机器学习引论期末报告

邓钰川 2021141460159

目录

1 引言

在本次课程报告中，我选择对 MNIST 和 Fasion MNIST 数据集进行研究。使用 SVM 模型，我能够在各自的测试集上实现 97.92% 和 88.28% 的准确率。此外，为了证明 SVM 的优越性，我还采用了其他三种方法进行比较：线性判别感知器、均方误差。此外，我还利用主成分分析和 LDA 技术对数据进行了预处理，并研究了数据扩充对分类模型的影响。

2 方法

2.1 SVM

SVM（支持向量机）方法是一种用于分类的算法，其核心思想是最大化分类间隔。具体地说，SVM 试图找到一个超平面来分割数据集，使得该超平面能够将两个不同类别的数据分隔开来，并且离超平面最近的点到该超平面的距离最大化。

这个距离通常被称为“间隔”，而离超平面最近的点则被称为“支持向量”。SVM 通过寻找这些支持向量来确定最优的超平面，并将它们与其他数据点分开。这样可以提高模型的预测精度和泛化性能。

另外，SVM 还可以通过使用核函数将非线性问题转化为线性问题来进行分类。这是因为有时候数据并不是线性可分的，但是通过使用某些特定的核函数，我们可以将数据映射到高维空间中，从而使其变得线性可分。

2.2 线性判别感知机

线性判别的感知机（Linear Discriminant Perceptron）是一种常见的二分类模型，其基本思想与 SVM 类似。它也是通过一个超平面来将两类数据分开。

具体地，感知机在训练过程中会不断调整超平面的参数，使得对于样本的分类误差最小化。感知机的训练过程通常使用随机梯度下降算法来进行优化。在每次迭代中，感知机会选取一些错误分类的样本进行更新，直到所有的样本都被正确分类或者达到了预设的迭代次数为止。

需要注意的是，感知机方法只能用于线性可分的情况，也就是说，如果样本不是线性可分的，则该方法无法很好地进行分类。此外，感知机对于噪声和异常点比较敏感，容易产生过拟合问题

2.3 均方误差

MSE（Mean Square Error）是一种回归问题中常用的损失函数。在训练模型时，通过计算模型输出结果与真实标签之间的差距（即残差），并将残差的平方求和作为损失函数，进而优化模型参数。

2.4 PCA

PCA（Principal Component Analysis，即主成分分析）是一种常见的数据降维方法。其主要思想是通过线性变换将高维数据映射到低维空间中，使得映射后的数据能够最大程度地保留原始数据的信息。

2.5 LDA

Fisher 方法是一种经典的线性判别方法，也叫做线性判别分析（Linear Discriminant Analysis，LDA）。该方法利用了两类数据之间的信息差异，构建一个线性投影，使得不同类别的数据在降维后的空间中能够更加明显地区分。

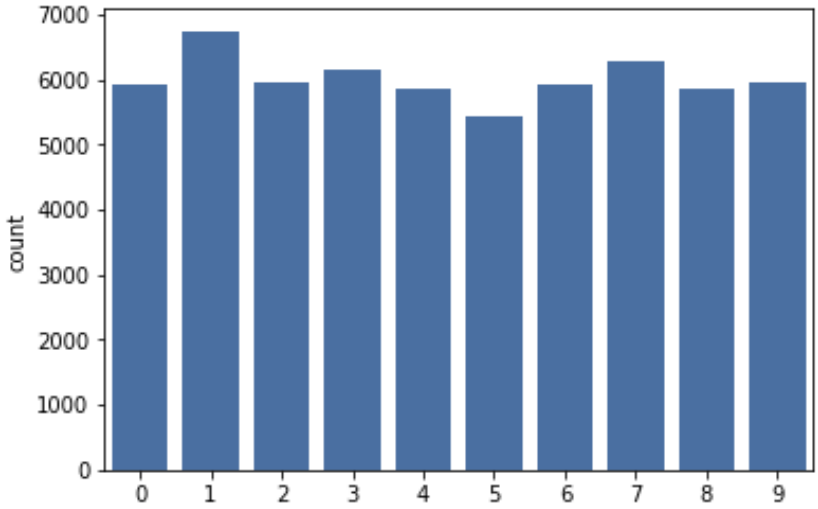


图 1: MNIST 数据分布

3 实验设置

3.1 数据集选择

在数据集上，我们选择了两个数据集，分别是 MNIST 手写数字识别数据集以及其的扩展版 Fashion MNIST. 这两个数据集分别包含了 60000 个训练图像和 10000 个测试图像，每个图像都是一个 28x28 像素的灰度图像，代表了一个手写数字或者一个物品。下图 1 2显示了两个数据集的数据分布情况

3.2 评估协议

我使用准确率 (Accuracy) 作为主要的评估指标，即正确预测的样本数占总样本数的比例。在二分类评估中，我不仅仅关注准确率，还使用 ROC 曲线来评估模型的性能表现。ROC 曲线以不同的阈值下计算出真阳性率（True Positive Rate）和假阳性率（False Positive Rate）之间的权衡，并给出了一个单一的指标，即曲线下面积（AUC）。我在 100 次迭代过程中记录了准确率和 AUC 的变化情况。

在全集评估过程中，我使用多种指标进行评估，并采用混淆矩阵进行可视化。具体而言，我使用 precision，recall，f1-score,accuracy,macro avg,weighted avg 等指标进行评估。其中，precision 表示模型识别出的正例中实际为正例的比例；recall 表示实际为正例的样本被模型识别出的比例；f1-score 是 precision 和 recall 的调和均值；accuracy 是所有正确分类的样本占总样本数的比例。macro avg 和 weighted avg 分别是对所有类别或按照样本权重进行平均的结果，可以帮助评估模型在不同类别或样本权重下的性能。

综上所述，我的评估协议包括准确率、ROC 曲线、precision、recall、f1-score、accuracy、macro avg 和 weighted avg 等指标，并使用混淆矩阵进行可视化。这些指标和可视化工具可以全面评估模型的性能表现。

3.3 实现细节

二分类模型设置 scikit-learn 提供了三个函数：cross val score，cross val predict 和 cross validate。后者提供了有关拟合时间，训练和测试分数的更多信息。为了配合相关的接口，我们先定义一个二分类模型的抽象类别 class Model。

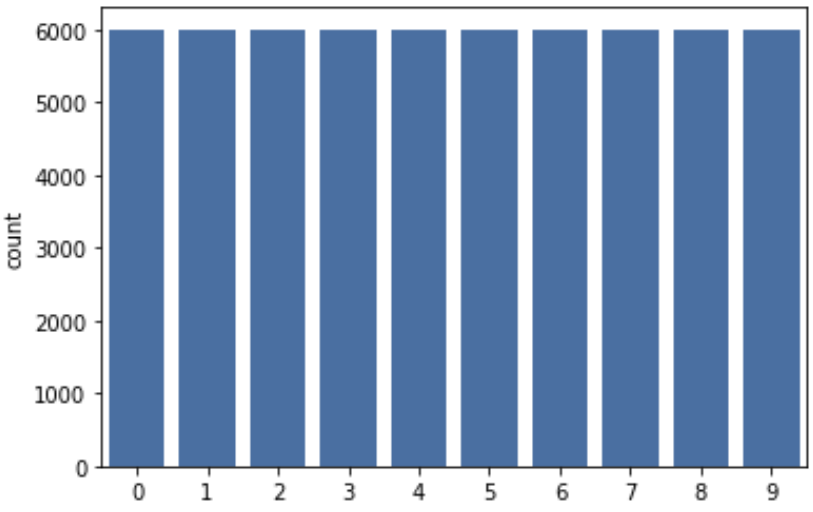


图 2: Fasion MNIST 数据分布

```python
from abc import abstractmethod
class Model:
    loss_saver = []

    @abstractmethod
    def fit(self, X, y):
        pass

    @abstractmethod
    def predict(self, X):
        pass
```

其可以辅助我收集训练过程中的 loss 信息，并提供训练的接口 fit 与预测的接口 predict。然后后续的程序继承这一个模型类，分别定义为 Perceptron，MSE，Fisher，SVM。(具体实现代码参见附录 B)。在训练过程中，我选择 100 次迭代对于每一次二分类的实验。

多分类模型设置多分类有两种实现方式，分别是 OVO(One-vs-One) 与 OVR(One-vs-Rest).

在 OVO 中，每个类别之间都进行二元分类。具体地，对于 k 个不同的类别，会训练 k(k-1)/2 个二元分类器来判断任意两个类别之间的区分度。在测试样本上，每个分类器会投票表决最终属于哪一类。OVO 的优点是每个分类器只需关注两个类别，因此对于大规模问题有利；缺点则是需要训练多个分类器。

而在 OVR 中，每个类别都被视为一个二元分类任务中的“正例”，而其余的所有类别则作为“负例”。具体地，对于 k 个不同的类别，会训练 k 个二元分类器来判断样本是否属于每个类别。在测试样本上，每个分类器会输出一个概率值，最终选择概率最高的类别作为预测结果。OVR 的优点是只需要训练 k 个分类器，并且可以处理不平衡数据集；缺点则是可能会出现重叠或不完整的类别。

考虑到 MNIST 数据集是不平衡的，我选择使用 OVR 的实现方法作为主要的判别方式。

实现硬件所有的实验都在 Legion Y7000 2019 上进行实验，这台机器使用 Intel 酷睿 i5 9300H 作为处理器。

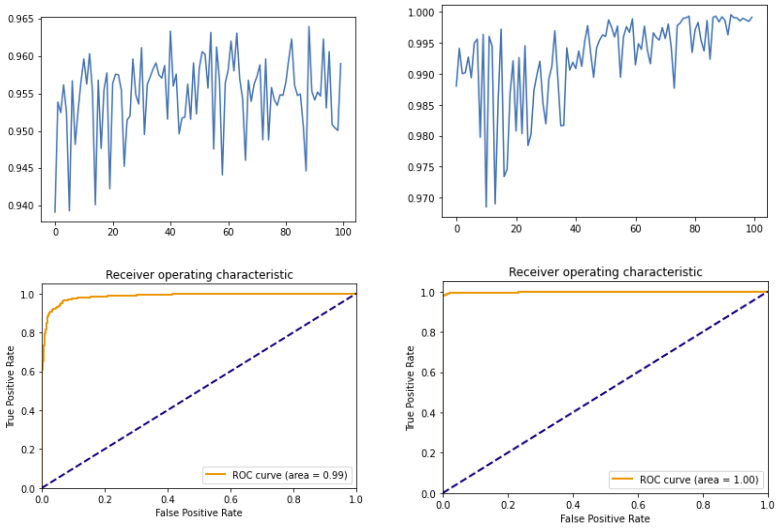
4 实验结果

4.1 二分类实验

我们选择对 5 和 8 进行分类。

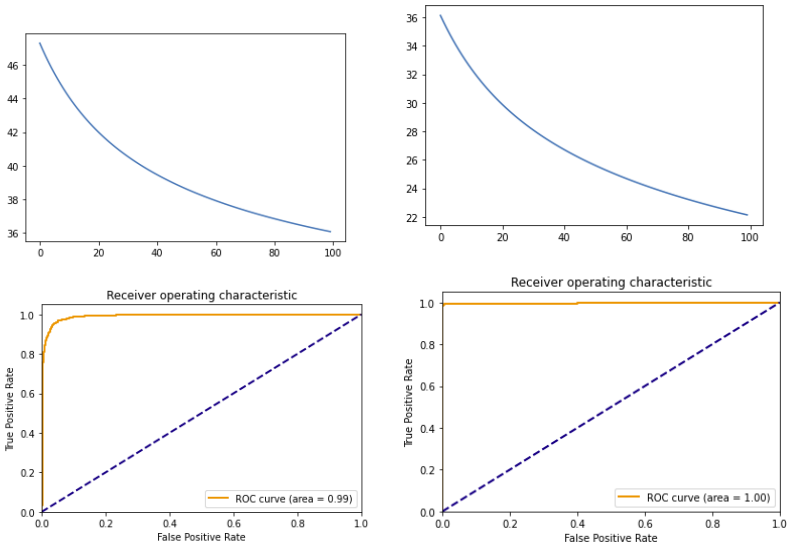
Method	Perceptron	MSE	Fisher	SVM
MNIST	94.05	95.70	95.33	99.57
Fasion MNIST	98.81	99.30	98.9	99.65

表 1: 二分类 accuracy 表现



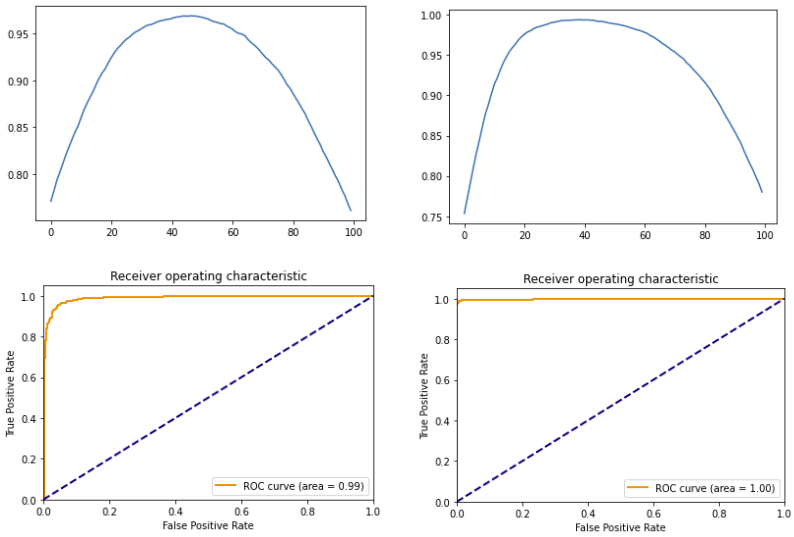
(a) MNIST (b) Fasion MNIST

图 3: 感知机训练迭代图与 ROC 曲线图



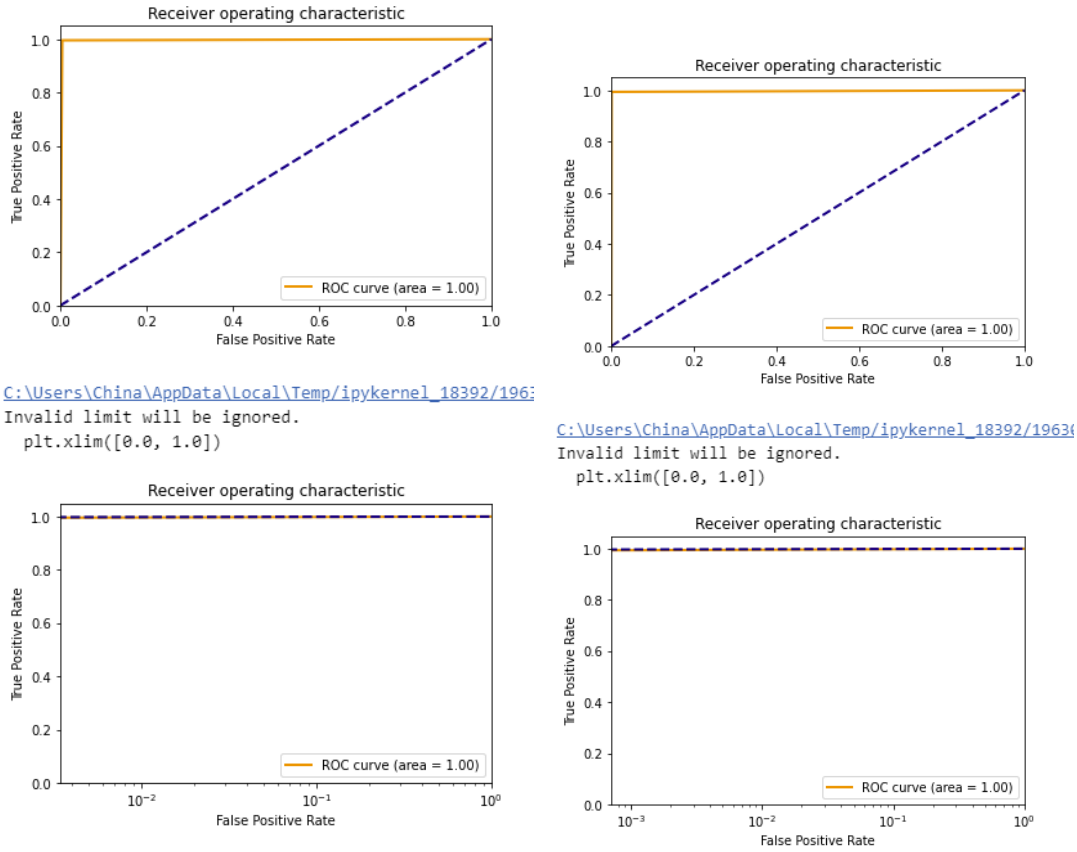
(a) MNIST (b) Fasion MNIST

图 4: MSE 训练迭代图与 ROC 曲线图



(a) MNIST (b) Fasion MNIST

图 5: Fisher+ 感知机训练迭代图与 ROC 曲线图



(a) MNIST (b) Fasion MNIST

图 6: SVM 训练迭代图与 ROC 曲线图

4.2 多分类实验结果

在多分类实验中，我们选择 LDA+ 感知机方法作为 baseline 与 SVM 方法作对比。并引入 PCA 与 LDA 保留 500 个主成分（500/748）进行对比。**LDA+Perceptron**

取得了 82.94% 在 MNIST 上的准确率，0.84 的 f1-score 在 MNIST 上。均方误差是 0.055。取得了 77.01% 在 Fasion MNIST 上的准确率，0.78 的 f1-score 在 Fasion MNIST 上。总训练时长为 1min5s，对每个数字速度平均为 3.85it/s。

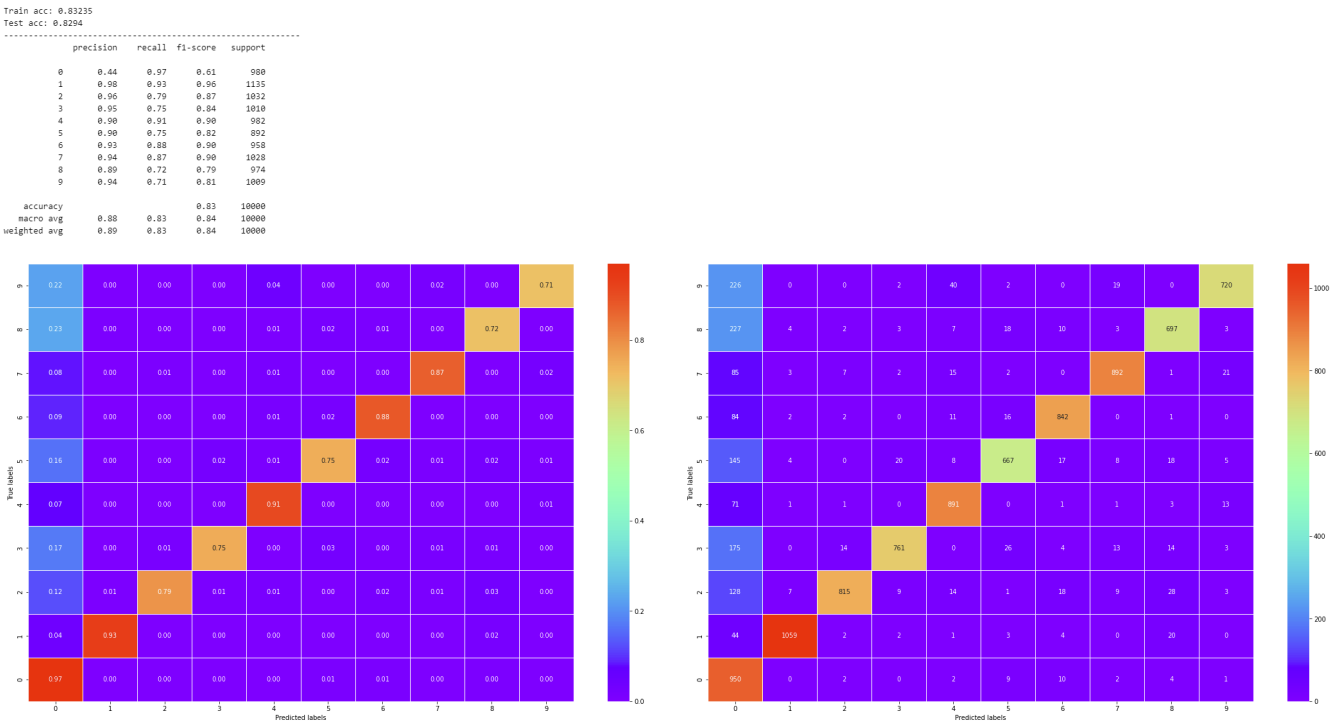


图 7: LDA+Perceptron on MNIST

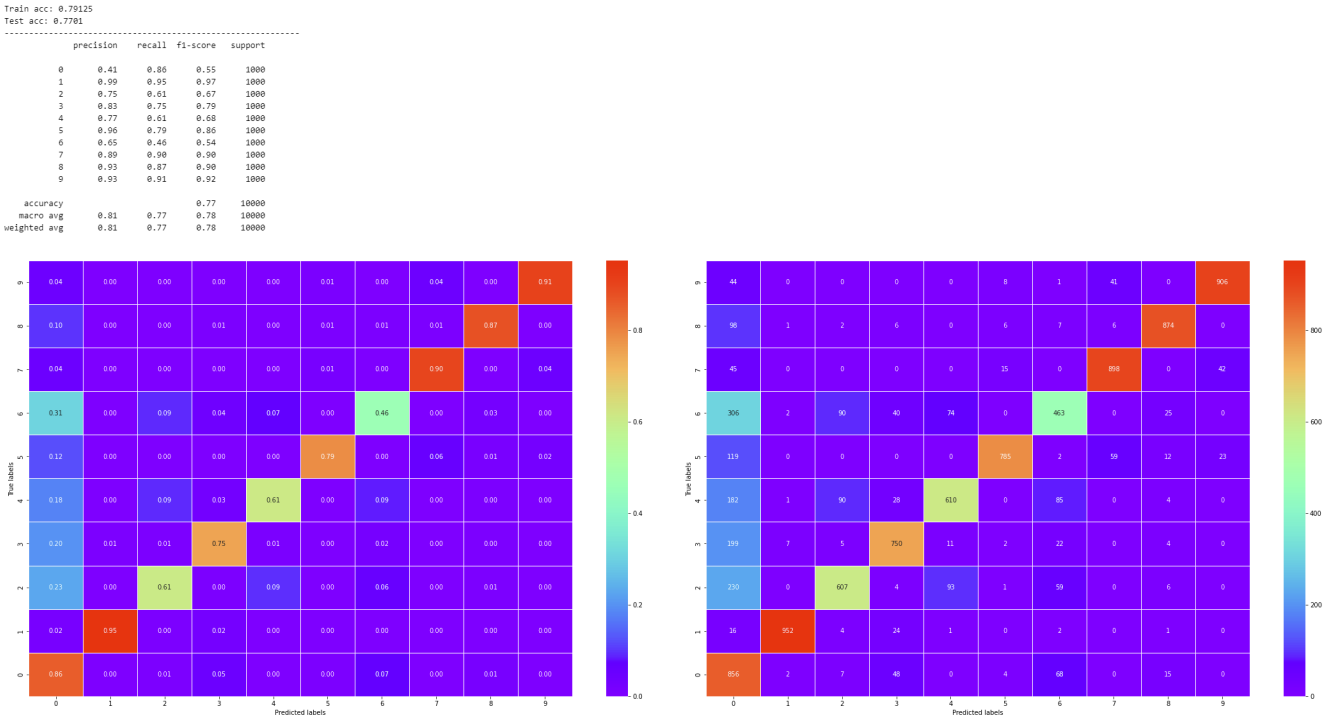


图 8: LDA+Perceptron on Fasion MNIST

SVM

取得了 97.92% 在 MNIST 上的准确率, 0.84 的 f1-score 在 MNIST 上, 0 的均方误差。取得了 88.28% 在 Fasion MNIST 上的准确率，0.88 的 f1-score 在 Fasion MNIST 上。总训练时长为 3min20s。

```
Train acc: 0.9899166666666667
Test acc: 0.9792
--------------------------------------------------------
              precision    recall  f1-score   support

           0       0.98      0.99      0.99       980
           1       0.99      0.99      0.99      1135
           2       0.98      0.97      0.98      1032
           3       0.97      0.99      0.98      1010
           4       0.98      0.98      0.98       982
           5       0.99      0.98      0.98       892
           6       0.99      0.99      0.99       958
           7       0.98      0.97      0.97      1028
           8       0.97      0.98      0.97       974
           9       0.97      0.96      0.97      1009

    accuracy                           0.98     10000
   macro avg       0.98      0.98      0.98     10000
weighted avg       0.98      0.98      0.98     10000
```

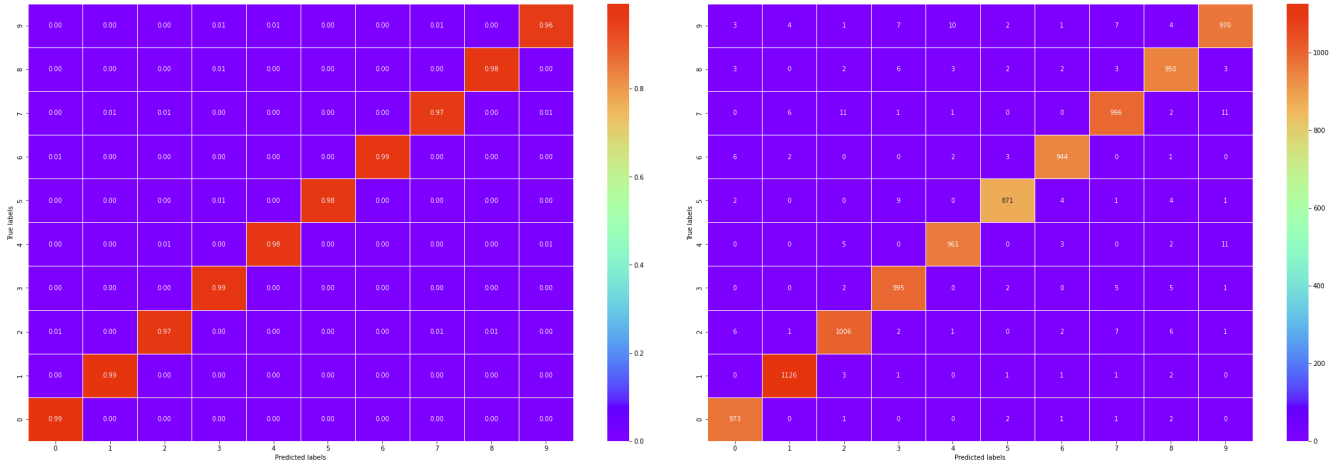


图 9: SVM on MNIST

```
Train acc: 0.9128
Test acc: 0.8828
--------------------------------------------------------
              precision    recall  f1-score   support

           0       0.83      0.86      0.84      1000
           1       0.99      0.96      0.98      1000
           2       0.79      0.82      0.80      1000
           3       0.87      0.89      0.88      1000
           4       0.81      0.81      0.81      1000
           5       0.96      0.95      0.96      1000
           6       0.72      0.65      0.69      1000
           7       0.93      0.95      0.94      1000
           8       0.97      0.98      0.97      1000
           9       0.96      0.95      0.96      1000

    accuracy                           0.88     10000
   macro avg       0.88      0.88      0.88     10000
weighted avg       0.88      0.88      0.88     10000
```

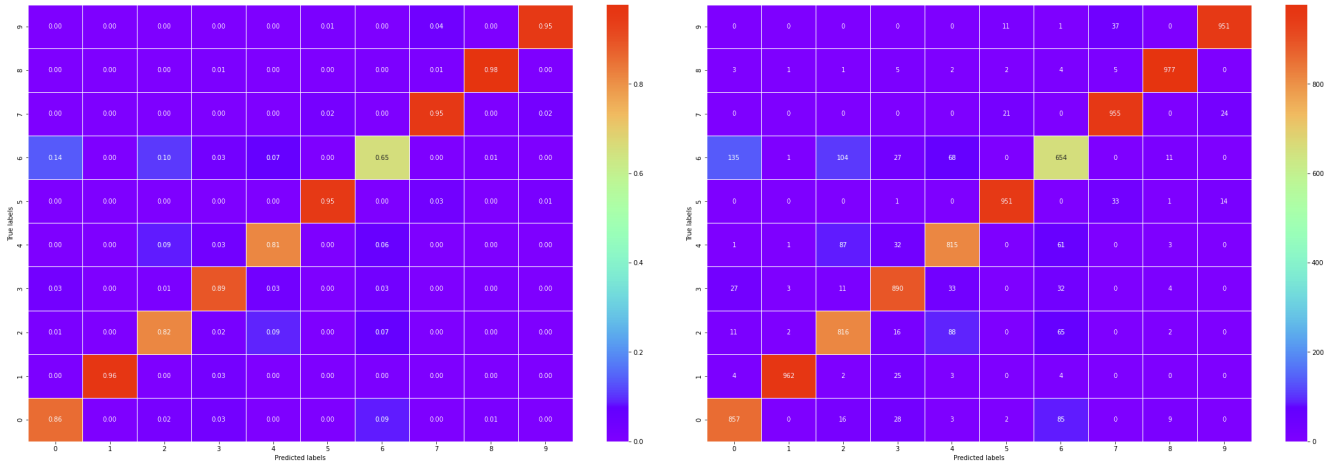


图 10: SVM on Fasion MNIST

PCALDA

我们可视化了 PCA 与 LDA 成分占比

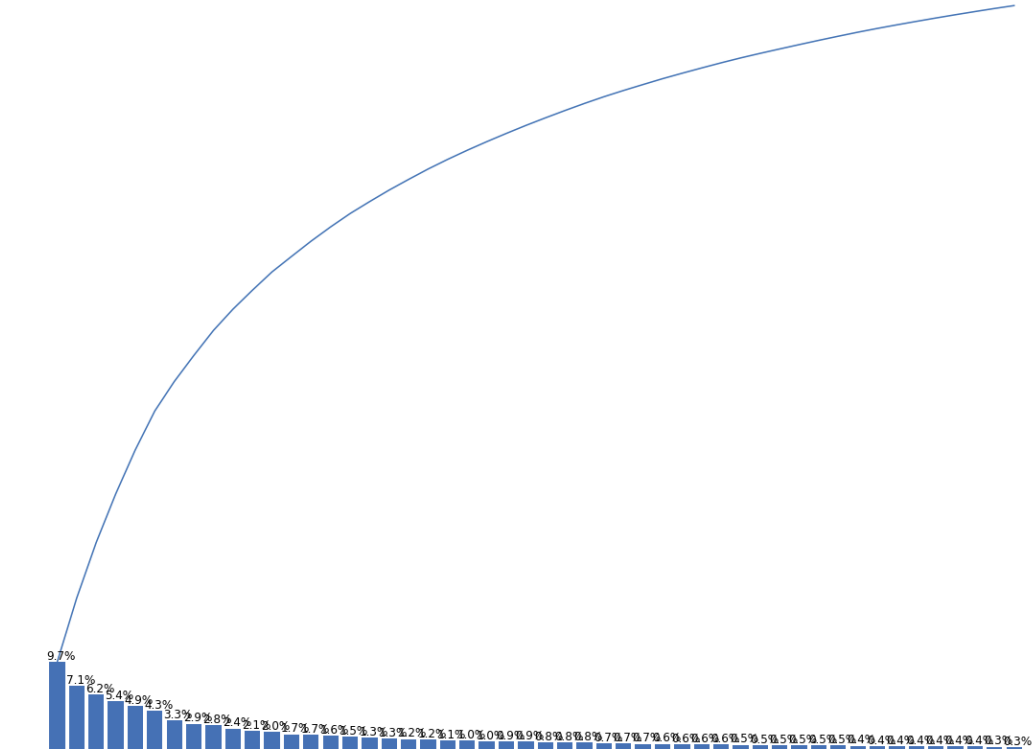


图 11: PCA 成分占比

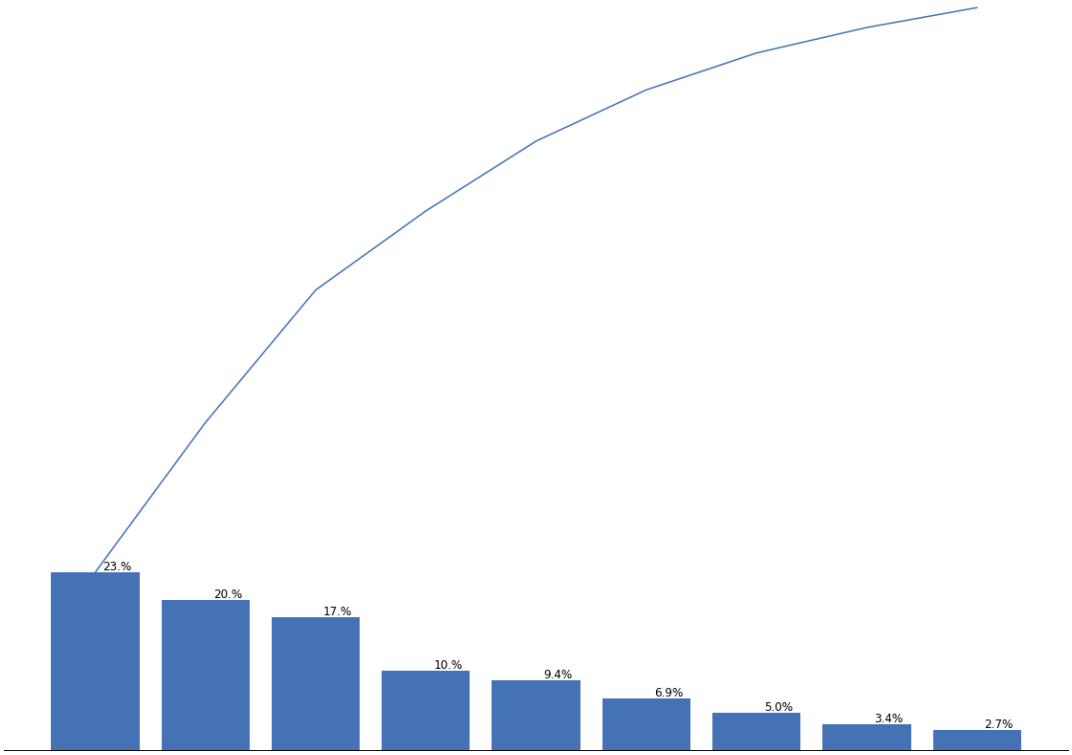


图 12: LDA 成分占比

对于 PCA 我们选取前两个主成分进行可视化

可以看出 SVM 效果最好，SVM+LDA 的效果次之，LDA+Perceptron 效果更次，PCA+SVM 效果最不好。我的分析是因为手写数字识别这个任务本身比较简单，LDA 和 PCA 更多时候是在杂乱无章的背景下进行实验，保留主成分去除噪声。而在本实验中，这两者反而去掉了相关的特征向量，使得 SVM 的性能下降。

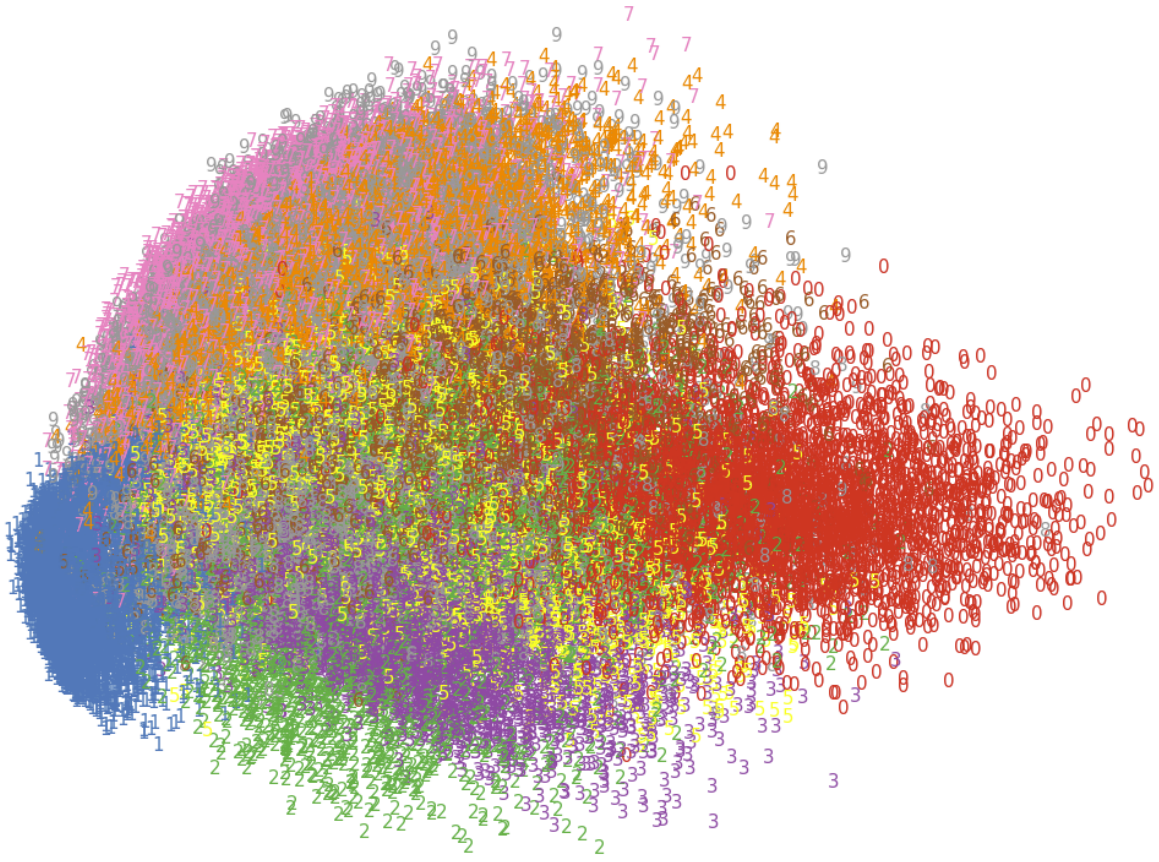


图 13: PCA 前两个主成分区分

```
Train acc: 0.8933166666666666
Test acc: 0.841
--------------------------------------------------------
              precision    recall  f1-score   support

           0       0.78      0.79      0.79      1000
           1       0.96      0.96      0.96      1000
           2       0.75      0.76      0.75      1000
           3       0.83      0.84      0.83      1000
           4       0.74      0.78      0.76      1000
           5       0.93      0.91      0.92      1000
           6       0.64      0.56      0.60      1000
           7       0.92      0.93      0.93      1000
           8       0.92      0.94      0.93      1000
           9       0.94      0.94      0.94      1000

    accuracy                           0.84     10000
   macro avg       0.84      0.84      0.84     10000
weighted avg       0.84      0.84      0.84     10000
```

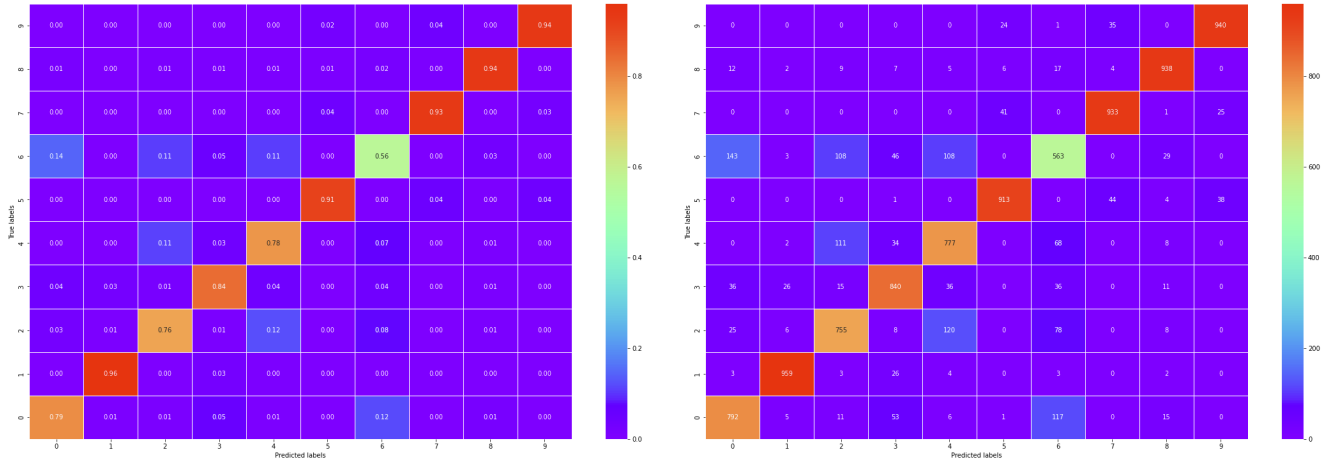


图 14: SVM+PCA on MNIST

```
Train acc: 0.8936833333333334
Test acc: 0.8735
--------------------------------------------------------
              precision    recall  f1-score   support

           0       0.96      0.88      0.92       980
           1       0.93      0.81      0.87      1135
           2       0.90      0.90      0.90      1032
           3       0.88      0.90      0.89      1010
           4       0.91      0.90      0.90       982
           5       0.86      0.82      0.84       892
           6       0.91      0.94      0.93       958
           7       0.74      0.91      0.82      1028
           8       0.83      0.87      0.85       974
           9       0.86      0.81      0.84      1009

    accuracy                           0.87     10000
   macro avg       0.88      0.87      0.87     10000
weighted avg       0.88      0.87      0.87     10000
```

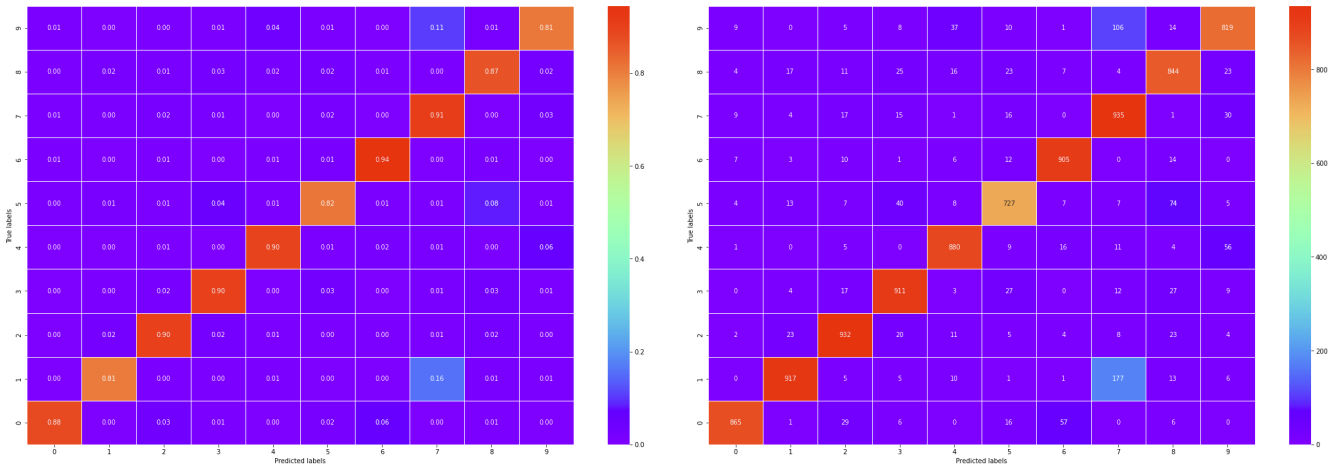


图 15: SVM+LDA on MNIST

A 数据读取与可视化代码

A.1 数据读取代码

```
# 外部依赖项
import numpy as np
import matplotlib.pyplot as plt
import time
from sklearn import metrics
# 定义一个偏差常量
EPS = 1E-8
import struct
# 读取 MNIST 与 Fasion_Mnist 的函数
## 读取图片部分
def mnist_load_images(file_name):
    # 在读取或写入一个文件之前，你必须使用 Python 内置 open() 函数来打开它。
    # file object = open(file_name [, access_mode][, buffering])
    # file_name 是包含您要访问的文件名的字符串值。                    ##
    # access_mode 指定该文件已被打开，即读，写，追加等方式。          ##
    # 0 表示不使用缓冲，1 表示在访问一个文件时进行缓冲。              ##
    # 这里 rb 表示只能以二进制读取的方式打开一个文件
    binfile = open(file_name, 'rb')
    ##   从一个打开的文件读取数据
    buffers = binfile.read()
    ##   读取 image 文件前 4 个整型数字
    magic, num, rows, cols = struct.unpack_from('>IIII', buffers, 0)
    ##   整个 images 数据大小为 60000*28*28
    bits = num * rows * cols
    ##   读取 images 数据
    images = struct.unpack_from('>' + str(bits) + 'B', buffers, struct.calcsize('>IIII'))
    ##   关闭文件
    binfile.close()
    ##   转换为 [60000,784] 型数组
    images = np.reshape(images, [num, rows * cols])
    return images

## 读取标记部分
def mnist_load_labels(file_name):
    ## 打开文件
    binfile = open(file_name, 'rb')
    ##   从一个打开的文件读取数据
```

```
    buffers = binfile.read()
    ##   读取 label 文件前 2 个整形数字，label 的长度为 num
    magic, num = struct.unpack_from('>II', buffers, 0)
    ##   读取 labels 数据
    labels = struct.unpack_from('>' + str(num) + "B", buffers, struct.calcsize('>II'))
    ##   关闭文件
    binfile.close()
    ##   转换为一维数组
    labels = np.reshape(labels, [num])
    return labels

def load_minst():
    mnist_train_image, mnist_train_label = mnist_load_images("dataset/mnist_dataset/train-images.idx3-ubyte
                        mnist_load_labels("dataset/mnist_dataset/train-labels.idx1-ubyte")
    mnist_test_image, mnist_test_label = mnist_load_images("dataset/mnist_dataset/t10k-images.idx3-ubyte")
                      mnist_load_labels("dataset/mnist_dataset/t10k-labels.idx1-ubyte")

    return mnist_train_image, mnist_train_label, mnist_test_image, mnist_test_label

def load_fasionminst():
    fasionmnist_train_image, fasionmnist_train_label = mnist_load_images("dataset/fasionmnist_dataset/trai
                        mnist_load_labels("dataset/fasionmnist_dataset/train-labels-idx1-ubyte")
    fasionmnist_test_image, fasionmnist_test_label = mnist_load_images("dataset/fasionmnist_dataset/t10k-i
                      mnist_load_labels("dataset/fasionmnist_dataset/t10k-labels-idx1-ubyte")

    return fasionmnist_train_image, fasionmnist_train_label,fasionmnist_test_image, fasionmnist_test_label

```

A.2 数据分布查看代码

```
# mnist
import seaborn as sns
sns.countplot(mnist_train_label, color = sns.color_palette()[0])
# fasionmnist

sns.countplot(fasionmnist_train_label, color = sns.color_palette()[0])
# mnist
plt.imshow(mnist_train_image[5].reshape(28,28))
plt.axis('off')# 关闭坐标轴
print('The digit in the image is {}'.format(mnist_test_image[0]))# 格式化打印
```

B 预处理与模型构建代码

```
# PCA
from sklearn.decomposition import PCA
pca = PCA(.90)  # .90 represents decomposition ratio
mnist_pca_dimentional_train1 = pca.fit_transform(mnist_train_image)
print(pca.explained_variance_)
print("n_components : ", mnist_pca_dimentional_train1.shape[1])
scree_plot(pca)
# Inverse Transfrom from 64 -> 784

apprx = pca.inverse_transform(mnist_pca_dimentional_train1)
plt.imshow(apprx[5].reshape(28,28))

from sklearn.decomposition import PCA

pca = PCA(n_components=50)   # Max Componenet values = 728
mnist_pca_dimentional_train2 = pca.fit_transform(mnist_train_image)
print(pca.explained_variance_)
print(mnist_pca_dimentional_train2.shape, type(mnist_pca_dimentional_train2))
scree_plot(pca)
# Inverse Transfrom from 64 -> 784

apprx = pca.inverse_transform(mnist_pca_dimentional_train2)
plt.imshow(apprx[5].reshape(28,28))

def plot_components(X, y):
    x_min, x_max = np.min(X, 0), np.max(X, 0)
    X = (X - x_min) / (x_max - x_min)
    plt.figure(figsize=(20, 15))
    for i in range(X.shape[0]):
        plt.text(X[i, 0], X[i, 1], str(y[i]),
                 color=plt.cm.Set1(y[i]),
                 fontdict={'size': 15})

    plt.xticks([]), plt.yticks([]), plt.ylim([-0.1,1.1]), plt.xlim([-0.1,1.1])
plot_components(mnist_pca_dimentional_train2, mnist_train_label)

from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
lda = LinearDiscriminantAnalysis()
```

```

result_lda = lda.fit(mnist_train_image,mnist_train_label).transform(mnist_train_image)
scree_plot(lda)
from sklearn.utils.validation import check_is_fitted
from sklearn.utils import check_array, check_X_y

def inverse_transform(lda, x):
    if lda.solver == 'lsqr':
        raise NotImplementedError("(inverse) transform not implemented for 'lsqr' "
                                  "solver (use 'svd' or 'eigen').")
    check_is_fitted(lda, ['xbar_', 'scalings_'], all_or_any=any)

    inv = np.linalg.pinv(lda.scalings_)

    print(inv.shape)
    x = check_array(x)
    print(x.shape)
    if lda.solver == 'svd':
        x_back = np.dot(x, inv) + lda.xbar_
    elif lda.solver == 'eigen':
        x_back = np.dot(x, inv)

    return x_back

apprx_lda = inverse_transform(lda,result_lda)
plt.imshow(apprx_lda[5].reshape(28,28))
lda = LinearDiscriminantAnalysis(n_components=8)    # Max Componenet values = 728
mnist_lda_dimentional_train = lda.fit(mnist_train_image,mnist_train_label).transform(mnist_train_image)
mnist_lda_dimentional_test = lda.fit(mnist_test_image,mnist_test_label).transform(mnist_test_image)
```

B.1 基本模型定义

我们先定义一个模型的抽象类别

```
from abc import abstractmethod
class Model:
    loss_saver = []

    @abstractmethod
    def fit(self, X, y):
        pass
```

```

    @abstractmethod
    def predict(self, X):
        pass
```

B.2 感知机

```
from tqdm import tqdm, trange

class Perceptron(Model):
    # 模型定义
    def __init__(self, alpha=0.03, n_iter=20):
        self.alpha = alpha # 学习率
        self.n_iter = n_iter # 迭代次数
        self.loss_saver = [] # 损失收集

    # 模型训练
    def fit(self, X, Y):
        m, n = np.shape(X)
        self.bias = 0
        self.W = np.ones(n)

        for i in tqdm(range(self.n_iter)):
            for x, y in zip(X, Y):
                y_hat = np.dot(x, self.W) + self.bias

                if y * y_hat < 0.:
                    self.W += x * self.alpha * y
                    self.bias += self.alpha * y

            self.loss_saver.append(self.loss(X, Y))

            if self.loss(X, Y) < EPS:
                return

    def predict(self, X):
        return np.dot(X, self.W) + self.bias

    def loss(self, X, Y):
        return metrics.accuracy_score(Y, np.sign(self.predict(X)))
```

B.3 MSE

```
from scipy import linalg
class MSE(Model):
    def __init__(self, c=0.03, n_iter=20):
        self.c = c
        self.n_iter = n_iter
        self.loss_saver = []

    def fit(self, X, Y):
        m, n = X.shape
        b = np.ones((m, 1))
        X = np.concatenate([X, b], axis=1)
        Y = Y.reshape((-1, 1))
        X = X * Y

        self.b = np.ones(m, dtype=np.float)
        X_shape = linalg.pinv(X)

        for i in trange(self.n_iter):
            self.W = np.dot(X_shape, self.b)
            self.err = np.dot(X, self.W) - self.b
            self.b = self.b + self.c * (self.err + np.abs(self.err))
            self.loss_saver.append(np.linalg.norm(self.err, ord=2))
            if self.loss_saver[-1] < EPS:
                return

    def predict(self, X):
        b = np.ones((X.shape[0], 1))
        X = np.concatenate([X, b], axis=1)
        return np.dot(X, self.W)
```

B.4 Fisher

```
from scipy import linalg
class Fisher(Model):
    def __init__(self, iter_n = 10):
        self.c = 0.
        self.iter_n = iter_n
        self.loss_saver = []
```

```
7
8       @staticmethod
9       def _cal_cov_avg(X):
10          u = np.mean(X, axis=0)
11          cov = np.cov(X, rowvar=False)
12
13          return cov, u
14
15      def fit(self, X, Y):
16          Y = Y.reshape((-1, 1))
17          X = X / 255
18          X_full = np.concatenate([X, Y], axis=1)
19          X_0 = X_full[X_full[:, -1] == -1]
20          X_1 = X_full[X_full[:, -1] == 1]
21          X_0, X_1 = X_0[:, :-1], X_1[:, :-1]
22          cov_0, u_0 = self._cal_cov_avg(X_0)
23          cov_1, u_1 = self._cal_cov_avg(X_1)
24
25          s_w = cov_0 + cov_1
26          s_w_inv = linalg.pinv(s_w)
27          self.W = np.dot(s_w_inv, u_0 - u_1)
28          self.u_0 = np.dot(u_0, self.W)
29          self.u_1 = np.dot(u_1, self.W)
30          acc = 0.
31          c = 0.
32
33          for i in trange(self.iter_n):
34              self.c = i / self.iter_n
35              l_acc = self.loss(X * 255, Y)
36              self.loss_saver.append(l_acc)
37              if l_acc > acc:
38                  c = self.c
39                  acc = l_acc
40
41          self.c = c
42
43      def loss(self, X, Y):
44          return metrics.accuracy_score(Y, np.sign(self.predict(X)))
45
46      def predict(self, X):
47          X = X / 255
```

```
48          return -(np.dot(X, self.W) - (self.u_0 + (self.u_1 - self.u_0) * self.c))
```

B.5 OVR

```
1  class OVR(Model):
2      def __init__(self, model, n_class):
3          self.model = model
4          self.n_class = n_class
5
6          self.models = []
7
8          for i in range(n_class):
9              self.models.append(model())
10
11     def fit(self, X, Y):
12         Y = Y.reshape((-1, 1))
13         X_full = np.concatenate([X, Y], axis=1)
14
15         for i in range(self.n_class):
16             X_o = X_full[X_full[:, -1] == i].copy()
17             X_r = X_full[X_full[:, -1] != i].copy()
18
19             X_o, X_r = X_o[:, :-1], X_r[:, :-1]
20             Y_o, Y_r = np.ones((X_o.shape[0], 1)), -np.ones((X_r.shape[0], 1))
21             tmp_data_o = np.concatenate([X_o, Y_o], axis=1)
22             tmp_data_r = np.concatenate([X_r, Y_r], axis=1)
23             tmp_data = np.concatenate([tmp_data_r, tmp_data_o], axis=0)
24             np.random.shuffle(tmp_data)
25             tmp_X, tmp_Y = tmp_data[:, :-1], tmp_data[:, -1]
26             self.models[i].fit(tmp_X, tmp_Y)
27
28     def predict(self, X):
29         n = X.shape[0]
30         res, maxx = np.zeros(n), np.zeros(n)
31
32         for i in range(self.n_class):
33             tmp_res = self.models[i].predict(X)
34
35             for j in range(n):
36                 if tmp_res[j] > maxx[j]:
```

```
                    res[j], maxx[j] = i, tmp_res[j]

        return res
```

B.6 SVM

```
from tqdm import trange

class SVM(Model):
    def __init__(self, n_iter=100, kernel='linear'):
        self.max_iter = n_iter
        self._kernel = kernel

    def init_args(self, features, labels):
        self.m, self.n = features.shape
        self.X = features
        self.Y = labels
        self.b = 0.0

        self.alpha = np.ones(self.m)
        self.E = [self._E(i) for i in range(self.m)]
        self.C = 1.0

    def _KKT(self, i):
        y_g = self._g(i) * self.Y[i]
        if self.alpha[i] == 0:
            return y_g >= 1
        elif 0 < self.alpha[i] < self.C:
            return y_g == 1
        else:
            return y_g <= 1

    def _g(self, i):
        r = self.b
        for j in trange(self.m):
            r += self.alpha[j] * self.Y[j] * self.kernel(self.X[i], self.X[j])
        return r

    def kernel(self, x1, x2):
        if self._kernel == 'linear':
```

```
            return sum([x1[k] * x2[k] for k in range(self.n)])
        elif self._kernel == 'poly':
            return (sum([x1[k] * x2[k] for k in range(self.n)]) + 1)**2

        return 0

    def _E(self, i):
        return self._g(i) - self.Y[i]

    def _init_alpha(self):
        index_list = [i for i in range(self.m) if 0 < self.alpha[i] < self.C]
        non_satisfy_list = [i for i in range(self.m) if i not in index_list]
        index_list.extend(non_satisfy_list)

        for i in index_list:
            if self._KKT(i):
                continue

            E1 = self.E[i]
            if E1 >= 0:
                j = min(range(self.m), key=lambda x: self.E[x])
            else:
                j = max(range(self.m), key=lambda x: self.E[x])
            return i, j

    def _compare(self, _alpha, L, H):
        if _alpha > H:
            return H
        elif _alpha < L:
            return L
        else:
            return _alpha

    def fit(self, features, labels):
        self.init_args(features, labels)

        for t in trange(self.max_iter):
            i1, i2 = self._init_alpha()

            if self.Y[i1] == self.Y[i2]:
                L = max(0, self.alpha[i1] + self.alpha[i2] - self.C)
```

```
76                 H = min(self.C, self.alpha[i1] + self.alpha[i2])
77             else:
78                 L = max(0, self.alpha[i2] - self.alpha[i1])
79                 H = min(self.C, self.C + self.alpha[i2] - self.alpha[i1])
80
81             E1 = self.E[i1]
82             E2 = self.E[i2]
83             eta = self.kernel(self.X[i1], self.X[i1]) + self.kernel(
84                 self.X[i2],
85                 self.X[i2]) - 2 * self.kernel(self.X[i1], self.X[i2])
86             if eta <= 0:
87                 continue
88
89             alpha2_new_unc = self.alpha[i2] + self.Y[i2] * (
90                 E1 - E2) / eta
91             alpha2_new = self._compare(alpha2_new_unc, L, H)
92
93             alpha1_new = self.alpha[i1] + self.Y[i1] * self.Y[i2] * (
94                 self.alpha[i2] - alpha2_new)
95
96             b1_new = -E1 - self.Y[i1] * self.kernel(self.X[i1], self.X[i1]) * (
97                 alpha1_new - self.alpha[i1]) - self.Y[i2] * self.kernel(
98                     self.X[i2],
99                     self.X[i1]) * (alpha2_new - self.alpha[i2]) + self.b
100            b2_new = -E2 - self.Y[i1] * self.kernel(self.X[i1], self.X[i2]) * (
101                alpha1_new - self.alpha[i1]) - self.Y[i2] * self.kernel(
102                    self.X[i2],
103                    self.X[i2]) * (alpha2_new - self.alpha[i2]) + self.b
104
105            if 0 < alpha1_new < self.C:
106                b_new = b1_new
107            elif 0 < alpha2_new < self.C:
108                b_new = b2_new
109            else:
110                b_new = (b1_new + b2_new) / 2
111
112            self.alpha[i1] = alpha1_new
113            self.alpha[i2] = alpha2_new
114            self.b = b_new
115
116            self.E[i1] = self._E(i1)
```

```
117             self.E[i2] = self._E(i2)
118
119     def predict(self, data):
120         r = self.b
121         for i in range(self.m):
122             r += self.alpha[i] * self.Y[i] * self.kernel(data, self.X[i])
123
124         return 1 if r > 0 else -1
125
126     def _weight(self):
127         yx = self.Y.reshape(-1, 1) * self.X
128         self.w = np.dot(yx.T, self.alpha)
129         return self.w
```

C 模型评估与可视化代码

C.1 二分类

```
print("Start evaluating.")
y_ = np.sign(model.predict(train_image))

print("Train acc: {}".format(metrics.accuracy_score(train_label, y_)))
t_y = model.predict(test_image)
print("Test acc: {}".format(metrics.accuracy_score(test_label, np.sign(t_y))))

print("-" * 60)

plt.figure()
x = np.arange(len(model.loss_saver))
y = np.array(model.loss_saver)
plt.plot(x, y)
plt.show()

fpr, tpr, _ = metrics.roc_curve(test_label, t_y)
roc_auc = metrics.auc(fpr, tpr)
lw = 2
plt.figure()
plt.plot(fpr, tpr, color='darkorange',
     lw=lw, label='ROC curve (area = %0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1], color='navy', lw=lw, linestyle='--')
plt.xlim([0.0, 1.0])
```

```
    plt.ylim([0.0, 1.05])
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('Receiver operating characteristic')
    plt.legend(loc="lower right")
    plt.show()

    plt.figure()
    plt.semilogx(fpr, tpr, color='darkorange',
         lw=lw, label='ROC curve (area = %0.2f)' % roc_auc)
    plt.plot([0, 1], [0, 1], color='navy', lw=lw, linestyle='--')
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('Receiver operating characteristic')
    plt.legend(loc="lower right")
    plt.show()

def evaluate2two(model:Model, train_image, train_label, test_image, test_label):
    print("Start evaluating.")
    y_ = np.sign(model.predict(train_image))

    print("Train acc: {}".format(metrics.accuracy_score(train_label, y_)))
    t_y = model.predict(test_image)
    print("Test acc: {}".format(metrics.accuracy_score(test_label, np.sign(t_y))))

    print("-" * 60)

    fpr, tpr, _ = metrics.roc_curve(test_label, t_y)
    roc_auc = metrics.auc(fpr, tpr)
    lw = 2
    plt.figure()
    plt.plot(fpr, tpr, color='darkorange',
         lw=lw, label='ROC curve (area = %0.2f)' % roc_auc)
    plt.plot([0, 1], [0, 1], color='navy', lw=lw, linestyle='--')
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('Receiver operating characteristic')
```

```
    plt.legend(loc="lower right")
    plt.show()

    plt.figure()
    plt.semilogx(fpr, tpr, color='darkorange',
         lw=lw, label='ROC curve (area = %0.2f)' % roc_auc)
    plt.plot([0, 1], [0, 1], color='navy', lw=lw, linestyle='--')
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('Receiver operating characteristic')
    plt.legend(loc="lower right")
    plt.show()


def train2one(model:Model, train_image, train_label, test_image, test_label):
    # training
    print("-" * 60)
    print("Train starting.")
    model.fit(train_image,train_label)
    print("Train finished.")

    # evaluating
    evaluate2one(model, train_image, train_label, test_image, test_label)

def train2two(model:Model, train_image, train_label, test_image, test_label):
    # training
    print("-" * 60)
    print("Train starting.")
    model.fit(train_image,train_label)
    print("Train finished.")

    # evaluating
    evaluate2two(model, train_image, train_label, test_image, test_label)
```

C.2 多分类

```
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report

```

```
def trainN(model, train_image, train_label, test_image, test_label):
    #training
    print("-" * 60)
    print("Train starting.")
    model.fit(train_image, train_label)
    print("Train finished.")
    print("Evaluate starting.")

    ty = model.predict(train_image)
    dy = model.predict(test_image)

    print("Train acc: {}".format(metrics.accuracy_score(train_label,ty)))
    print("Test acc: {}".format(metrics.accuracy_score(test_label,dy)))

    print("-" * 60)
    print(classification_report(test_label,dy))
    con_mat = confusion_matrix(test_label, dy)
    con_mat_norm = con_mat.astype('float') / con_mat.sum(axis=1)[:, np.newaxis]

    fig = plt.figure()
    ax1 = fig.add_subplot(1, 2, 1)
    sns.heatmap(con_mat_norm, annot=True, fmt='.2f', cmap='rainbow', lw=.5)
    ax1.set_ylim(0, 10)
    ax1.set_xlabel('Predicted labels')
    ax1.set_ylabel('True labels')
    ax1.figure.set_size_inches(30,10)

    ax2 = fig.add_subplot(1, 2, 2)
    sns.heatmap(con_mat, annot=True, fmt='.0f', cmap='rainbow', lw=.5)
    ax2.set_ylim(0, 10)
    ax2.set_xlabel('Predicted labels')
    ax2.set_ylabel('True labels')
    ax2.figure.set_size_inches(30,10)

    fig.tight_layout(pad=0.4, w_pad=3.0, h_pad=3.0)
```

```
def scree_plot(pca):
    num_components = len(pca.explained_variance_ratio_)
    ind = np.arange(num_components)
    vals = pca.explained_variance_ratio_
```

```

    plt.figure(figsize=(20, 15))
    ax = plt.subplot(111)
    cumvals = np.cumsum(vals)
    ax.bar(ind, vals)
    ax.plot(ind, cumvals)
    for i in range(num_components):
        ax.annotate(r"%s%%" % ((str(round(vals[i]*100,1))[:3])), (ind[i]+0.2, vals[i]),
                    va="bottom",
                    ha="center",
                    fontsize=12)

    ax.xaxis.set_tick_params(width=0)
    ax.yaxis.set_tick_params(width=1, length=6)

    ax.set_xlabel("Principal Component")
    ax.set_ylabel("Variance Explained (%)")
    plt.title('Explained Variance Per Principal Component')
```