PCA特征降维&J48特征选择算法实现报告

周政

1.PCA算法

协方差矩阵及其特征向量和特征值计算函数

```
#协方差矩阵及其特征向量和特征值计算函数
def CovFeatureVecsVals(X):
  mean_vec = np.mean(X, axis=0)
  #归一化
  for i in range(len(X)):
    X[i,:] = (X[i,:] - mean_vec[:])
  cov_mat = np.cov(X.T)
  eig_vals, eig_vecs = np.linalg.eig(cov_mat)
  #组成特征值和特征向量对(通过numpy.linalg.eig()计算出的特征值和特征向量是从大到小对应排好序的.)
  eig_pairs = [(eig_vals[i], eig_vecs[:,i]) for i in range(len(eig_vals))]
  return eig_pairs
```

传入数据集X，将特征列减去其均值减少差异程度，然后计算各特征的协方差矩阵及其特征向量与特征值，组成协方差矩阵的特征值与特征向量的tuple组返回

降维矩阵计算函数

```
15  #降维矩阵计算函数:
16  def ReduceDimension(eig_pairs,d):
17    matrix_w = np.hstack(
18      (eig_pairs[i][1].reshape(4,1) for i in range(d))
19    )
20    return matrix_w
```

传入装有特征值与特征向量对的list,以及要降的维度d,来构建降维矩阵并返回，得到的降维矩阵是一个nxd的矩阵.

PCA降维调度函数

```python
#PCA降维调度函数
def RunPCA(data,d):
    X = np.array(data,dtype = 'float64')
    d = int(d)
    if d >= X.shape[1] or d <= 0:
        return 0
    #计算协方差矩阵及其特征向量和特征值
    eig_pairs = CovFeatureVecsVals(X)
    #计算降维矩阵
    matrix_w = ReduceDimension(eig_pairs,d)
    #降维
    Y = X.dot(matrix_w)
    return Y

```

通过传入初始数据集和需要降维的维数对各函数进行调用最后计算出降维后的矩阵Y(mxd).

Python文件执行程序及画图

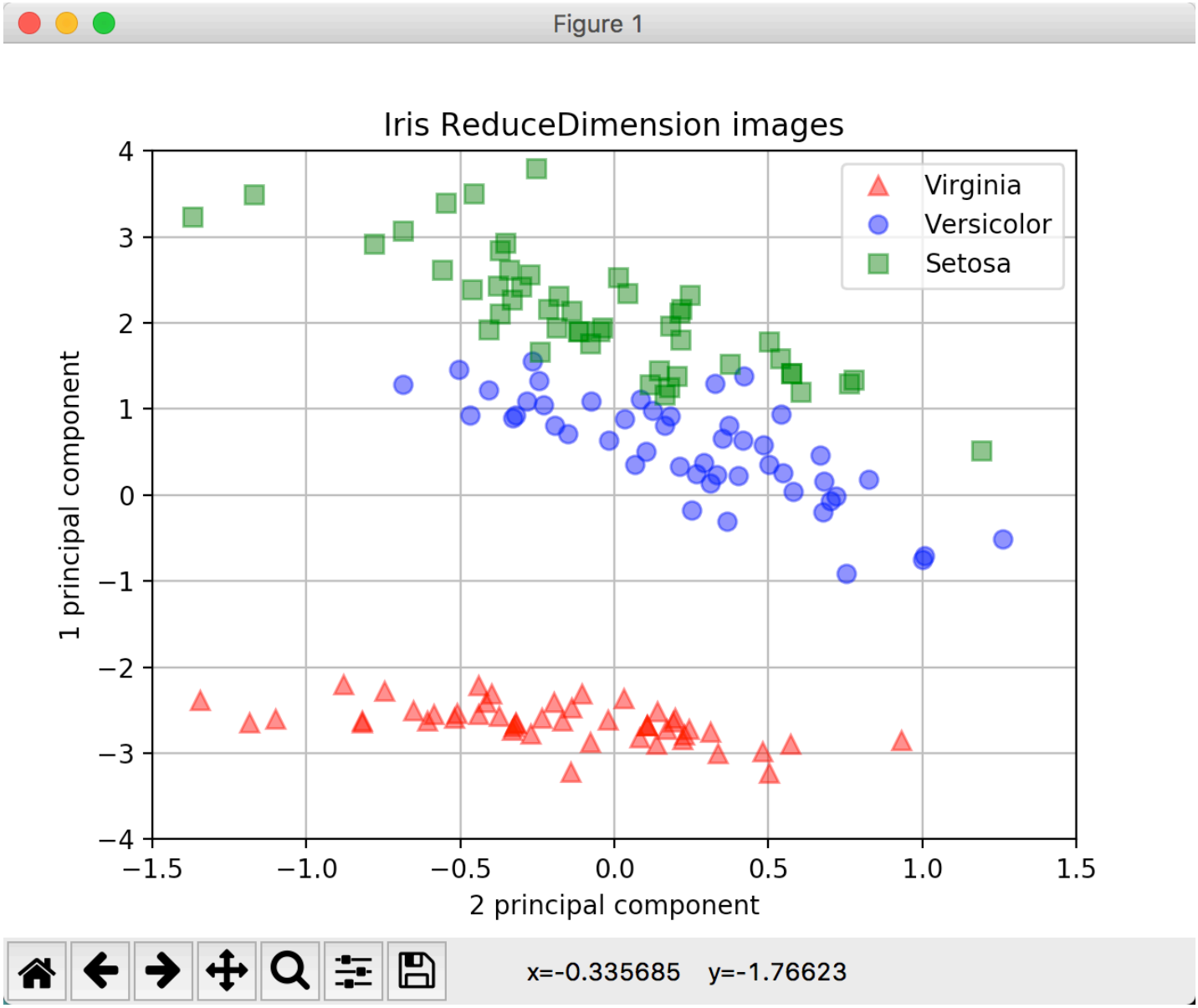
```
if __name__ == '__main__':
  #导入数据
  data = []
  with open('dataset/iris.data') as f:
    for i in f.readlines():
      data.append(i.strip().split(',')[:-1])
  #运行PCA调度函数
  Y = RunPCA(data,2)
  if type(Y) == type(0):
      print("无法正确降维！")
  else:
    print(Y)
    plt.plot(Y[0:49,1],Y[0:49,0], '^', markersize=7, color='red', alpha=0.5,
      label='Virginia')
    plt.plot(Y[50:99,1],Y[50:99,0], 'o', markersize=7, color='blue', alpha=0.5,
      label='Versicolor')
    plt.plot(Y[100:149,1],Y[100:149,0], 's', markersize=7, color='green', alpha=0.5,
      label='Setosa')

    plt.xlim([-1.5,1.5])
    plt.ylim([-4,4])

    plt.xlabel('2 principal component')
    plt.ylabel('1 principal component')
    plt.legend()
    plt.title('Dois componentes principais da Base de Dados Iris')

    plt.tight_layout
    plt.grid()
    plt.show()
```

结果



1.利用J48进行特征选择

之前作业写好的J48决策树

```python
#香农熵计算
def ShannonEntropy(dataSet):
    labelCounts = {} #记录标签出现次数
    dataNum = len(dataSet) #数据条数
    for data in dataSet:
        cLabel = data[-1]
        if cLabel not in labelCounts.keys():
            labelCounts[cLabel] = 0
        labelCounts[cLabel] += 1
    SE = 0.0
    for key in labelCounts:
        percent = float(labelCounts[key])/dataNum
        SE -= percent*log(percent,2)
    return SE
```

```python
#划分数据集
def DiviDataSet(dataSet, feature, v):
    dividDataSet = []
    for data in dataSet:
        if data[feature] == v:
            tempDataSet = data[:feature]
            tempDataSet.extend(data[feature+1:])
            dividDataSet.append(tempDataSet)
    return dividDataSet
```

```python
#选择根节点属性来划分数据集
def selectBestFeature(dataSet):
    numAttr = len(dataSet[0]) - 1
    baseEnt = ShannonEntropy(dataSet)
    bestInfoInc = 0.0
    bestFeature = -1
    for i in range(numAttr):
        uValues = set([example[i] for example in dataSet])
        newEnt = 0.0
        for value in uValues:
            subDataSet = DiviDataSet(dataSet, i, value)
            percent = len(subDataSet)/float(len(dataSet))
            newEnt += percent * ShannonEntropy(subDataSet)
            InfoInc = baseEnt - newEnt
        if (InfoInc > bestInfoInc):
            bestInfoInc = InfoInc
            bestFeature = i
    return bestFeature
```

```python
#计算出现次数最多的分类
def majorityCnt(classList):
    classCount = {}
    for v in classList:
        if v not in classCount.keys():
            classCount[v] = 0
        classCount[v] += 1
    sortedClassCount = sorted(classCount.iteritems(), key=operator.itemgetter(1),
        reverse=True)
    return sortedClassCount[0][0]
```

```python
#建树
def createTree(dataSet,labels,n):
    classList = [example[-1] for example in dataSet]
    if classList.count(classList[0]) == len(classList):
        return classList[0]
    if len(dataSet[0]) == 1:
        return majorityCnt(classList)
    bestFeat = selectBestFeature(dataSet)
    bestFeatLabel = labels[bestFeat]
    if bestFeatLabel not in FeatureSubset:
        if len(FeatureSubset) == n:
            return
        FeatureSubset.append(bestFeatLabel)
    DTree = {bestFeatLabel:{}}
    del(labels[bestFeat])
    featValues = [example[bestFeat] for example in dataSet]
    uniqueVals = set(featValues)
    for value in uniqueVals:
        subLabels = labels[:]
        DTree[bestFeatLabel] [value] = createTree(DiviDataSet(dataSet, bestFeat,
            value),subLabels,n)
    return DTree
```

但对原来的递归建树函数做了一点修改，新加了传入参数n来获取前n层节点所用到的特征装入FeatureSubset中并在FeatureSubset长度到达n后return不再递归返回结果.

结果

```
Last login: Tue Aug 22 12:20:13 on ttys003
 Zain@MacBook-Pro  ~  wb/J48\&PCA
 Zain@MacBook-Pro  ~/wb/J48&PCA  master  py3 J48FeatureSelector.py
['safety', 'persons', 'buying', 'maint']
 Zain@MacBook-Pro  ~/wb/J48&PCA  master  _
```

Github地址