计算机与信息技术学院

本科生《机器学习》课程作业

第一次大作业报告

学　　号 **20241068**

姓　　名 魏振杰

班　　级 天佑 **2004** 班

日　　期 **12** 月 **18** 日

机器学习第一次大作业

一、基本任务的完成

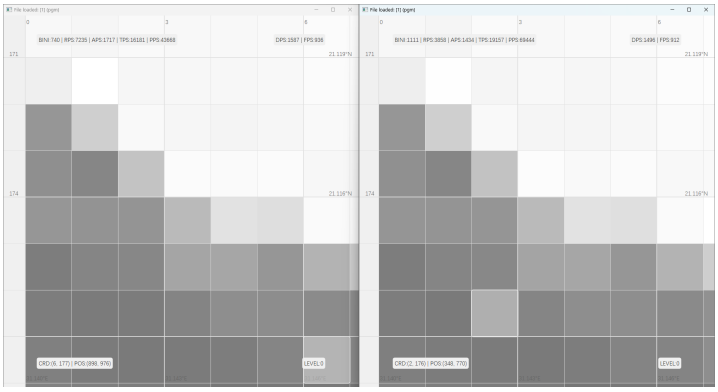
0 对实验要求中几个基本问题的回答

- 如何划分训练集/测试集?
  - 统一采用**十折交叉验证**，该方法下每组数据都有机会成为训练集/数据集。
- 选取何种分类特征?
  - 利用**卷积核**对图像进行处理，得到残差图像，而后对像素进行**直方图统计**，在统计结果中**选取较有代表性的数据**作为特征。
- 提取特征的基本流程?
  - **读取图片->卷积处理->统计直方图->选取合适特征。**
- 选取何种分类模型?
  - 基本实验采用**朴素贝叶斯**完成，辅助使用LR、LDA、k近邻、SVM等模型完成。
- 实验结果的展示对比?
  - 本实验采用**精度、查准率、查全率、f1值**四个维度的评价指标，详情请查阅报告。
- 实验心得体会?
  - 详情请查阅报告最后一章。

1 实验数据的分析

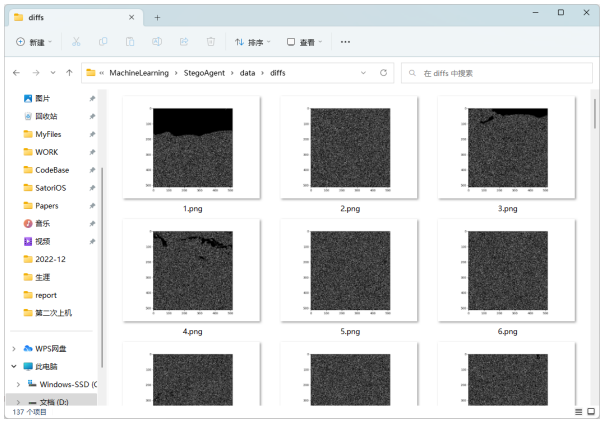
在实验开始之前，我首先尝试分析数据的特征。经查阅相关资料发现，本次数据集涉及到一种**隐写技术**，即向普通的图片中写入经过加密后的数据，并尽可能使得图片的变化**肉眼不可辨别**。

事实上，我尝试过使用我自己开发的**图片像素查看工具**，发现确实很难区分二者的区别。下图左`Cover`，右`Stego`。



那么隐写技术到底是怎样完成的呢？我首先对Cover/Stego两两做了相减，得到了如下数据。经过分析不难发现，Stego中的每一个像素，都相对于Cover有`0，-1，1`的变化（**0和1在下图中显示为黑色的像素点，而-1由于数据溢出变为255则显示白色的像素点**），以此便可在图像中嵌入早已准备好的二进制信息，从而实现了隐形的信息传递。该隐写技术对像素的处理过程可以简单总结如下。（原图像每个像素点的三个通道值均相同，为方便描述，此处用灰度值代替）

- 对于原像素灰度值为0或255的情况，跳过不做处理。（因为对此像素点进行的增减值处理可能会引起像素由黑变白或者由白变黑，无法达到隐写的目的）
- 除上述情况外，对原像素值进行+1/-1处理，可以约定+1代表二进制的1，-1代表二进制的0。
- 在读取信息时，首先找到Cover和对应的Stego做差，而后遍历残差图的每个像素，若该像素值为0则跳过，若为1则记为二进制的1，若为255则记为二进制的0，从而获取到隐写的信息。



2 实验数据的预处理

经过上述分析可以发现，实验数据必须经过进一步的预处理才能输入模型进行训练，否则将会遇到以下**两个致命问题**：

- **数据维度过多。**数据中每张图像的总像素数为$512 \times 512 = 262144$，也就是说，即使只考虑图像的一个通道，每个数据的**特征维度也达到了数十万的级别**。且不论对如此大的数据进行训练所需的**时间、空间复杂度个人PC难以承受**，仅仅考虑到**数据的个数远小于数据维度的个数**这一个方面就可以预见直接将数据不经处理放入模型的做法是很难有好的结果的。因此，必须采用合适的方法对数据进行降维。
- **图像具有对偶性。**可以进行这样一个思考实验：给我任意一张图片（可能是Cover，也可能是Stego）以及需要隐写的数据，那么我可以根据这些**构造任意一种图像类型**。也就是说，如果不考

虑图像的自然特征（比如**自然拍出的图片相对隐写后的图片可能会平滑一些**）的话，理论上是无法仅凭图片自身完成二分类任务的。因此，我们需要对图像进行处理，以最大限度地**保留其自然特征**，并将该特征作为数据特征进行训练。

下面介绍我的图像预处理过程：

2.1 图像的读入

经过实验发现，采用`opencv-python`的`imread`函数读取图像是一个比较方便的选择。我将数据集放到了实验目录下并将文件夹改名为`data`，因此，可用以下代码读取图像：

```
path = 'data'
src = 'Stego' if typ == 1 else 'Cover'
img = cv2.imread(path + f'/{src}/{idx+1}.pgm')
```

2.2 图像的卷积

卷积核的选取遵循如下原则：

> ①卷积核往往是**行数和列数均为奇数的矩阵**，以便定位其中心。
>
> ②卷积核元素的总和体现输出的亮度，若元素总和为1，卷积后的图像与原图像亮度基本一致；若元素总和为0，则卷积后的图像基本上是黑色，其中较亮的部分往往就是提取出图像的某种特征。这里为了减少无关特征的影响，拟**采取总元素和为0的卷积核。**
>
> ③一般卷积核可以根据作用分为**高通滤波器、低通滤波器和均值滤波器**等，其中高通滤波器仅允许图像中的高频部分（即图片中变化较剧烈的部分）通过，往往用于对图像进行锐化处理、增强图像中物体边缘等。如Sobel算子、Prewitt算子、锐化滤波器等；而低通滤波器仅允许图像中低频部分（即图片中变化较平缓的部分）通过，往往用于对图像进行模糊/平滑处理、消除噪点等。如高斯滤波器、均值滤波器等。这里为了发现图象被修改的痕迹，倾向于**采用高通滤波器（即锐化卷积核）**。

本次实验采取的卷积核为：

```
kernels = [
    [[0, -1, 0],[-1, 4, -1], [0, -1, 0]],
    [-1, 1],
    [[1, 0, -1], [1, 0, -1], [1, 0, -1]],
    [[-1, 2, -2, 2, -1],[2, -6, 8, -6, 2],[-2, 8, -12, 8, -2],[2, -6, 8, -6, 2],[-1, 2, -2, 2, -1]]
]
```

实验将会对同一个图像依次应用每一个卷积核，并将得到的结果进行拼接。实验采用`cv2`的`filter2D`函数完成卷积操作。

```
cov = cv2.filter2D(img, cv2.CV_8U, kernel=np.array(ker))
```

2.3 特征的提取

上述操作后将会生成大量的特征，因此本节将按如下两个步骤进行特征提取和降维：

- 统计**灰度直方图**，略去像素位置信息

```
hist = cv2.calcHist([cov], [0], None, [256], [0, 256]).flatten()
```

- 对直方图进行**一维池化操作**，进一步降低特征维度

```
pool = nn.MaxPool1d(5)
hist = pool(torch.tensor(hist.reshape(1,-1))).numpy().squeeze()
```

2.4 编写预处理函数

最后，可以将上述步骤整合为图像预处理函数如下：

```
def pre_process(idx: int, typ: int = 1):
    res = []
    src = 'Stego' if typ == 1 else 'Cover'
    img = cv2.imread(path + f'/{src}/{idx+1}.pgm')
    for ker in kernels:
        cov = cv2.filter2D(img, cv2.CV_8U, kernel=np.array(ker))
        hist = cv2.calcHist([cov], [0], None, [256], [0, 256]).flatten()
        hist = pool(torch.tensor(hist.reshape(1,-1))).numpy().squeeze()
        res = np.concatenate((res, hist), axis=0)
    res = np.append(res, float(typ))
    return res
```

该函数接受两个参数：

> idx: int。图像的编号；typ: int 图像的类型（Stego为1，Cover为0）

函数返回特征提取的结果，即一个$n \times m$的矩阵，其中n是数据集的长度，m包含$m-1$维的特征和最后一维的数据标记。

3 使用至少一种分类模型

使用多项式朴素贝叶斯模型对提取的特征进行分类，相关代码如下。

```
model=MultinomialNB()
prediction=cross_val_predict(model,data_x,data_y,cv=10)
acc = accuracy_score(data_y,prediction)
precision = precision_score(data_y,prediction)
recall = recall_score(data_y,prediction)
f = f1_score(data_y,prediction)
```

实验结果为如下。可见，目前为止，已经实现了对数据集的初步的二分类，上述特征提取流程取得了一定的效果。

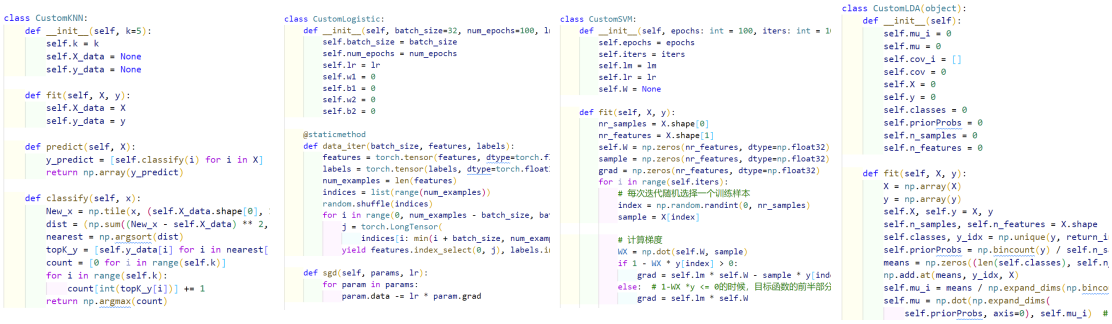
```
MultinomialNB的四项指标
精度: 0.72
查准率: 0.6964285714285714
查全率: 0.78
f1值: 0.7358490566037736
```

使用其他模型的结果请参见**第二章2.1小节**的讨论。

4 尝试手动实现分类模型

本人尝试参照`sklearn`的格式（即实现`fit`/`predict`函数）手动实现了KNN、SVM、LDA和逻辑回归，其中**KNN的完成度最高**，模型训练结果与调包相同，但**SVM的实现存在问题**，分类结果的精度较低。其中，**逻辑回归使用前向传播和梯度下降的方式实现。**此外，为了使我的模型和标准模型的使用方式尽可能类似，我还额外实现了自己的**十折交叉验证**函数。由于篇幅原因，这里不再贴出代码细节，详

情可查看源码。



二、进阶探索与尝试

1 优化特征提取流程

1.1 优化数据降维过程

首先对特征提取函数（也就是上述的预处理函数）进行优化。

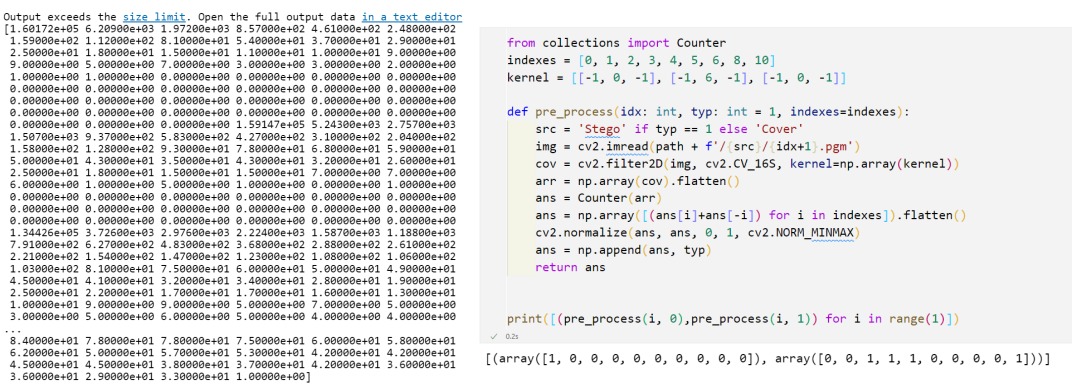
核心思路如下：

- **不再**同时使用多个卷积核（下面会针对不同卷积核的效果进行**实验讨论**）
- 卷积核的卷积结果**数据类型**改为 `cv2.CV_16S`，即允许**出现负数**
- 将 `calcHist` 替换为 `Counter` 函数，因为前者**无法统计负值**
- 不用池化，而对灰度统计结果**手动取值**，方法是**将正负同值的数量合并**，且只取 `indexes` 中定义的灰度值
- 添加**归一化**处理，将统计得到的灰度数量处理为0-1之间的值

```python
indexes = [0, 1, 2, 3, 4, 5, 6, 8, 10]

def pre_process(idx: int, typ: int = 1, indexes=indexes):
    src = 'Stego' if typ == 1 else 'Cover'
    img = cv2.imread(path + f'/{src}/{idx+1}.pgm')
    cov = cv2.filter2D(img, cv2.CV_16S, kernel=np.array(kernel))
    arr = np.array(cov).flatten()
    ans = Counter(arr)
    ans = np.array([(ans[i]+ans[-i]) for i in indexes]).flatten()
    cv2.normalize(ans, ans, 0, 1, cv2.NORM_MINMAX)
    ans = np.append(ans, typ)
    return ans
```

进行**优化前后**的数据预处理的结果分别如左右两图所示，可见优化后的特征更为**清晰、简洁**。



使用新的数据预处理函数后，再使用朴素贝叶斯模型进行训练，发现**效果明显变好。**（左一是优化前结果，**右一至三是优化后结果**）

```
MultinomialNB的四项指标
精度：0.72
查准率：0.6964285714285714
查全率：0.78
f1值：0.7358490566037736
```

```
GaussianNB的四项指标
精度：0.77
查准率：0.684931506849315
查全率：1.0
f1值：0.8130081300813007
```

```
BernoulliNB的四项指标
精度：0.865
查准率：0.8411214953271028
查全率：0.9
f1值：0.8695652173913043
```

```
MultinomialNB的四项指标
精度：0.87
查准率：0.8490566037735849
查全率：0.9
f1值：0.8737864077669903
```

1.2 特征提取的进一步实验

经过上述分析可以发现，`indexes`决定了经过预处理后的特征中**保留哪些特征作为最终输入模型的特征**，即**决定了特征降维的过程**，新的特征提取函数的分类效果是和`indexes`的选取有**较强的相关性**的。那么什么样的`indexes`的效果最好呢？针对此，我做了如下实验：

针对卷积核（`kernel_0`）`[[0, -1, 0], [-1, 4, -1], [0, -1, 0]]`，分别采取**不同的indexes组合**，使用**朴素贝叶斯分类器**对数据进行训练和预测，记录**高斯方法（acc1-f1）**、**伯努利方法（acc2-f2）**和**多项式方法（acc3-f3）**结果至Excel中（`Result_ker_0.xlsx`）。排序取得**效果最好**的索引集如下。可见，此时整体的分类效果已经相较于最初有了**非常明显的进步**，证明了机器学习针对此类数据的**有效性。**

no	indexes	acc1	pc1	rc1	f1	acc2	pc2	rc2	f2	acc3	pc3	rc3	f3
41	[0, 3, 5]	0.94	0.89	1	0.94	0.94	0.89	1	0.94	0.94	0.89	1	0.94
43	[0, 1, 3, 5]	0.94	0.89	1	0.94	0.94	0.89	1	0.94	0.94	0.89	1	0.94
47	[0, 1, 2, 3, 5]	0.94	0.89	1	0.94	0.94	0.89	1	0.94	0.94	0.89	1	0.94

对于同一个卷积核，当索引集中混入较大的index**（如8、16）**时，模型会受到干扰（**较大的残差值出现的概率极低**，相比之下会使得其他有效值在**归一化的过程**中被粗鲁地划分为1，从而丢失了部分信息），导致精度下降：

no	indexes	acc1	pc1	rc1	f1	acc2	pc2	rc2	f2	acc3	pc3	rc3	f3
104	[3, 5, 8]	0.85	0.86	0.84	0.85	0.85	0.86	0.84	0.85	0.85	0.86	0.84	0.85
165	[0, 2, 5, 16]	0.85	0.83	0.89	0.86	0.87	0.87	0.86	0.87	0.87	0.87	0.86	0.87
229	[0, 2, 5, 8, 16]	0.85	0.83	0.89	0.86	0.87	0.87	0.86	0.87	0.87	0.87	0.86	0.87

在某些情况下，**查准率可以达到1**，但此时模型显得**非常谨慎**，**查全率极低**，因而精度较低。由此可见查全率与查准率对于机器学习问题来说是**鱼与熊掌**的关系，一般来说二者不可得兼。

no	indexes	acc1	pc1	rc1	f1	acc2	pc2	rc2	f2	acc3	pc3	rc3	f3
7	[0, 1, 2]	0.59	1	0.17	0.29	0.6	0.59	0.67	0.63	0.59	1	0.17	0.29
5	[0, 2]	0.58	1	0.16	0.27	0.58	1	0.16	0.27	0.58	1	0.16	0.27
11	[0, 1, 3]	0.56	1	0.12	0.22	0.52	0.55	0.25	0.34	0.56	1	0.12	0.22

索引集中包含**0-5（较小的残差值）**中至少三个时，查全率最高，一般接近于1，可以认为**能够查找出所有正例**，否则查全率会打折扣。当只包含少量的**小残差值特征**时，查全率急剧变小，精度在0.5附近徘徊，可以认为**与瞎猜几无区别。**

no	indexes	acc1	pc1	rc1	f1	acc2	pc2	rc2	f2	acc3	pc3	rc3	f3
88	[3, 4, 8]	0.52	0.8	0.06	0.12	0.52	0.8	0.06	0.12	0.47	0.46	0.38	0.41
194	[1, 8, 16]	0.52	0.67	0.06	0.11	0.5	0.5	0.14	0.22	0.5	0.5	0.14	0.22
10	[1, 3]	0.5	0	0	0	0.51	1	0.02	0.03	0.47	0.46	0.38	0.41

1.3 卷积核的选取实验

上述实验结果是**卷积核相关的。**事实上，上述讨论所采取的卷积核（`kernel_0`）（`[[0, -1, 0], [-1, 4, -1], [0, -1, 0]]`）是目前看来效果最好的卷积核，此外我还尝试了其他卷积核以作对比。下面均从所有的实验结果中**选取精度最高的索引集**（前三个）。

- 普通残差卷积核（`kernel_1`）：`[-1,1]`

no	indexes	acc1	pc1	rc1	f1	acc2	pc2	rc2	f2	acc3	pc3	rc3	f3
7	[0, 1, 2]	0.84	0.76	1	0.86	0.84	0.76	1	0.86	0.84	0.76	1	0.86
11	[0, 1, 3]	0.72	0.64	1	0.78	0.72	0.64	1	0.78	0.72	0.64	1	0.78
15	[0, 1, 2, 3]	0.72	0.64	1	0.78	0.72	0.64	1	0.78	0.73	0.65	1	0.79

- 边缘检测卷积核（`kernel_2`）：`[[1, 0, -1],[1, 0, -1],[1, 0, -1]]`

no	indexes	acc1	pc1	rc1	f1	acc2	pc2	rc2	f2	acc3	pc3	rc3	f3
21	[0, 2, 4]	0.73	0.87	0.53	0.66	0.73	0.87	0.53	0.66	0.73	0.87	0.53	0.66
23	[0, 1, 2, 4]	0.73	0.87	0.53	0.66	0.73	0.87	0.53	0.66	0.73	0.87	0.53	0.66
29	[0, 2, 3, 4]	0.73	0.87	0.53	0.66	0.73	0.87	0.53	0.66	0.73	0.87	0.53	0.66

- 高通滤波卷积核（`kernel_3`）：`([[-1, 2, -2, 2, -1],[2, -6, 8, -6, 2],[-2, 8, -12, 8, -2],[2, -6, 8, -6, 2],[-1, 2, -2, 2, -1]])/12.0`

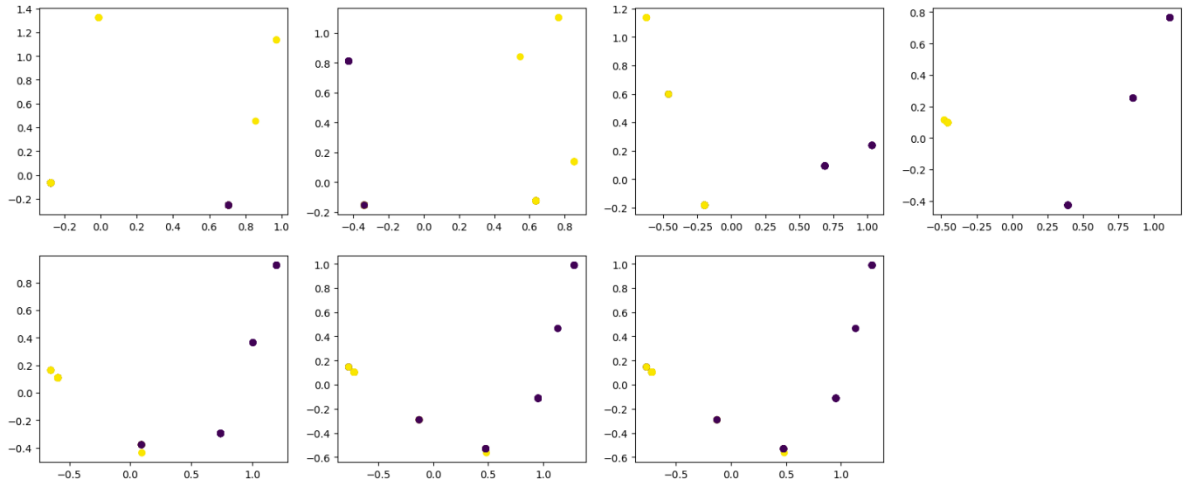
no	indexes	acc1	pc1	rc1	f1	acc2	pc2	rc2	f2	acc3	pc3	rc3	f3
37	[0, 2, 5]	0.93	0.94	0.92	0.93	0.93	0.94	0.92	0.93	0.93	0.94	0.92	0.93
53	[0, 2, 4, 5]	0.93	0.94	0.92	0.93	0.93	0.94	0.92	0.93	0.93	0.94	0.92	0.93
133	[0, 2, 16]	0.93	0.89	0.98	0.93	0.93	0.89	0.98	0.93	0.93	0.89	0.98	0.93

综上，可知**高通滤波卷积核**（`kernel_0(0.94)`/`kernel_3(0.93)`）整体的表现都**比较亮眼**，可知针对图像隐写问题，使用该类卷积核对图像的锐化操作是比较优质的选择，这也与我在**报告第一章2.2节**的推断相符合。

1.4 使用PCA进行降维

除了上述讨论之外，我还尝试直接使用PCA进行降维。下面是使用PCA分别将基准`indexes`中的**前3-9维降维至2维**时的结果，可见PCA将**6维降至2维的效果较好**，这也与上述讨论出来的最好的`indexes`相符合。

```
[GaussianNB()]: 精度:0.59, 查准率:1.0, 查全率:0.17, f1值:0.29
[GaussianNB()]: 精度:0.76, 查准率:0.88, 查全率:0.59, f1值:0.71
[GaussianNB()]: 精度:0.73, 查准率:0.65, 查全率:1.0, f1值:0.79
[GaussianNB()]: 精度:0.92, 查准率:0.89, 查全率:0.97, f1值:0.93
[GaussianNB()]: 精度:0.88, 查准率:0.88, 查全率:0.88, f1值:0.88
[GaussianNB()]: 精度:0.86, 查准率:0.85, 查全率:0.88, f1值:0.86
[GaussianNB()]: 精度:0.86, 查准率:0.85, 查全率:0.88, f1值:0.86
```



2 优化模型训练过程

2.1 其他分类模型

在完成上述对特征提取部分的讨论后，我又将目光放在了**不同的模型**之上。我将`sklearn`中比较常见的适合用于分类的模型全部拿来跑了一遍，结果发现如下**令人震惊**的现象。下面左图是针对我**任意选取**的一个`indexes`集的模型训练测试结果，右图是选取经过上述讨论后得到的**最优**的`indexes`训练结果。可以发现，当`indexes`的选取不是最优时，**模型之间的差异会影响分类的效果**，但当`indexes`的选取是**最优**的时，可以说，分类的结果基本已经**完全取决于特征所包含的信息**，而与选择的模型无关了。可见，此时所有模型的查全率均为**1**，也许我们可以推断，隐写图像中所包含的特征信息**足以使我们将其分辨出来**，但我们并不总是有足够的把握**认为一张图片是没有经过隐写处理的**，这也与我在**报告第一章第2节**的讨论相呼应。

model	acc	pc	rc	f1
SVC	0.87	0.87	0.86	0.87
ExtraTreesClassifier	0.85	0.87	0.83	0.85
BaggingClassifier	0.85	0.86	0.84	0.85
DecisionTreeClassifier	0.85	0.87	0.83	0.85
KNeighborsClassifier	0.85	0.84	0.88	0.85
RandomForestClassifier	0.84	0.83	0.84	0.84
GradientBoostingClassifier	0.84	0.83	0.86	0.85
LogisticRegression	0.84	0.81	0.88	0.84
AdaBoostClassifier	0.82	0.81	0.84	0.82
MLPClassifier	0.82	0.81	0.84	0.82
SGDClassifier	0.79	0.78	0.8	0.79
GaussianNB	0.77	0.69	1	0.82

```
[RandomForestClassifier()]:     精度:0.94, 查准率:0.89, 查全率:1.0, f1值:0.94
[AdaBoostClassifier()]:         精度:0.94, 查准率:0.89, 查全率:1.0, f1值:0.94
[GradientBoostingClassifier()]: 精度:0.94, 查准率:0.89, 查全率:1.0, f1值:0.94
[ExtraTreesClassifier()]:       精度:0.94, 查准率:0.89, 查全率:1.0, f1值:0.94
[BaggingClassifier()]:          精度:0.94, 查准率:0.89, 查全率:1.0, f1值:0.94
[DecisionTreeClassifier()]:     精度:0.94, 查准率:0.89, 查全率:1.0, f1值:0.94
[KNeighborsClassifier()]:       精度:0.94, 查准率:0.89, 查全率:1.0, f1值:0.94
[SVC()]:                        精度:0.94, 查准率:0.89, 查全率:1.0, f1值:0.94
[GaussianNB()]:                 精度:0.94, 查准率:0.89, 查全率:1.0, f1值:0.94
[LogisticRegression()]:         精度:0.94, 查准率:0.89, 查全率:1.0, f1值:0.94
[SGDClassifier()]:              精度:0.94, 查准率:0.89, 查全率:1.0, f1值:0.94
[MLPClassifier()]:              精度:0.94, 查准率:0.89, 查全率:1.0, f1值:0.94
```

2.2 使用深度学习

为了测试在不进行较激进的特征提取策略下，使用本身适用于多维特征数据的深度学习类模型是否有更好的结果，我使用MLP模型跑17维的indexes的结果如下，可见随着维数的增多，精度并没有相应的提升。

```
indexes = [0,1,2,3,5,8,9,10,11,12,13,14,15,16]
```

```
train(data_x, data_y, models=[mlp])
```

3.2s

```
[MLPClassifier(hidden_layer_sizes=(100, 100, 100), max_iter=1000, random_state=1)]: 精度:0.89, 查准率:0.9, 查全率:0.89, f1值:0.89

[0.89, 0.9, 0.89, 0.89]
```

三、实验心得与总结

实验数据的预处理和数据特征的选取是实验中极为关键的一环。好的特征提取可以在很大程度上提升机器学习的效果。虽然现在的模型参数越来越大，数据集越来越大，计算机的算力也在逐年攀升，但做好对数据的分析仍然具有至关重要的意义。此外，我通过此次实验收获到的另外一个感悟就是实验的重要性。无论一个人的理论学习的多么扎实，只有在实验中亲自提出了大胆的想法，做了大胆的尝试，才能

真正的对机器学习的问题产生自己的独到见解，才能更好地驾驭这些令人眼花缭乱的数据处理方法和模型。

四、源代码基本使用方法

1 代码文件解读

`base.ipynb`：完成**基础要求**的相关代码，包含最初的数据预处理方法和朴素贝叶斯验证代码；

`utils.py`：完成**进阶探索**用到的相关代码，包含优化后的数据预处理函数、特征处理函数以及批量训练函数；

`multi.ipynb`：用于进行**第二章实验**的代码，为了加快试验速度，我使用了多进程模块，该代码可以分批完成**第二章1.2节**提到的实验，实验生成的数据文件存储在`result_ker_x.xlsx`中；

`sk_models.ipynb`：用于调用标准模型进行试验，得到结果；

`cus_model.py`：用于实现自己的模型；

`test_cus_models.ipynb`：用于测试自己实现的模型；

`mlp/pca.ipynb`：用于进行**第二章2.2/2.3节**的实验；

`stego_agent.py`：最初的代码，包含了最初分析数据集的实验性代码，**留作纪念。**