目录

伪代码：

（1）计算已知类别的数据集中的点与当前点之间的距离；
（2）按照距离递增次序排序；
（3）选取与当前点距离最小的k个点；
（4）确定前k个点所在类别出现的频率；
（5）返回前k个点出现频率最高的类别作为当前点的预测分类。

k近邻算法实现：

```
def knnClassify(inX, dataSet, labels, k):
    #(1) 计算已知数据和测试点数据的距离（采用欧式距离计算）
    #采用numpy矩阵计算的方式快速计算测试点数据和每个已知数据的欧式距离
    dataSetSize = dataSet.shape[0]
    diffMat = np.tile(inX,(dataSetSize,1))-dataSet
    sqDiffMat = diffMat**2
    sqDistances = sqDiffMat.sum(axis=1)
    distances = sqDistances**0.5

    # (2) 按照距离大小排序
    sortedDistIndicies = distances.argsort() #获得从小到大排序的索引

    # (3) 选取距离最小的前k个点，统计他们的label和次数
    classCount = {}
    for i in range(k):
        voteLabel = labels[sortedDistIndicies[i]]
        classCount[voteLabel] = classCount.get(voteLabel,0)+1

    # (4) 返回字典classCount中频率最高的label作为预测结果
    sortedClassCount = sorted(classCount.iteritems(),key=operator.itemgetter(1),reverse=True)
    return sortedClassCount[0][0]
```

实例

实例1：使用k近邻算法改进约会网站的配对效果

背景：

约会网站会根据人选的特征数据赋予不同的标签。
在本实例中使用了3个特征：（1）每年的飞行里程数（2）玩游戏时间的占用时间比（3）每周消费的冰淇淋公升数；
赋予3种标签：不喜欢、一般、喜欢

任务：

（1）准备数据：把数据特征从文本文件中读取出来
（2）分析数据：使用Matplotlib创建散点图
（3）测试数据：使用k近邻算法验证数据集

准备数据

数据保存在txt文件中，每个样本占据一行，总共有1000行，每个特征用用'\t'隔开，最后一个为label数据。需要把数据保存到numpy矩阵中

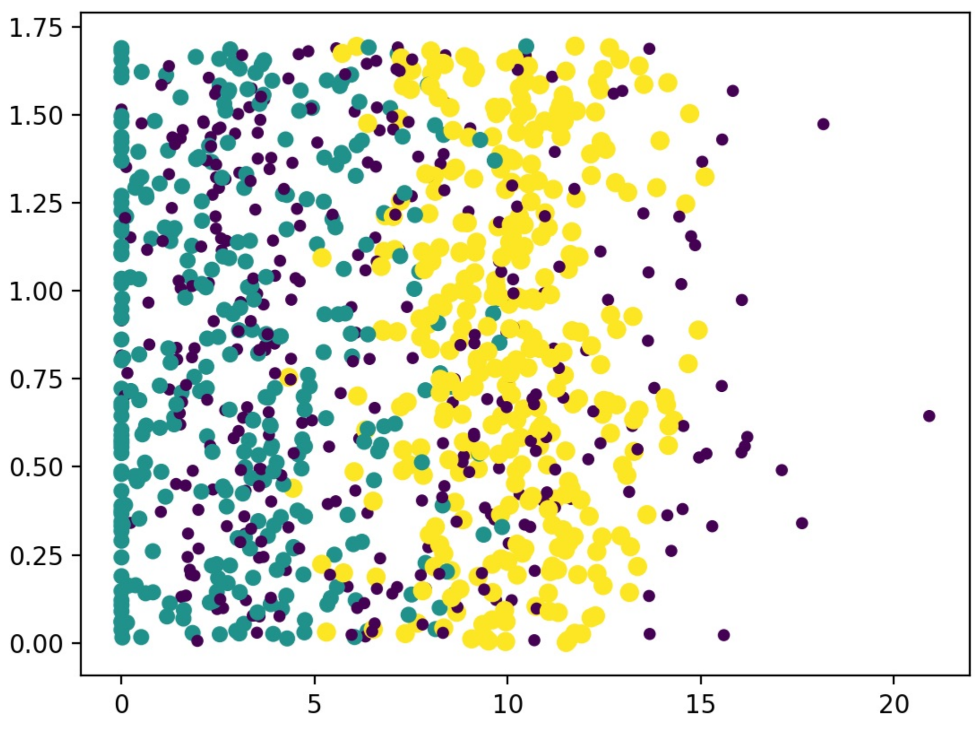
```
def file2matrix(filename):
    fr = open(filename)
    arrayOLines = fr.readlines()
    numberOfLines = len(arrayOLines) #读取文本中的行数

    #建立一个存放特征数据的numpy矩阵和label数据的列表
    returnMat = np.zeros((numberOfLines,3))
    classLabelVector = []
    index = 0
    for line in arrayOLines:
        line = line.strip() #删除每行前面的空白符
        listFromLine = line.split('\t') #把字符串按照指定分隔符进行切片，并把结果返回为字符串列表
        returnMat[index,:] = listFromLine[0:3]
        classLabelVector.append(listFromLine[-1])
        index+=1
    return returnMat,classLabelVector
```

分析数据

使用Matplotlib分析特征之间的关系

```
import matplotlib
import matplotlib.pyplot as plt
fig = plt.figure()
ax = fig.add_subplot(111)
ax.scatter(datingDataMat[:,1],datingDataMat[:,2],15.0*np.array(datingLabels),15.0*np.array(datingLabels))
plt.show()
```

结果图：



测试数据

首先对不同数量级的特征进行归一化，然后调用knn算法进行测试

```
#归一化特征到[0,1]
#根据公式: normValue = (value-min)/(max-min)
#用np.tile化为矩阵进行计算
def autoNorm(dataSet):
    #0表示选取每一列的最大最小值
    minValue = dataSet.min(0)
    maxValue = dataSet.max(0)
    ranges = maxValue-minValue
    normDataSet = np.zeros(np.shape(dataSet))
    m = dataSet.shape[0]
    normDataSet = dataSet - np.tile(minValue,(m,1))
    normDataSet = normDataSet/np.tile(ranges,(m,1))
    return normDataSet,ranges,minValue

def datingClassTest():
    testRatio = 0.10 #测试数据的比例
    datingDataMat,datingLabels = \
        file2matrix('/Users/jinyitao/Desktop/机器学习相关/《机器学习实
战》/machinelearninginaction/Ch02/datingTestSet.txt')
    normMat,ranges,minVals = autoNorm(datingDataMat)
    m = normMat.shape[0] #m为样本总数
    numTestVecs = int(m*testRatio)
    errorCount = 0.0
    for i in range(numTestVecs):
        classifierResult = knnClassify(normMat[i,:],normMat[numTestVecs:m,:],datingLabels[numTestVecs:m],3)
        print "the classifier came back with: %s,the real answer is: %s"%(classifierResult,datingLabels[i])
        if (classifierResult!=datingLabels[i]):
            errorCount+=1.0
    print "the total error rate is: %f"% (errorCount/float(numTestVecs))
```

运行结果：

the classifier came back with: smallDoses,the real answer is: smallDoses
the classifier came back with: didntLike,the real answer is: didntLike
...
the classifier came back with: largeDoses,the real answer is: didntLike
the total error rate is: 0.050000

实例2：使用k近邻算法处理CIFAR-10的图像分类问题

CIFAR-10数据集：

总共有10个类别的60000张32×32的RGB图像，其中每个类别的物体有6000张。其中50000张作为训练集，10000张作为测试集。

读取CIFAR-10数据集

```
def load_CIFAR_batch(filename):
  """ load single batch of cifar """
  with open(filename, 'rb') as f:
    datadict = pickle.load(f)
    X = datadict['data']
    Y = datadict['labels']
    # numpy的shape[n,d,row,col]
    # reshape为(10000,3,32,32) , transpose为(10000,32,32,3)
    # astype为float型
    X = X.reshape(10000, 3, 32, 32).transpose(0,2,3,1).astype("float")
    Y = np.array(Y)
    return X, Y

def load_CIFAR10(ROOT):
  """ load all of cifar """
```

```
  xs = []
  ys = []
  for b in range(1,6):
    f = os.path.join(ROOT, 'data_batch_%d' % (b,))
    X, Y = load_CIFAR_batch(f)
    #把5个batch整合起来
    xs.append(X)
    ys.append(Y)
  #最终的Xtr为（50000，32，32，3）
  Xtr = np.concatenate(xs)
  Ytr = np.concatenate(ys)
  del X, Y
  Xte, Yte = load_CIFAR_batch(os.path.join(ROOT, 'test_batch'))
  return Xtr, Ytr, Xte, Yte

# Load the raw CIFAR-10 data.
cifar10_dir = '/Users/jinyitao/Desktop/机器学习相关/机器学习测试数据集/cifar10_py'
X_train, y_train, X_test, y_test = load_CIFAR10(cifar10_dir)

# As a sanity check, we print out the size of the training and test data.
print 'Training data shape: ', X_train.shape
print 'Training labels shape: ', y_train.shape
print 'Test data shape: ', X_test.shape
print 'Test labels shape: ', y_test.shape

# 显示一部分图片
classes = ['plane', 'car', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship', 'truck']
num_classes = len(classes)
samples_per_class = 7
for y, className in enumerate(classes):
    idxs = np.flatnonzero(y_train == y) #返回符合要求的序号
    idxs = np.random.choice(idxs, samples_per_class, replace=False) #在符合要求的label中随机选择序号
    for i, idx in enumerate(idxs):
        plt_idx = i * num_classes + y + 1
        plt.subplot(samples_per_class, num_classes, plt_idx)
        plt.imshow(X_train[idx].astype('uint8'))
        plt.axis('off')
        if i == 0:
           plt.title(className)
plt.savefig("/Users/jinyitao/Desktop/机器学习相关/《机器学习实战》/myProject/1_knn/cifar10.jpg")
plt.show()
```

输出：
Training data shape: (50000, 32, 32, 3)
Training labels shape: (50000,)
Test data shape: (10000, 32, 32, 3)
Test labels shape: (10000,)

测试图片：

KNearestNeighbor类

成员函数主要由train()、predict()、compute_distances()组成
主要函数：

train()

训练数据集的读入

```
def train(self,X,y):
        self.X_train = X
        self.y_train = y
```

predict()

主要分两个步骤：
（1）计算欧式距离
（2）knn统计预测信息

```
def predict(self, X, k=1, num_loops=0):
        """
            Predict labels for test data using this classifier.

            Inputs:
            - X: A numpy array of shape (num_test, D) containing test data consisting
                 of num_test samples each of dimension D.
            - k: The number of nearest neighbors that vote for the predicted labels.
            - num_loops: Determines which implementation to use to compute distances
              between training points and testing points.

            Returns:
            - y: A numpy array of shape (num_test,) containing predicted labels for the
              test data, where y[i] is the predicted label for the test point X[i].
            """
        if num_loops == 0:
            dists = self.compute_distances_no_loops(X)
        elif num_loops == 1:
            dists = self.compute_distances_one_loop(X)
        elif num_loops == 2:
```

```python
            dists = self.compute_distances_two_loops(X)
        else:
            raise ValueError('Invalid value %d for num_loops' % num_loops)

        return self.predict_labels(dists, k=k)

    # 预测结果
    def predict_labels(self, dists, k=1):
        """
        Given a matrix of distances between test points and training points,
        predict a label for each test point.

        Inputs:
        - dists: A numpy array of shape (num_test, num_train) where dists[i, j]
          gives the distance betwen the ith test point and the jth training point.

        Returns:
        - y: A numpy array of shape (num_test,) containing predicted labels for the
          test data, where y[i] is the predicted label for the test point X[i].
        """
        num_test = dists.shape[0]
        y_pred = np.zeros(num_test)
        for i in xrange(num_test):
            # A list of length k storing the labels of the k nearest neighbors to
            # the ith test point.
            closest_y = []
            #########################################################################
            # TODO:                                                                 #
            # Use the distance matrix to find the k nearest neighbors of the ith    #
            # testing point, and use self.y_train to find the labels of these       #
            # neighbors. Store these labels in closest_y.                           #
            # Hint: Look up the function numpy.argsort.                             #
            #########################################################################
            # numpy.argsort 返回数组从小到大的索引值
            kids = np.argsort(dists[i])
            closest_y = self.y_train[kids[:k]]
            #########################################################################
            # TODO:                                                                 #
            # Now that you have found the labels of the k nearest neighbors, you    #
            # need to find the most common label in the list closest_y of labels.   #
            # Store this label in y_pred[i]. Break ties by choosing the smaller     #
            # label.                                                                #
            #########################################################################
            count = 0
            label = 0
            for j in closest_y:
                tmp = 0
                for kk in closest_y:
                    tmp += (kk == j)
                if tmp > count:
                    count = tmp
                    label = j
            y_pred[i] = label
            # y_pred[i] = np.argmax(np.bincount(closest_y))
            #########################################################################
            #                           END OF YOUR CODE                            #
            #########################################################################

        return y_pred
```

计算欧式距离

两次循环

建立一个array.shape=[num_test,num_train]
[i,j]位置的欧式距离通过两个for循环计算

```python
    def compute_distances_two_loops(self, X):
        """
        Compute the distance between each test point in X and each training point
        in self.X_train using a nested loop over both the training data and the
```

```
        test data.

        Inputs:
        - X: A numpy array of shape (num_test, D) containing test data.

        Returns:
        - dists: A numpy array of shape (num_test, num_train) where dists[i, j]
          is the Euclidean distance between the ith test point and the jth training
          point.
        """
        num_test = X.shape[0]
        num_train = self.X_train.shape[0]
        dists = np.zeros((num_test, num_train))
        for i in xrange(num_test):
            for j in xrange(num_train):
                #####################################################################
                # TODO:                                                             #
                # Compute the l2 distance between the ith test point and the jth    #
                # training point, and store the result in dists[i, j]. You should   #
                # not use a loop over dimension.                                    #
                #####################################################################
                dists[i, j] = np.sqrt(np.dot(X[i] - self.X_train[j], X[i] - self.X_train[j]))
                #######################################################################
                #                         END OF YOUR CODE                            #
                #######################################################################
        return dists
```

一次循环

本质和二次循环一样，用了axis=1指定行相加

```
def compute_distances_one_loop(self, X):
        """
        Compute the distance between each test point in X and each training point
        in self.X_train using a single loop over the test data.

        Input / Output: Same as compute_distances_two_loops
        """
        num_test = X.shape[0]
        num_train = self.X_train.shape[0]
        dists = np.zeros((num_test, num_train))
        for i in xrange(num_test):
            #######################################################################
            # TODO:                                                               #
            # Compute the l2 distance between the ith test point and all training #
            # points, and store the result in dists[i, :].                        #
            #######################################################################
            dists[i, :] = np.sqrt(np.sum(np.square(X[i] - self.X_train), axis=1))
            #######################################################################
            #                         END OF YOUR CODE                            #
            #######################################################################
        return dists
```

矩阵简化计算

我们记测试矩阵为P，大小为M×D，训练矩阵为C，大小为N×D
（1）记P_i是P的第i行，同理C_j是C的第j行:

$$P_i = [P_{i1}\ P_{i2}\ P_{i3}\ \cdots\ P_{iD}]$$

$$C_j = [C_{j1}\ C_{j2}\ C_{j3}\ \cdots\ C_{jD}]$$

（2）计算P_i和C_j之间的欧式距离：

$$d(P_i, C_j) = \sqrt{(P_{i1} - C_{j1})^2 + (P_{i2} - C_{j2})^2 + (P_{i3} - C_{j3})^2 + \cdots + (P_{iD} - C_{jD})^2}$$

$$= \sqrt{(P_{i1}^2 + P_{i2}^2 + P_{i3}^2 + \cdots + P_{iD}^2) + (C_{j1}^2 + C_{j2}^2 + C_{j3}^2 + \cdots + C_{jD}^2) - 2 * (P_{i1}C_{j1} + P_{i2}C_{j2}) + P_{i3}C_{j3} + \cdots + P_{iD}C_{jD}}$$

$$= \sqrt{||P_i||^2 + ||C_j||^2 - 2P_iC_j'}$$

(3）推广到矩阵的每一行：

$$res(i) = \sqrt{(||P_i||^2 \ \ ||P_i||^2 \ \ \cdots \ ||P_i||^2) + (||C_1||^2 \ \ ||C_2||^2 \ \ \cdots \ ||C_j||^2) - 2P_i(C_1' \ \ C_2' \ \ \cdots \ C_D')}$$

$$= \sqrt{(||P_i||^2 \ \ ||P_i||^2 \ \ \cdots \ ||P_i||^2) + (||C_1||^2 \ \ ||C_2||^2 \ \ \cdots \ ||C_j||^2) - 2P_iC'}$$

(4）推广到矩阵计算：

$$res = \sqrt{\begin{pmatrix} ||P_1||^2 & ||P_1||^2 & \cdots & ||P_1||^2 \\ ||P_2||^2 & ||P_2||^2 & \cdots & ||P_2||^2 \\ \cdots & \cdots & \cdots & \cdots \\ ||P_M||^2 & ||P_M||^2 & \cdots & ||P_M||^2 \end{pmatrix} + \begin{pmatrix} ||C_1||^2 & ||C_2||^2 & \cdots & ||C_N||^2 \\ ||C_1||^2 & ||C_2||^2 & \cdots & ||C_N||^2 \\ \cdots & \cdots & \cdots & \cdots \\ ||C_1||^2 & ||C_2||^2 & \cdots & ||C_N||^2 \end{pmatrix} - 2PC'}$$

$$= \sqrt{\begin{pmatrix} ||P_1||^2 \\ ||P_2||^2 \\ \cdots \\ ||P_M||^2 \end{pmatrix}_{M\times 1} * \begin{pmatrix} 1 & 1 & \cdots & 1 \end{pmatrix}_{1\times N} + \begin{pmatrix} 1 \\ 1 \\ \cdots \\ 1 \end{pmatrix}_{M\times 1} * \begin{pmatrix} ||C_1||^2 & ||C_2||^2 & \cdots & ||C_N||^2 \end{pmatrix}_{1\times N} - 2PC'}$$

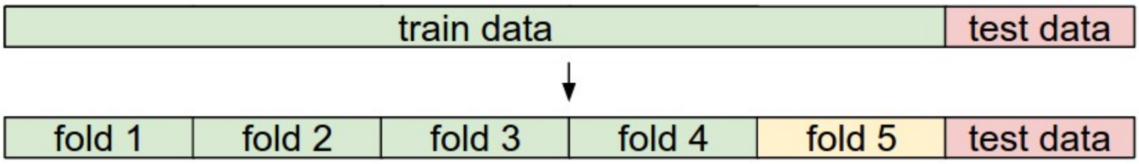
python代码实现：

```
def compute_distances_no_loops(self, X):
        """
        Compute the distance between each test point in X and each training point
        in self.X_train using no explicit loops.

        Input / Output: Same as compute_distances_two_loops
        """
        num_test = X.shape[0]
        num_train = self.X_train.shape[0]
        dists = np.zeros((num_test, num_train))
        #########################################################################
        # TODO:                                                                 #
        # Compute the l2 distance between all test points and all training      #
        # points without using any explicit loops, and store the result in      #
        # dists.                                                                #
        #                                                                       #
        # You should implement this function using only basic array operations; #
        # in particular you should not use functions from scipy.                #
        #                                                                       #
        # HINT: Try to formulate the l2 distance using matrix multiplication    #
        #       and two broadcast sums.                                         #
        #########################################################################
        dists = np.sqrt(self.getNormMatrix(X, num_train).T + self.getNormMatrix(self.X_train, num_test)- 2 *
np.dot(X,self.X_train.T))
        #########################################################################
        #                         END OF YOUR CODE                              #
        #########################################################################
        return dists

    def getNormMatrix(self, x, lines_num):
        """
        Get a lines_num x size(x, 1) matrix
        """
        return np.ones((lines_num, 1)) * np.sum(np.square(x), axis=1)
```

测试结果：
Got 282 / 1000 correct => accuracy: 0.282000
Two loop version took 44.513359 seconds
One loop version took 104.611335 seconds
No loop version took 0.576688 seconds
化简后的矩阵计算比循环计算速度快了很多

采用交叉验证法选择最佳的k值

交叉验证实际上是将数据的训练集进行拆分，分成多个组，构成多个训练和测试集， 来筛选较好的超参数。



如图所示，可以分为5组数据，（分别将 fold 1,2..5 作为验证集， 将剩余的数据作为训练集，训练得到超参数）

测试结果：

可以得到当k=8时，训练集的正确率最高，但也仅30%+，也说明k近邻算法处理分类问题的局限性。