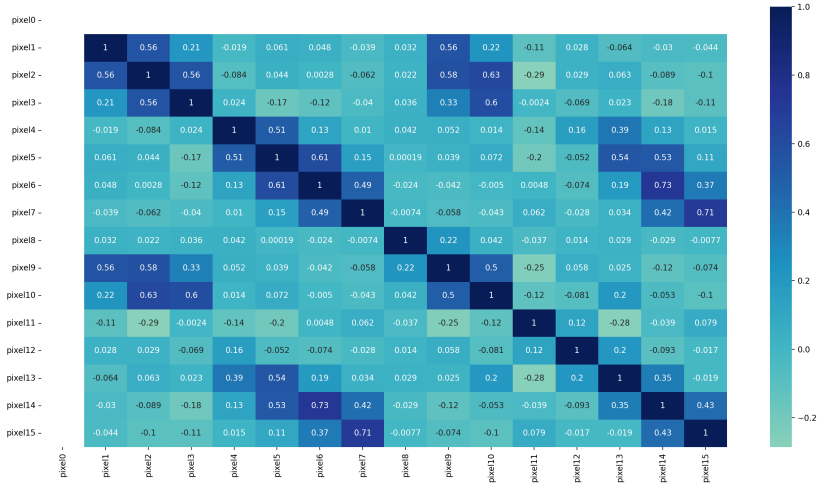

Figure 6: 1-16 列数据之间的相关系数矩阵

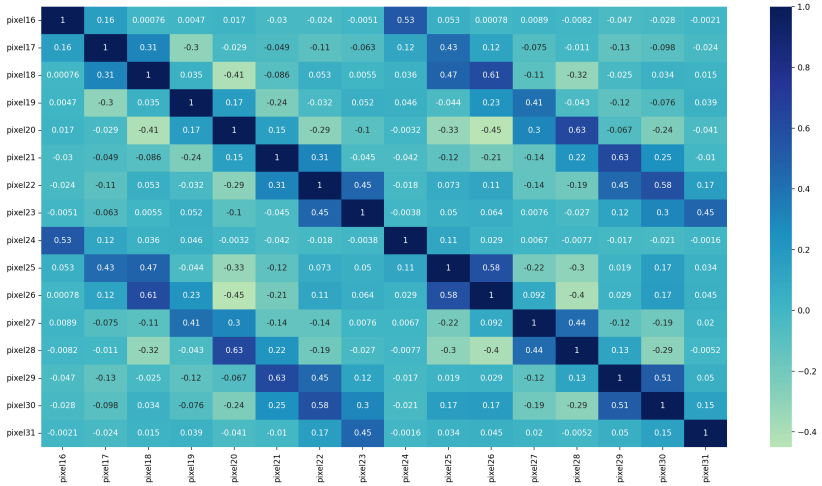

Figure 7: 17-32 列数据之间的相关系数矩阵

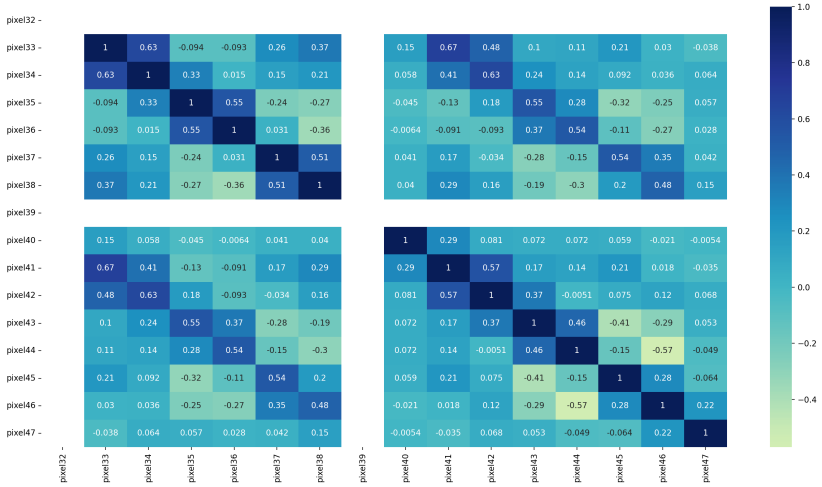

Figure 8: 33-48 列数据之间的相关系数矩阵

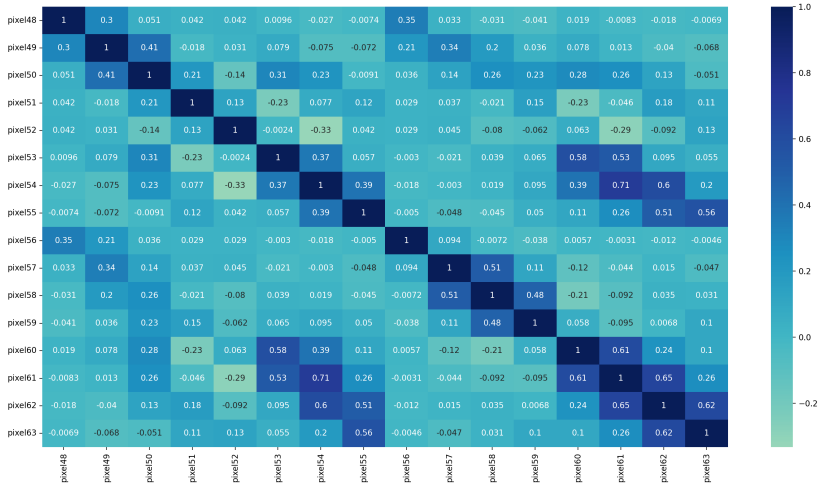

Figure 9: 49-64 列数据之间的相关系数矩阵

3 Random Forest

3.1 基础理论

随机森林的优点:

1. 精度高
2. 能够在大数据集上有效运行
3. 引入了不容易过度拟合的随机性
4. 随机森林具有良好的抗噪声能力，但当数据噪声较大时，会出现过拟合。
5. 能够处理高维数据而不进行降维
6. 它不仅可以处理离散数据，也可以处理连续数据，不需要对数据集进行标准化
7. 该算法训练速度快，可以得到各变量的重要性排序
8. 简单的并行化
9. 即使对于默认值问题，也能得到良好的结果
10. 超参数的数量不是很大，我们可以直观地理解它们所代表的意义

随机森林的缺点:

尽管随机森林算法速度足够快，但当随机森林中有很多决策树时，训练所需的空间和时间会很大，导致模型速度变慢。

随机森林算法可供选择的超参数和选项:

- n_estimators:

即最大的弱学习器的个数。一般来说 n_estimators 太小，容易欠拟合，n_estimators 太大，计算量会太大，并且 n_estimators 到一定的数量后，再增大 n_estimators 获得的模型提升会很小，所以一般选择一个适中的数值。默认是 100。

- oob_score :

即是否采用袋外样本来评估模型的好坏。默认识 False。个人推荐设置为 True，因为袋外分数反应了一个模型拟合后的泛化能力。

- criterion:

即 CART 树做划分时对特征的评价标准。分类模型和回归模型的损失函数是不一样的。分类 RF 对应的 CART 分类树默认是基尼系数 gini, 另一个可选择的标准是信息增益。回归 RF 对应的 CART 回归树默认是均方差 mse，另一个可以选择的标准是绝对值差 mae。一般来说选择默认的标准就已经很好的。

- max_depth: 每棵树最大深度
- min_samples_split: 划分节点时最小样本数
- min_samples_leaf: 每个叶子节点最小样本数
- min_weight_fraction_leaf: 划分节点时需要考虑的最多特征数
- max_feature: 划分节点时需要考虑的最多特征数
- max_leaf_nodes: 最大叶节点数
- bootstrap: 是否使用 bootstrap 对样本进行抽样
- oob_score: 是否记录 oob score
- random_state: 对样本和特征进行随机采样的随机器对象
- class_weight: 类别的权值
- max_samples:bootstrap 为 True 时每次抽样的最大样本数

3.2 实验设计及结果分析

本实验选择 max_features、n_estimators、max_leaf_nodes 三个参数进行实验：
不同参数选择在测试集上得到的结果如下：

Table 1: 不同参数选择在 TestSet-1 上得到的结果表

max_features	**n_estimators**	**max_leaf_nodes**	**test score**
auto	100	10	0.908333
		50	0.969444
		100	0.975000
	200	10	0.919444
		50	0.966667
		100	0.977778
	400	10	0.905556
		50	0.972222
		100	0.975000
sqrt	100	10	0.880556
		50	0.961111
		100	0.977778
	200	10	0.897222
		50	0.969444
		100	0.975000
	400	10	0.902778
		50	0.969444
		100	0.977778
log2	100	10	0.891667
		50	0.969444
		100	0.977778
	200	10	0.916667
		50	0.975000
		100	0.975000
	400	10	0.908333
		50	0.969444
		100	0.980556

分析解释： max_features 是选择的特征个数，有平方根、对数和 None（选取全部特征）等选项，该参数是随机森林模型最重要的参数，减小特征数会减小决策树之间的相关性但同时降低每棵树的分类能力，反之同理，因此需要选择最合适的特征数。

max_leaf_nodes 是最大叶子节点数。通过限制最大叶子节点数，可以防止过拟合，默认是"None"，即不限制最大的叶子节点数。如果加了限制，算法会建立在最大叶子节点数内最优的决策树。如果特征不多，可以不考虑这个值，但是如果特征分成多的话，可以加以限制，具体的值可以通过交叉验证得到。

n_estimators 是生成决策树的数量，理论上来说决策树数量越多，模型对样本估计的偏差越小，观察决策树数量对随机森林算法的影响，发现当决策树数量为 200 时，在测试集上的准确率整体上大于决策树数量为 100 的情况，具体最合适的决策树数量可以通过网格搜索寻找。

4 EX3 与 EX1 和 EX2 的对比

EX1 中，利用 lasso 回归对数据进行了处理：

Table 2: Lasso 指标结果图

Lasso_alpha	**train**	**k-folds**					**test**
0.0063	0.217	0.219	0.240	0.193	0.229	0.247	0.211

如上表所示，Lasso 变换在 alpha 取最佳值时，在训练集、交叉验证和测试集上的指标均达到了一个较理想的值。可见 Lasso 是一种适合多维度特征回归和分类任务的模型。但是缺点是 alpha 等参数每一次都需要做适当的调整。

EX2 中，利用 SVM 对数据进行了处理：

Table 3: SVM 测试集上的得分和交叉验证得分表

kernel type	**num of C**	**test score**	**cross-validation score**				
linear			1st	2nd	3rd	4th	5th
	0.01	0.901	0.877	0.896	0.846	0.874	0.898
	1	0.915	0.896	0.888	0.872	0.901	0.921
	5	0.920	0.901	0.877	0.872	0.903	0.911
polynomial	0.01	0.850	0.781	0.836	0.809	0.801	0.825
	1	0.939	0.906	0.901	0.867	0.893	0.906
	5	0.944	0.924	0.906	0.880	0.901	0.927
Gaussian	0.01	0.826	0.742	0.804	0.781	0.775	0.764
	1	0.925	0.903	0.901	0.885	0.898	0.919
	5	0.948	0.916	0.911	0.901	0.919	0.935
sigmoid	0.01	0.826	0.742	0.804	0.781	0.775	0.764
	1	0.817	0.789	0.794	0.7395	0.764	0.764
	5	0.803	0.765	0.778	0.744	0.770	0.772

如上表所示：

惩罚系数 C 是其中最重要的参数：C 越大则对错误分类样本的惩罚力度越大，因此训练集准确率更高，但是泛化能力会有所降低即测试集分类准确率降低；C 越小则对错误分类样本的惩罚力度越小，泛化能力有所增强。因此若出现过拟合现象，则应该适当减小 C，反之出现欠拟合现象则应该适当增大 C。

从表中可以看出，C 太小的话会导致训练出现欠拟合现象，导致精度下降，从而使得训练集与测试集错误率都较高；而 C 偏大的话可能会出现过拟合的情况，从 Gaussian 和 polynomial 中的结果就可以看出。从使用的 kernel 上可以看出，Gaussian 与 polynomial 核函数在训练集上错误率较低，而 Linear 在测试集上表现最好，其测试集错误率最低，具有较好的泛化能力。

总结：

对于模型选择和超参数调谐，可以针对个体估算器（如 Logistic 回归）或包括多个算法，特征化和其他步骤的整个管道完成调整。我们可以一次调整整个流水线，而不是单独调整管道中的每个元素，这样会大大提高我们的效率。

我们可以使用 CrossValidator 和 TrainValidationSplit 等工具进行模型选择。其中包含以下常用方法和手段：

- Estimator：算法或管道调整
- Set of ParamMaps：可供选择的参数，有时称为“参数网格”进行搜索
- Evaluator：衡量拟合模型对延伸测试数据有多好的度量

另一方面，这些模型选择工具将输入数据分成单独的训练和测试数据集。对于每个（训练，测试）对，遍历一组 ParamMaps：对于每个 ParamMap，它们使用这些参数适合 Estimator，获得拟合的 Model，并使用 Evaluator 评估 Model 的性能。最终选择由最佳性能参数组合生成的模型。使用者可以是回归问题的回归估值器，二进制数据的 BinaryClassificationEvaluator 或多类问题的 MulticlassClassificationEvaluator。用于选择最佳 ParamMap 的默认度量可以被这些评估器中的每一个的 setMetricName 方法覆盖。为了帮助构建参数网格，我们可以使用 ParamGridBuilder 实用程序。

还有个十分重要的技巧——交叉验证，CrossValidator 首先将数据集分成一组折叠，这些折叠用作单独的训练和测试数据集。例如，k = 5 倍，CrossValidator 将生成 5 个（训练，测试）数据集对，每个数据集使用 4/5 的数据进行训练，1/5 进行测试。为了评估一个特定的 ParamMap，CrossValidator 通过在 5 个不同的（训练，测试）数据集对上拟合 Estimator 来计算 5 个模型的平均评估度量。在确定最佳 ParamMap 之后，CrossValidator 最终使用最好的 ParamMap 和整个数据集重新拟合 Estimator。

5 类比不平衡的解决

5.1 理论基础

考虑使用 Focal Loss 解决:

Focal Loss 是 RetinaNet[1] 的核心, 也是其中最具创新性的一部分。它起源于二分类交叉熵 CE 损失函数, 即:

$$\mathrm{CE}\left(p_t\right) = -\log\left(p_t\right)$$

其中 p_t 根据目标类别 y 来确定具体值: 当目标类别为前景, 即 $y = 1$ 时, $p_t = p, p \in [0, 1]$; 当目标类别为背景, 即 $y = -1$ 时, $p_t = 1 - p, p \in [0, 1]$ 。如图10所示, CE 曲线即是图中蓝色的曲线（即最上面的曲线）。

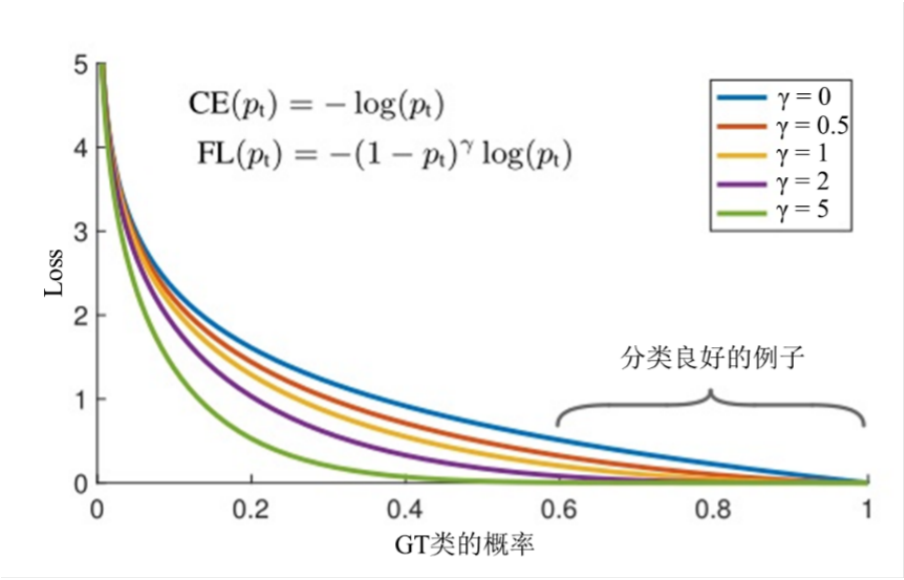

Figure 10: CE 和 FL 损失函数曲线 [1]

相比于其他曲线, 蓝色曲线是变化最为平缓的, 即使在 $p > 0.5$ (即较易分类样本) 的时候, 它的 loss 值仍是处于一个较高的水平, 由于训练过程中, 易分类样本的数量极为庞大, 即使单个易分类样本的 loss 极低, 但是乘以数量以后总 loss 将是一个巨大的数值, 这便会导致易分类样本主导训练的过程。

由于前景和背景数量极为不平衡（背景数量一般相比前景过于庞大）, 往往会导致训练时过多将注意力集中在背景上。为了解决这一问题, 常用的思想是引入权重 α, $\alpha \in [0,1] \circ \alpha$ 的定义可类比于 α, 同样为了表示方便, 便用 α_t 代表引入的权重。此时 CE 损失函数被改写为

$$\mathrm{CE}\left(p_t\right) = -\alpha_t \log\left(p_t\right)$$

可以看出若设置 α 值较大, 则前景对应损失值较大, 便能中和其数量较少的劣势; 而与此同时 $1-\alpha$ 的值较小, 也能限制住背景数量庞大的情况。虽然解决了正负样本比例不协调的问题, 但是之前提到的易分类样本和难分类样本的数量不协调问题仍然未解决, 这时何凯明在《Focal Loss for Dense Object Detection》一文中提到引入调节因子 $(1-p_t)^{\gamma}$, 新的损失函数 Focal Loss 便诞生了:

$$\mathrm{FL}\left(p_t\right) = -\alpha_t \left(1-p_t\right)^{\gamma} \log\left(p_t\right)$$

其中 $\gamma \in [0,5]$ 。

可以看出, 当 $p_t \to 1$ 的时候, 样本趋于极易分类, 此时整个损失函数的值趋于 0 ; 而当 $p_t \to 0$, 即某样本被误分类, 导致 p_t 很小, 那么此时的 FL 中调节因子 $(1-p_t)^{\gamma}$ 是趋于 1 的, 即相对原来的 CE 损失函数没有大改变。$(1-p_t)^{\gamma}$ 的加入, 解决了由于易分类样本数量比例较大而造成的易分类样本主导训练的情况。而如图11 所示, 当 γ 增大时, 损失函数曲线也变得陡峭, γ 越大, 易分类样本的 loss 越小, 但是 γ 过大会导致难分类样本的 loss 值也大幅下降。根据实验表明, 一般情况下, 取 $\gamma = 2$ 为最佳。γ 和 α_t 的最佳值是相互影响的, 且对于不同场景有不同的取值, 所以在评估准确度时, 要将两者结合起来综合调节。

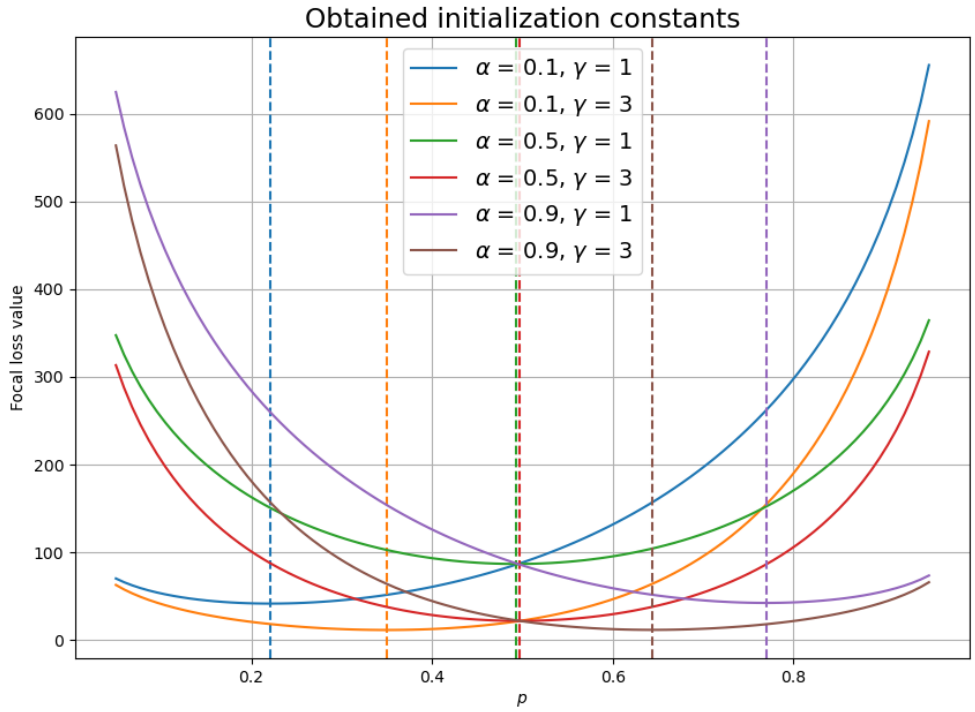

Figure 11: focal_loss 曲线

5.2 训练及结果可视化

本人基于 LGBM 算法，在其基础上实现 focal_loss，之后训练 1000 个 epochs 后，得到最终结果，如图12所示。

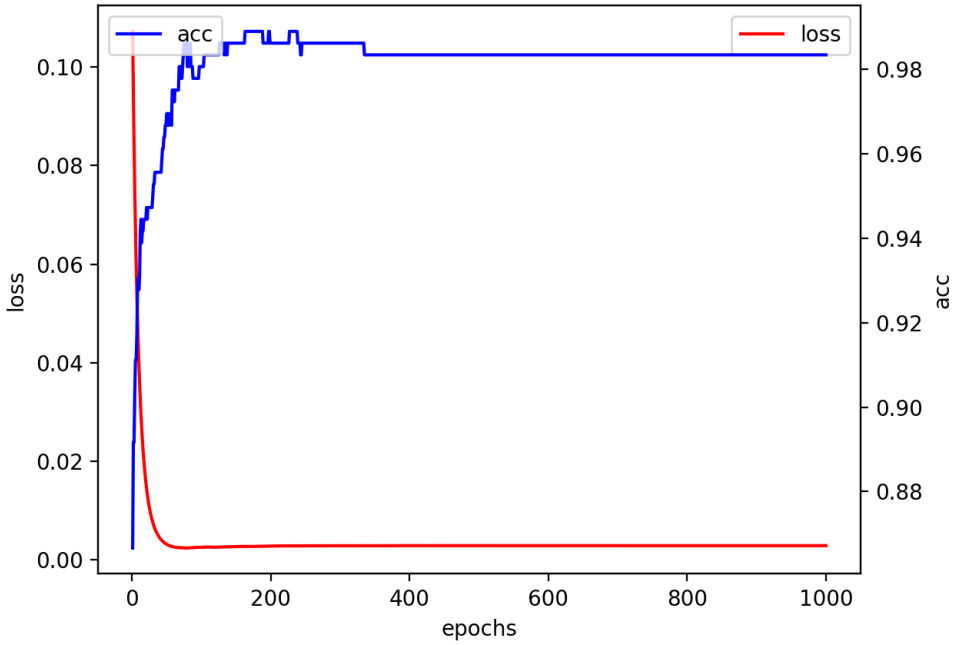

Figure 12: 基于 LGBM 的 focal_loss 训练过程曲线

最终得到测试集上的精度为 0.983

5.3 代码

```
from operator import index
import numpy as np
import lightgbm as lgb
from numpy.lib.function_base import append

from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from scipy.misc import derivative
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings("ignore")
losses = []
acces = []
def focal_loss_lgb(y_pred, dtrain, alpha, gamma, num_class):
    a, g = alpha, gamma
    y_true = dtrain.label
    # N observations x num_class arrays
    y_true = np.eye(num_class)[y_true.astype("int")]
    y_pred = y_pred.reshape(-1, num_class, order="F")
    # alpha and gamma multiplicative factors with BCEWithLogitsLoss
    def fl(x, t):
        p = 1 / (1 + np.exp(-x))
        return (
            -(a * t + (1 - a) * (1 - t))
```

```
            * ((1 - (t * p + (1 - t) * (1 - p))) ** g)
            * (t * np.log(p) + (1 - t) * np.log(1 - p))
        )

    partial_fl = lambda x: fl(x, y_true)
    grad = derivative(partial_fl, y_pred, n=1, dx=1e-6)
    hess = derivative(partial_fl, y_pred, n=2, dx=1e-6)
    # flatten in column-major (Fortran-style) order

    return grad.flatten("F"), hess.flatten("F")


def focal_loss_lgb_eval_error(y_pred, dtrain, alpha, gamma, num_class):
    a, g = alpha, gamma
    y_true = dtrain.label
    y_true = np.eye(num_class)[y_true.astype("int")]
    y_pred = y_pred.reshape(-1, num_class, order="F")
    p = 1 / (1 + np.exp(-y_pred))
    loss = (
        -(a * y_true + (1 - a) * (1 - y_true))
        * ((1 - (y_true * p + (1 - y_true) * (1 - p))) ** g)
        * (y_true * np.log(p) + (1 - y_true) * np.log(1 - p))
    )
    # a variant can be np.sum(loss)/num_class

    res = []
    for i in y_pred:
        i = list(i)
        res.append(i.index(max(i)))

    tr = []
    for i in y_true:
        i = list(i)
        tr.append(i.index(max(i)))
    acces.append(accuracy_score(tr, res))
    # global losses

    losses.append(np.mean(loss))

    return "focal_loss", np.mean(loss), False

if __name__ == "__main__":
    # very inadequate dataset as is perfectly balanced, but just to illustrate
    digits = datasets.load_digits()
    X = digits.data
    y = digits.target

    X_tr, X_val, y_tr, y_val = train_test_split(X, y, test_size=0.2, random_state=1)
    lgbtrain = lgb.Dataset(X_tr, y_tr, free_raw_data=True)
    lgbeval = lgb.Dataset(X_val, y_val)
    focal_loss = lambda x, y: focal_loss_lgb(x, y, 0.25, 2.0, 10)
    eval_error = lambda x, y: focal_loss_lgb_eval_error(x, y, 0.25, 2.0, 10)
    params = {
        "learning_rate": 0.1,
        "num_boost_round": 100,
        "num_class": 10,
        "verbose": -1,
    }
```

```
    model = lgb.train(
        params, lgbtrain, valid_sets=[lgbeval], fobj=focal_loss, feval=eval_error
    )
    y_pred = list(model.predict(X_val))
    res = []
    for i in y_pred:
        i = list(i)
        res.append(i.index(max(i)))
    y_pred = np.array(res)
    print(accuracy_score(y_val, y_pred))
    epochs = [i for i in range(1, len(losses) + 1)]
    fig = plt.figure()
    ax = fig.add_subplot(111)
    ax2 = ax.twinx()
    ax.plot(epochs, losses, label="loss", color="red")
    ax2.plot(epochs, acces, label="acc", color="blue")
    ax.set_xlabel("epochs")
    ax.set_ylabel("loss")
    ax.legend(loc=0)
    ax2.set_ylabel("acc")
    ax2.legend(loc=2)
    plt.show()
```

6 图片像素分类

如图13为整体结果图，

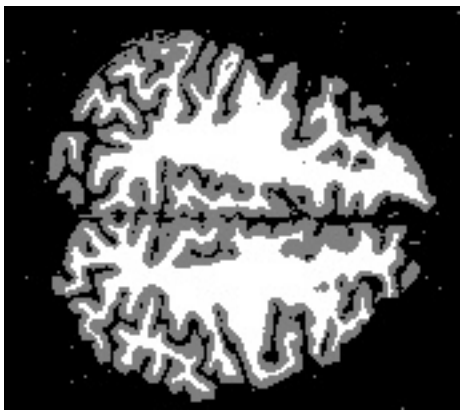

Figure 13: 分类后的效果展示图

如图14、15、16为三类分别生产的图，其中白色是前景，黑色是背景：

Figure 14: outside brain

Figure 15: gray matter

Figure 16: white matter

如图17所示，为 log_likelihood 随迭代次数的变化曲线。

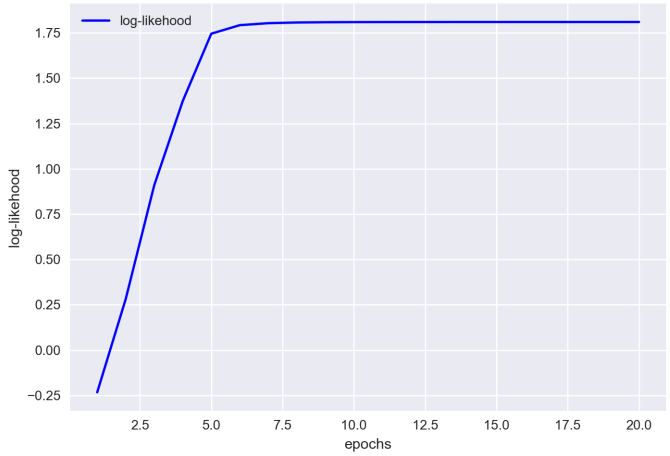

Figure 17: log_likelihood 曲线

具体步骤详见代码注释:

代码:

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.patches import Ellipse
from scipy.stats import multivariate_normal
from sklearn.preprocessing import MinMaxScaler
import cv2

plt.style.use("seaborn")

# 更新W
def update_W(X, Mu, Var, Pi):
    n_points, n_clusters = len(X), len(Pi)
    pdfs = np.zeros(((n_points, n_clusters)))
    for i in range(n_clusters):
        pdfs[:, i] = Pi[i] * multivariate_normal.pdf(X, Mu[i], np.diag(Var[i]))
    W = pdfs / pdfs.sum(axis=1).reshape(-1, 1)
    return W


# 更新pi
def update_Pi(W):
    Pi = W.sum(axis=0) / W.sum()
    return Pi


# 计算log似然函数
def logLH(X, Pi, Mu, Var):
    n_points, n_clusters = len(X), len(Pi)
    pdfs = np.zeros(((n_points, n_clusters)))
    for i in range(n_clusters):
        pdfs[:, i] = Pi[i] * multivariate_normal.pdf(X, Mu[i], np.diag(Var[i]))
    return np.mean(np.log(pdfs.sum(axis=1)))


```

```
# 更新Mu
def update_Mu(X, W):
    n_clusters = W.shape[1]
    Mu = np.zeros((n_clusters, 2))
    for i in range(n_clusters):
        Mu[i] = np.average(X, axis=0, weights=W[:, i])
    return Mu


# 更新Var
def update_Var(X, Mu, W):
    n_clusters = W.shape[1]
    Var = np.zeros((n_clusters, 2))
    for i in range(n_clusters):
        Var[i] = np.average((X - Mu[i]) ** 2, axis=0, weights=W[:, i])
    return Var


if __name__ == "__main__":
    # load the data
    data = []
    with open("brainimage.txt") as f:
        for line in f.readlines():
            temp = []
            s = line.strip().split("    ")
            for item in s:
                data.append(int(item))
    data = np.array(data)

    # normalizing the data
    ss = MinMaxScaler()
    data = data.reshape(-1, 1)
    data = ss.fit_transform(data)

    var = data.var()
    # 初始化
    n_clusters = 3
    n_points = len(data)
    Mu = [[0.3], [0.5], [0.7]]
    Var = [[var], [var], [var]]
    Pi = [1 / n_clusters] * 3
    W = np.ones((n_points, n_clusters)) / n_clusters
    Pi = W.sum(axis=0) / W.sum()
    # 迭代
    loglh = []
    for i in range(20):
        loglh.append(logLH(data, Pi, Mu, Var))
        W = update_W(data, Mu, Var, Pi)
        Pi = update_Pi(W)
        Mu = update_Mu(data, W)
        print("log-likehood:%.3f" % loglh[-1])
        Var = update_Var(data, Mu, W)

    pred = W
    predicted_cls = []
    for i in pred:
        i = list(i)
        predicted_cls.append(i.index(max(i)))
```

```
93
94      # 画图
95      idx = 0
96      res = []
97      for i in range(151):
98          temp = []
99          for j in range(171):
100             if predicted_cls[idx] == 0:
101                 temp.append(255)
102             else:
103                 temp.append(0)
104             idx += 1
105         res.append(temp)
106     res = np.array(res)
107
108     cv2.imwrite("outside brain.jpg", res)
109
110     idx = 0
111     res = []
112     for i in range(151):
113         temp = []
114         for j in range(171):
115             if predicted_cls[idx] == 1:
116                 temp.append(255)
117             else:
118                 temp.append(0)
119             idx += 1
120         res.append(temp)
121     res = np.array(res)
122
123     cv2.imwrite("gray matter.jpg", res)
124
125     idx = 0
126     res = []
127     for i in range(151):
128         temp = []
129         for j in range(171):
130             if predicted_cls[idx] == 2:
131                 temp.append(255)
132             else:
133                 temp.append(0)
134             idx += 1
135         res.append(temp)
136     res = np.array(res)
137
138     cv2.imwrite("white matter.jpg", res)
139
140     epochs = [i for i in range(1, 21)]
141     fig = plt.figure()
142     ax = fig.add_subplot(111)
143     ax.plot(epochs, loglh, label="log-likehood", color="blue")
144     ax.set_xlabel("epochs")
145     ax.set_ylabel("log-likehood")
146     ax.legend(loc=0)
147     plt.show()
```

References

[1] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár, "Focal loss for dense object detection," in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 2980–2988.