基于 Fashion-MNIST 数据集的分类问题研究

李国庭，16051216, 数据挖掘作业四

摘　要　本文针对 Fashion-MNIST 数据集上的分类问题，分别使用 Python 实现和调用 sklearn 库实现 SVM(Support Vector Machine)分类算法进行了仿真实验。然后通过对不同核函数和参数的选择，发现使用高斯核的 SVM 算法在本数据集上表现最好。

关键词　图像分类；　SVM；高斯核；线性核

0　引言

图像分类在图像检索，目标检测等诸多领域都是一项非常重要的任务，该技术拥有非常广阔的应用前景。针对图像分类任务，目前常用的分类算法有 KNN, Logistic 回归，SVM 等。上述算法在以往针对经典的 MNIST 数据集都取得了很好的分类效果，本文基于新的 Fashion-MNIST[1]数据集，分别使用 Python 实现和调用 sklearn 库实现 SVM(Support Vector Machine)分类算法进行仿真实验。

1　SVM 算法简介

1.1　SVM基本思想：间隔最大化

支持向量机(Support Vector Machines, SVM)一种常见的判别方法。其核心思想在于：在特征空间中，寻找距离决策边界（分隔超平面）最近的点（支持向量），使得支持向量到分隔超平面的距离最大化。[3]

对应的优化模型为：

$$argmax_{w,b}\left(\min_{n}\left(label(w^T+b)\right)\frac{1}{||w||}\right) \tag{3.1}$$

应用拉格朗日乘子法将其转换为对偶问题：

$$\max_{\alpha}\left(\sum_{i=1}^{m}\alpha-\frac{1}{2}\sum_{i,j=1}^{m}\text{label}^{(i)}\text{label}^{(j)}\,\alpha_i\alpha_j<x^{(i)},x^{(j)}>\right) \tag{3.2}$$

$$st.\,\alpha\geq 0,\sum_{i=1}^{m}\alpha_i\text{label}^{(i)}=0$$

引入松弛变量，允许某些数据点处于分隔面错误一侧，约束条件变为[2]：

$$st.C \geq \alpha \geq 0, \sum_{i=1}^{m} \alpha_i \text{label}^{(i)} = 0$$

1.2 径向基核函数[2]

对于一些无法用线性模型进行分类的数据集，我们可以尝试用非线性模型进行分类。此时我们需要借助核函数将数据映射到高维空间，从而将一个非线性不可分的问题转换为非线性可分问题。

径向基核函数是 SVM 中常用的一个核函数，定义如下：

$$k(x,y) = \exp\left(\frac{-||x-y||^2}{2\sigma^2}\right) \tag{3.3}$$

其中，σ是用户定义的用于确定到达率或者说函数值跌落到 0 的速度参数。为了下文方便说明，这里我们定义：

$$gamma = \frac{1}{2\sigma^2} \tag{3.4}$$

1.3 SMO优化算法

SMO 算法是用于训练 SVM 的优化算法之一。它将大优化问题分解为多个小优化问题来求解，并且对它们进行顺序求解的结果与将它们作为整体来求解的结果是一致的。在结果完全相同的同时，SMO 算法的求解时间短很多。[2]步骤如下：

第一步选取一对α_i和α_j，选取方法使用启发式方法。第二步，固定除α_i和α_j之外的其他参数，确定 W 极值条件下的α_i，α_j由α_i表示。

2 实验说明

本文使用了 Fasion-MNIST[1]作为基准数据集，在该数据集上测试分类算法的效果。Fashion-MNIST 是一个替代 MNIST 手写数字集的图像数据集，由德国时尚科技公司 Zalando 旗下的研究部门提供。其涵盖了来自 **10** 种类别的共 **7** **万**个不同商品的正面图片，按 60000/10000 的训练测试数据划分，28 x 28的灰度图片。如下图所示。

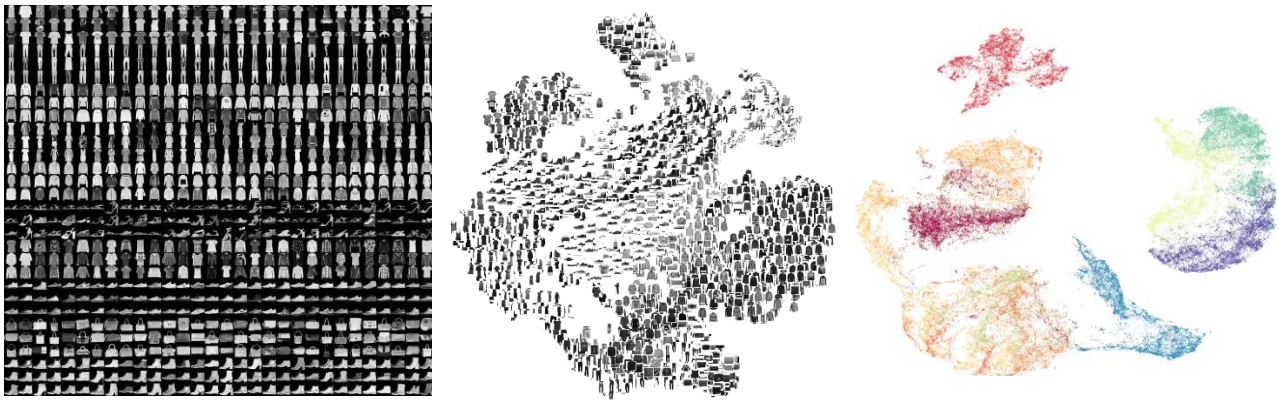

图 1　Fasion-MNIST 数据集（左）t-SNE 可视化（中）UMAP 可视化（右）

数据集标签字段说明如下表。

表 1　数据集标签字段说明

标签	说明	标签	说明
0	T-shirt	5	Scandal
1	Trouser	6	Shirt
2	Pullover	7	Sneaker
3	Dress	8	Bag
4	Coat	9	Ankle boot

本文在每个类别中随机选择 700 张图片，并将其划分为训练集和测试集分别为 6000 张和 1000 张，利用三种不同的分类算法，统计在测试集上的分类精度。

3　SVM 分类结果及分析

作者使用了 OVO 策略训练了 45 个 SVM 模型进行多分类。结果表明，在使用高斯核，且σ=10 , C=100 时有较好的分类效果，而线性核在相同 C 下，表现大大逊于高斯核。

表 4　SVM 分类结果及分析

内核	C	准确率
RBF, σ=10	100	0.8080

同时，在调整参数时，我们发现，C 参数越大，σ越小，训练准确率越高，但是越容易过拟合。虽然 SVM 有很高的准确率，但对于规模较大的数据集，训练时间很长。

4　利用 sklearn[4]对各个分类器进行评价

由于上述三种分类器均是作者通过 Python 实现，没有很好地对算法进行优化，故在比较

不同分类器的分类精度时，会存在一定偏差，这种偏差并不是算法本身带来的。因此，我们为了避免评价时的偏差，使用了目前流行的 sklearn 库中的相应算法对数据集进行分类并比较分类效果。

结果如下：

表 5　不同分类器分类结果比较

分类器	准确率
KNN(k=8)	0.8310
SVM(‘rbf’)	0.8590
SVM(‘poly’)	0.8170
SVM(‘linear’)	0.7510

从上表可以看出，使用高斯核函数的 SVM 分类器，分类精度最高。

5　总结

本文针对 Fashion-MNIST 数据集上的分类问题，分别使用 Python 实现和调用 sklearn 库实现 SVM(Support Vector Machine)分类算法进行了仿真实验。对于 SVM 分类器，使用线性核的效果要劣于高斯核，其原因在于该数据集有非线性的特性，无法使用线性模型进行分类。尽管作者实现的 SVM 限于编程和理论水平没有突出优于其他分类器，但是从 Sklearn 的结果来看，SVM 的分类效果要远远高于其他分类器。

参 考 文 献

[1] Xiao, Han , K. Rasul , and R. Vollgraf . "Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms." (2017).

[2] 哈林顿李锐. 机器学习实战 ： Machine learning in action[M]. 人民邮电出版社, 2013.

[3] 李航. 统计学习方法[M]. 清华大学出版社, 2012.

[4] Pedregosa F , Gaël Varoquaux, Gramfort A , et al. Scikit-learn: Machine Learning in Python[J]. Journal of Machine Learning Research, 2012, 12(10):2825-2830.

附 录

```
import numpy as np
import pandas as pd
import random
def kernelTrans(X, A, kTup):
    m,n = np.shape(X)
    K = np.mat(np.zeros((m,1)))
    if kTup[0]=='lin': K = X * A.T
    elif kTup[0]=='rbf':
        for j in range(m):
            deltaRow = X[j,:] - A
            K[j] = deltaRow*deltaRow.T
        K = np.exp(K/(-1*kTup[1]**2))
    else: raise NameError('Houston We Have a Problem -- That Kernel is not recognized')
    return K

def clipAlpha(aj,H,L):
    if aj > H:
        aj = H
    if L > aj:
        aj = L
    return aj

def calcEk(oS, k):
    fXk = float(np.multiply(oS.alphas,oS.labelMat).T*oS.K[:,k] + oS.b)
    Ek = fXk - float(oS.labelMat[k])
    return Ek

def updateEk(oS, k):
    Ek = calcEk(oS, k)
    oS.eCache[k] = [1,Ek]

def selectJrand(i,m):
    j=i
```

```
        while (j==i):
            j = int(random.uniform(0,m))
        return j
    class optStruct:
        def __init__(self,dataMatIn, classLabels, C, toler, kTup):
            self.X = dataMatIn
            self.labelMat = classLabels
            self.C = C
            self.tol = toler
            self.m = np.shape(dataMatIn)[0]
            self.alphas = np.mat(np.zeros((self.m,1)))
            self.b = 0
            self.eCache = np.mat(np.zeros((self.m,2)))
            self.K = np.mat(np.zeros((self.m,self.m)))
            for i in range(self.m):
                self.K[:,i] = kernelTrans(self.X, self.X[i,:], kTup)

    def selectJ(i, oS, Ei):
        maxK = -1; maxDeltaE = 0; Ej = 0
        oS.eCache[i] = [1,Ei]
        validEcacheList = np.nonzero(oS.eCache[:,0].A)[0]
        if (len(validEcacheList)) > 1:
            for k in validEcacheList:
                if k == i: continue
                Ek = calcEk(oS, k)
                deltaE = np.abs(Ei - Ek)
                if (deltaE > maxDeltaE):
                    maxK = k; maxDeltaE = deltaE; Ej = Ek
            return maxK, Ej
        else:
            j = selectJrand(i, oS.m)
            Ej = calcEk(oS, j)
        return j, Ej

    def innerL(i, oS):
```

```
        Ei = calcEk(oS, i)
        if ((oS.labelMat[i]*Ei < -oS.tol) and (oS.alphas[i] < oS.C)) or ((oS.labelMat[i]*Ei > oS.tol)
and (oS.alphas[i] > 0)):
            j,Ej = selectJ(i, oS, Ei)
            alphaIold = oS.alphas[i].copy(); alphaJold = oS.alphas[j].copy();
            if (oS.labelMat[i] != oS.labelMat[j]):
                L = max(0, oS.alphas[j] - oS.alphas[i])
                H = min(oS.C, oS.C + oS.alphas[j] - oS.alphas[i])
            else:
                L = max(0, oS.alphas[j] + oS.alphas[i] - oS.C)
                H = min(oS.C, oS.alphas[j] + oS.alphas[i])
            if L==H: return 0
            eta = 2.0 * oS.K[i,j] - oS.K[i,i] - oS.K[j,j]
            if eta >= 0: return 0
            oS.alphas[j] -= oS.labelMat[j]*(Ei - Ej)/eta
            oS.alphas[j] = clipAlpha(oS.alphas[j],H,L)
            updateEk(oS, j)
            if (np.abs(oS.alphas[j] - alphaJold) < 0.00001): return 0
            oS.alphas[i] += oS.labelMat[j]*oS.labelMat[i]*(alphaJold - oS.alphas[j])
            updateEk(oS, i)
            b1 = oS.b - Ei- oS.labelMat[i]*(oS.alphas[i]-alphaIold)*oS.K[i,i] -
oS.labelMat[j]*(oS.alphas[j]-alphaJold)*oS.K[i,j]
            b2 = oS.b - Ej- oS.labelMat[i]*(oS.alphas[i]-alphaIold)*oS.K[i,j]-
oS.labelMat[j]*(oS.alphas[j]-alphaJold)*oS.K[j,j]
            if (0 < oS.alphas[i]) and (oS.C > oS.alphas[i]): oS.b = b1
            elif (0 < oS.alphas[j]) and (oS.C > oS.alphas[j]): oS.b = b2
            else: oS.b = (b1 + b2)/2.0
            return 1
        else: return 0

    def smoP(dataMatIn, classLabels, C, toler, maxIter,kTup=('lin', 0)):
        oS = optStruct(np.mat(dataMatIn),np.mat(classLabels).transpose(),C,toler, kTup)
        iter = 0
        entireSet = True; alphaPairsChanged = 0
        while (iter < maxIter) and ((alphaPairsChanged > 0) or (entireSet)):
```

```
            alphaPairsChanged = 0
            if entireSet:
                for i in range(oS.m):
                    alphaPairsChanged += innerL(i,oS)
                    print ("fullSet, iter: %d i:%d, pairs changed %d" %
(iter,i,alphaPairsChanged))
                iter += 1
            else:
                nonBoundIs = np.nonzero((oS.alphas.A > 0) * (oS.alphas.A < C))[0]
                for i in nonBoundIs:
                    alphaPairsChanged += innerL(i,oS)
                    print ("non-bound, iter: %d i:%d, pairs changed %d" %
(iter,i,alphaPairsChanged))
                iter += 1
            if entireSet: entireSet = False
            elif (alphaPairsChanged == 0): entireSet = True
            print ("iteration number: %d" % iter)
        return oS.b,oS.alphas

    def norm(df):
        dat = df.iloc[:,1:].copy()
        label = df.iloc[:,0].copy()
        dat /= 255
        return pd.concat([label, dat], axis=1)

    def trainSVM(trainData, C=200, toler=0.0001, maxIter=10000, kTup=('rbf', 10)):
        dataArr = trainData[2][:,1:].copy()
        labelArr = list(trainData[2][:,0].copy())
        b,alphas = smoP(dataArr, labelArr, C, toler, maxIter, kTup)
        datMat=np.mat(dataArr); labelMat = np.mat(labelArr).transpose()
        svInd=np.nonzero(alphas.A>0)[0]
        sVs=datMat[svInd]
        labelSV = labelMat[svInd];
        print ("there are %d Support Vectors" % np.shape(sVs)[0])
        model = {'alphas':alphas, 'b':b, 'svInd':svInd, 'sVs':sVs, 'labelSV':labelSV,
```

```
'label_x':trainData[0], 'label_y':trainData[1], 'kTup':kTup}
        return model
    def predict(kTup, C):
        models = []
        for i in range(len(trainSet)):
            print("Training Model1: %d" % i)
            models.append(trainSVM(trainSet[i], C=C, kTup=kTup))
        print("Train and Test..")
        datMat = np.array(train_set.iloc[:,1:]); labelMat = np.array(train_set.iloc[:,0])
        m,n = np.shape(datMat)
        errorCount = 0
        for i in range(m):
            count = [0]*classNum
            for j in range(len(models)):
                sVs = models[j]['sVs']; kTup = models[j]['kTup']; labelSV = models[j]['labelSV']
                svInd = models[j]['svInd']; alphas = models[j]['alphas']; b = models[j]['b']
                label_x = models[j]['label_x']; label_y = models[j]['label_y']
                kernelEval = kernelTrans(sVs,datMat[i,:],kTup)
                predict=kernelEval.T * np.multiply(labelSV,alphas[svInd]) + b
                if(np.sign(predict) == 1):
                    count[label_x] += 1
                else:
                    count[label_y] += 1
            if(np.argmax(count) != labelMat[i]):
                errorCount += 1
        print ("the training error rate is: %f" % (float(errorCount)/m))

        datMat = np.array(test_set.iloc[:,1:]); labelMat = np.array(test_set.iloc[:,0])
        m,n = np.shape(datMat)
        errorCount = 0
        for i in range(m):
            count = [0]*classNum
            for j in range(len(models)):
                sVs = models[j]['sVs']; kTup = models[j]['kTup']; labelSV = models[j]['labelSV']
                svInd = models[j]['svInd']; alphas = models[j]['alphas']; b = models[j]['b']
```

```
                label_x = models[j]['label_x']; label_y = models[j]['label_y']
                kernelEval = kernelTrans(sVs,datMat[i,:],kTup)
                predict=kernelEval.T * np.multiply(labelSV,alphas[svInd]) + b
                if(np.sign(predict) == 1):
                    count[label_x] += 1
                else:
                    count[label_y] += 1
            if(np.argmax(count) != labelMat[i]):
                errorCount += 1
        print ("the testing error rate is: %f" % (float(errorCount)/m))

df = pd.read_csv('train.csv')
test_set = pd.read_csv('test.csv')
train_set = pd.read_csv('train.csv')

df = norm(df)
test_set = norm(test_set)
train_set = norm(train_set)

train = []
classNum = len(df['label'].drop_duplicates())
for i in range(classNum):
    train.append(df[df['label'] == i])
trainSet = []
for i in range(classNum-1):
    for j in range(i+1, classNum):
        temp_i = np.array(train[i]).copy()
        temp_j = np.array(train[j]).copy()
        temp_i[:,0] = 1.0
        temp_j[:,0] = -1.0
        temp = np.concatenate([temp_i,temp_j])
        trainSet.append((i,j, temp))

predict(kTup=('rbf', 10), C=100)
```