第 4 章 用 K 近邻算法进行分类和回归

K 近邻模型是一种用于回归任务和分类任务的简单模型。算法中的“邻居”代表度量空间中的训练实例。度量空间是定义了集合中所有成员之间距离的特征空间。K 近邻的工作原理是：存在一个样本数据集合，也称作训练样本集，并且样本集中每个数据都存在标签，即我们知道样本集中每一数据与所属分类的对应关系。输入没有标签的新数据后，将新数据的每个特征与样本集中数据对应的特征进行比较，然后算法提取样本集中特征最相似数据（最近邻）的分类标签。一般来说，我们只选择样本数据集中前 k 个最相似的数据，这就是 K-近邻算法中 K 的出处，通常 k 是不大于 20 的整数。最后，选择 K 个最相似数据中出现次数最多的分类，作为新数据的分类。

使用 k-近邻算法分类爱情片和动作片。有人曾经统计过很多电影的打斗镜头和接吻镜头，图4.1显示了 6 部电影的打斗和接吻镜头数。假如有一部未看过的电影，如何确定它是爱情片还是动作片呢？我们可以使用 kNN 来解决这个问题。

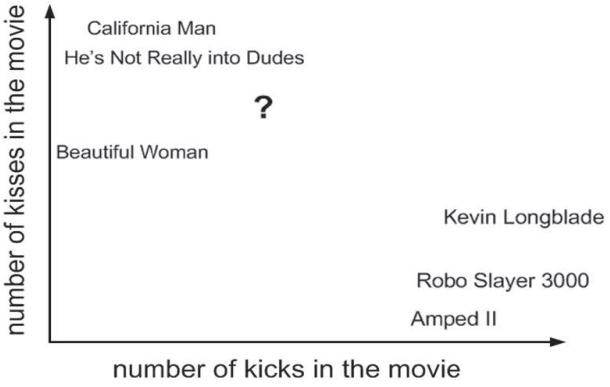


图 4.1: 使用打斗和接吻镜头数分类电影

首先我们需要知道这个未知电影存在多少个打斗镜头和接吻镜头，具体数字参见表4.1。

表 4.1: 每部电影的打斗镜头数、接吻镜头数以及电影评估类型

电影名称	打斗镜头	接吻镜头	电影类型
California Man	3	104	爱情片
He's Not Really into Dudes	2	100	爱情片
Beautiful Woman	1	81	爱情片
Kevin Longblade	101	10	动作片
Robo Slayer 3000	99	5	动作片
Amped II	98	2	动作片
?	18	90	未知

即使不知道未知电影属于哪种类型，我们也可以通过某种方法计算出来。首先计算未知电影与样本集中其他电影的距离，如表4.1所示。此处暂时不要关心如何计算得到这些距离值，使用 Python 实现电影分类应用时，会提供具体的计算方法。

二维空间中的欧式距离计算公式为：

$$d(p,q) = d(q,p) = \sqrt{(q_1 - p_1)^2 + (q_2 - p_2)^2} \tag{4.1}$$

表 4.2: 已知电影与未知电影的距离

电影名称	与未知电影的距离
California Man	20.5
He's Not Really into Dudes	18.7
Beautiful Woman	19.2
Kevin Longblade	115.3
Robo Slayer 3000	117.4
Amped II	118.9

现在我们得到了样本集中所有电影与未知电影的距离，按照距离递增排序，可以找到 k 个距离最近的电影。假定 k=3，则三个最靠近的电影依次是 He' s Not Really into Dudes、Beautiful Woman 和 California Man。k-近邻算法按照距离最近的三部电影的类型，决定未知电影的类型，而这三部电影全是爱情片，因此我们判定未知电影是爱情片。

k-近邻算法的一般流程

1. 收集数据：可以使用任何方法。
2. 准备数据：距离计算所需要的数值，最好是结构化的数据格式。
3. 分析数据：可以使用任何方法。
4. 训练算法：此步骤不适用于 k-近邻算法。
5. 测试算法：计算错误率。
6. 使用算法：首先需要输入样本数据和结构化的输出结果，然后运行 k-近邻算法判定输入数据分别属于哪个分类，最后应用对计算出的分类执行后续的处理。

注 K 近邻算法是一种惰性学习模型（lazy classifier）。惰性学习模型也称为基于实例的学习模型。通常对训练数据集集不做处理或进行少量的处理。和线性回归模型不同，K 近邻在训练阶段不会估计模型的参数。

4.1 K 近邻模型的实现

示例：电影的分类。该任务通过统计电影镜头中打斗镜头和亲吻镜头的个数判断电影的类型。

4.1.1 自己动手实现

```
import numpy as np
import matplotlib.pyplot as plt

X_train = np.array([
    [3, 104],
    [2, 100],
    [1, 81],
    [101, 10],
    [99, 5],
    [98, 2],
])
y_train = ['love', 'love', 'love', 'kick', 'kick', 'kick']
```

```
plt.figure()
plt.title('Kissing and kicks by movie')
plt.xlabel('number of kicks')
plt.ylabel('number of kiss')

for i, x in enumerate(X_train):
    plt.scatter(x[0], x[1], c='k', marker='x' if y_train[i] == 'love' else 'D')
plt.grid(True)
plt.show()
```

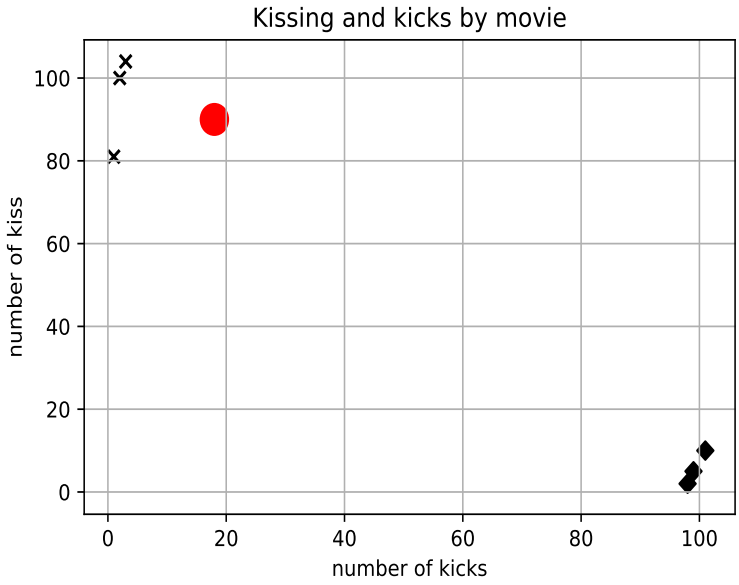


图 4.2: 使用打斗和接吻镜头数分类电影

```
mport numpy as np
import operator

"""
Parameters:
    无
Returns:
    group - 数据集
    labels - 分类标签
"""
# 函数说明:创建数据集
def createDataSet():
    #六组二维特征
    group = np.array([[3,104],[2,100],[1,81],[101,10],[99,5],[98,2]])
    #六组特征的标签
    labels = ['爱情片','爱情片','爱情片','动作片','动作片','动作片']
    return group, labels

"""
Parameters:
```

```
    inX - 用于分类的数据(测试集)
    dataSet - 用于训练的数据(训练集)
    labes - 分类标签
    k - kNN算法参数,选择距离最小的k个点
Returns:
    sortedClassCount[0][0] - 分类结果
"""
# 函数说明:kNN算法,分类器
def classify0(inX, dataSet, labels, k):
    #numpy函数shape[0]返回dataSet的行数

    # 请自己编写代码
    return sortedClassCount[0][0]

if __name__ == '__main__':
    #创建数据集
    group, labels = createDataSet()
    #测试集
    test = [101,20]
    #kNN分类
    test_class = classify0(test, group, labels, 3)
    #打印分类结果
    print(test_class)
```

4.1.2 sklearn 实现

在 sklearn 库中有实现KNN 的分类器, 可以直接用于分类:

Examples 来自 sklearn

```
>>> X = [[0], [1], [2], [3]]
>>> y = [0, 0, 1, 1]
>>> from sklearn.neighbors import KNeighborsClassifier
>>> neigh = KNeighborsClassifier(n_neighbors=3)
>>> neigh.fit(X, y)
KNeighborsClassifier(...)
>>> print(neigh.predict([[1.1]]))
[0]
>>> print(neigh.predict_proba([[0.9]]))
[[0.666... 0.333...]]
```

动手实现电影分类:

```
from sklearn.preprocessing import LabelBinarizer
from sklearn.neighbors import KNeighborsClassifier

lb = LabelBinarizer()
y_train_binarized = lb.fit_transform(y_train)

K = 3
```

```
clf = KNeighborsClassifier(n_neighbors=K)
clf.fit(X_train, y_train_binarized.reshape(-1))
prediction_binarized = clf.predict(np.array([18, 90]).reshape(1, -1))[0]
predicted_label = lb.inverse_transform(prediction_binarized)
```

我们的标签是字符串，所以首先使用 LabelBinarizer 将其转换为整数。

4.1.3 模型的评估

使用 K 近邻分类器对测试数据集进行预测，同时对分离器的性能进行评估。

表 4.3: 每部电影的打斗镜头数、接吻镜头数以及电影评估类型

电影名称	打斗镜头	接吻镜头	电影类型
A	45	40	爱情片
B	5	90	爱情片
C	60	81	爱情片
D	80	20	动作片

```
X_test = np.array([
    [45, 40],
    [5, 90],
    [60, 81],
    [80, 20]
])
y_test = ['love', 'love', 'love', 'kick']
y_test_binarized = lb.transform(y_test)
print('Binarized labels: %s' % y_test_binarized.T[0])

predictions_binarized = clf.predict(X_test)
print('Binarized predictions: %s' % predictions_binarized)
print('Predicted labels: %s' % lb.inverse_transform(predictions_binarized))
```

当然，也可以利用 sklearn 实现模型的评估：

```
from sklearn.metrics import accuracy_score
print('Accuracy: %s' % accuracy_score(y_test_binarized, predictions_binarized))
```

错误率和准确率来衡量分类器任务的成功程度的。错误率指的是在所有测试样例中错分的样例比例。实际上，这样的度量错误掩盖了样例如何被分错的事实。在机器学习中，有一个普遍适用的称为**混淆矩阵**（confusion matrix）的工具，它可以帮助人们更好地了解分类中的错误。有这样一个关于在房子周围可能发现的动物类型的预测，这个预测的三类问题的混淆矩阵如表4.4所示。

表 4.4: 一个三类问题的混淆矩阵

混淆矩阵	狗	猫	鼠
狗	24	2	5
猫	2	27	0
鼠	4	2	30

利用混淆矩阵就可以更好地理解分类中的错误了。如果矩阵中的非对角元素均为 0，就会得到一个完美的分类器。

接下来，我们考虑另外一个混淆矩阵，这次的矩阵只针对一个简单的二类问题。在表 7-3 中，给出了该混淆矩阵。在这个二类问题中，如果将一个正例判为正例，那么就可以认为产生了一个真正例（True Positive，TP，也称真阳）；如果对一个反例正确地判为反例，则认为产生了一个真反例（True Negative，TN，也称真阴）。相应地，另外两种情况则分别称为伪反例（False Negative，FN，也称假阴）和伪正例（False Positive，FP，也称假阳）。这 4 种情况如表 7-3 所示。

表 4.5: 一个二类问题的混淆矩阵，其中的输出采用了不同的类别标签

混淆矩阵	+1	-1
+1	真正例（TP）	伪反例（FN）
-1	伪正例（FP）	真反例（TN）

在分类中，当某个类别的重要性高于其他类别时，我们就可以利用上述定义来定义出多个比错误率更好的新指标。第一个指标是正确率（Precision），它等于 TP/(TP+FP)，给出的是预测为正例的样本中的真正正例的比例。第二个指标是召回率（Recall），它等于 TP/(TP+FN)，给出的是预测为正例的真实正例占所有真实正例的比例。在召回率很大的分类器中，真正判错的正例的数目并不多。

```
from sklearn.metrics import precision_score
print('Precision: %s' % precision_score(y_test_binarized, predictions_binarized))

from sklearn.metrics import recall_score
print('Recall: %s' % recall_score(y_test_binarized, predictions_binarized))

from sklearn.metrics import f1_score
print('F1 score: %s' % f1_score(y_test_binarized, predictions_binarized))
```

我们可以很容易构造一个高正确率或高召回率的分类器，但是很难同时保证两者成立。如果将任何样本都判为正例，那么召回率达到百分之百而此时正确率很低。构建一个同时使正确率和召回率最大的分类器是具有挑战性的。

4.2 KNN 模型回归

现在我们用 KNN 模型进行一个回归任务，利用一个人的身高和性别来预测其体重。

训练数据：

测试数据

我们将对 KNeighborsRegressor 类进行实例化和拟合，并使用它来预测体重。在这个数据集中，性别已经编码为二元值特征。

注 性别的取值为 [0,1], 而表示身高的特征取值范围为 [155 191]。

```
import numpy as np
import matplotlib.pyplot as plt
```

表 4.6: 训练数据

身高（cm）	性别	体重 kg
158	男性	64
170	男性	66
183	男性	84
191	男性	80
155	女性	49
163	女性	59
180	女性	67
158	女性	54
178	女性	77

表 4.7: 测试数据

身高（cm）	性别	体重 kg
168	男性	65
170	男性	61
160	女性	52
169	女性	67

```
from sklearn.neighbors import KNeighborsRegressor

X_train = np.array([
    [158, 64, 1],
    [170, 86, 1],
    [183, 84, 1],
    [191, 80, 1],
    [155, 49, 0],
    [163, 59, 0],
    [180, 67, 0],
    [158, 54, 0],
    [170, 67, 0]
])
y_train = [7, 12, 29, 18, 11, 16, 29, 22, 36]

X_test = np.array([
    [160, 66, 1],
    [196, 87, 1],
    [168, 68, 0],
    [177, 74, 0]
])
y_test = [9, 13, 26, 21]

K = 1
clf = KNeighborsRegressor(n_neighbors=K)
clf.fit(X_train, y_train)
predictions = clf.predict(np.array(X_test))
predictions
```

4.2.1 衡量模型

除了之前引入的模型系数衡量模型的性能外，还有其他的指标可以用于衡量模型的性能。

- **MAE**: 平均绝对误差
- **MSE**：均方误差

MAE 是预测结果误差绝对值的均值：

$$MAE = \frac{1}{n}\sum_{i=0}^{n-1}|y_i - \hat{y}_i| \tag{4.2}$$

MSE 是更常见的指标，是预测结果误差平方的均值：

$$MAE = \frac{1}{n}\sum_{i=0}^{n-1}(y_i - \hat{y}_i)^2 \tag{4.3}$$

```
from sklearn.metrics import r2_score
print(r2_score(y_test, predictions))
```

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.neighbors import KNeighborsRegressor

X_train = np.array([
    [158, 1],
    [170, 1],
    [183, 1],
    [191, 1],
    [155, 0],
    [163, 0],
    [180, 0],
    [158, 0],
    [170, 0]
])
y_train = [64, 86, 84, 80, 49, 59, 67, 54, 67]

X_test = np.array([
    [160, 1],
    [196, 1],
    [168, 0],
    [177, 0]
])
y_test = [66, 87, 68, 74]

K = 3
clf = KNeighborsRegressor(n_neighbors=K)
clf.fit(X_train, y_train)
predictions = clf.predict(np.array(X_test))
print('Predicted weights: %s' % predictions)
```

```
print('Actual weights: %s' % y_test)
```

4.3 特征归一化

当特征具有相同的取值范围时，许多学习算法将会表现更好。在处理这种不同取值范围的特征值时，我们通常采用的方法是将数值归一化，如将取值范围处理为 0 到 1 或者-1 到 1 之间。下面的公式可以将任意取值范围的特征值转化为 0 到 1 区间内的值:

$$newValue = (oldValue - min)/(max - min) \tag{4.4}$$

其中 min 和 max 分别是数据集中的最小特征值和最大特征值。

我们增加一个新函数 autoNorm()，该函数可以自动将数字特征值转化为 0 到 1 的区间。

```
"""
Parameters:
    dataSet - 特征矩阵
Returns:
    normDataSet - 归一化后的特征矩阵
    ranges - 数据范围
    minVals - 数据最小值
"""
# 函数说明:对数据进行归一化
def autoNorm(dataSet):
    #获得数据的最小值
    minVals = dataSet.min(0)
    maxVals = dataSet.max(0)
    #最大值和最小值的范围
    ranges = maxVals - minVals
    #shape(dataSet)返回dataSet的矩阵行列数
    normDataSet = np.zeros(np.shape(dataSet))
    #返回dataSet的行数
    m = dataSet.shape[0]
    #原始值减去最小值
    normDataSet = dataSet - np.tile(minVals, (m, 1))
    #除以最大和最小值的差,得到归一化数据
    normDataSet = normDataSet / np.tile(ranges, (m, 1))
    #返回归一化数据结果,数据范围,最小值
    return normDataSet, ranges, minVals
```

在函数 autoNorm() 中，我们将每列的最小值放在变量 minVals 中，将最大值放在变量 maxVals 中，其中 dataSet.min(0) 中的参数 0 使得函数可以从列中选取最小值，而不是选取当前行的最小值。然后，函数计算可能的取值范围，并创建新的返回矩阵。正如公式4.4，

为了归一化特征值，我们必须使用当前值减去最小值，然后除以取值范围。需要注意的是，特征值矩阵有 1000×3 个值，而 minVals 和 range 的值都为 1×3。为了解决这个问题，我们使用 NumPy 库中 tile() 函数将变量内容复制成输入矩阵同样大小的矩阵，注意这是具体特征值相除，而对于某些数值处理软件包，/ 可能意味着矩阵除法，但在 NumPy 库中，矩阵除法需要使用函数 linalg.solve(matA,matB)。

标准归一化：

将所有实例特征值减去均值来将其居中，然后将每个实例的特征值除以特征的标准差对其缩放。均值为 0，方差为 1 的数据称为**标准化数据**。

$$newValue = (oldValue - \mu)/\sigma \tag{4.5}$$

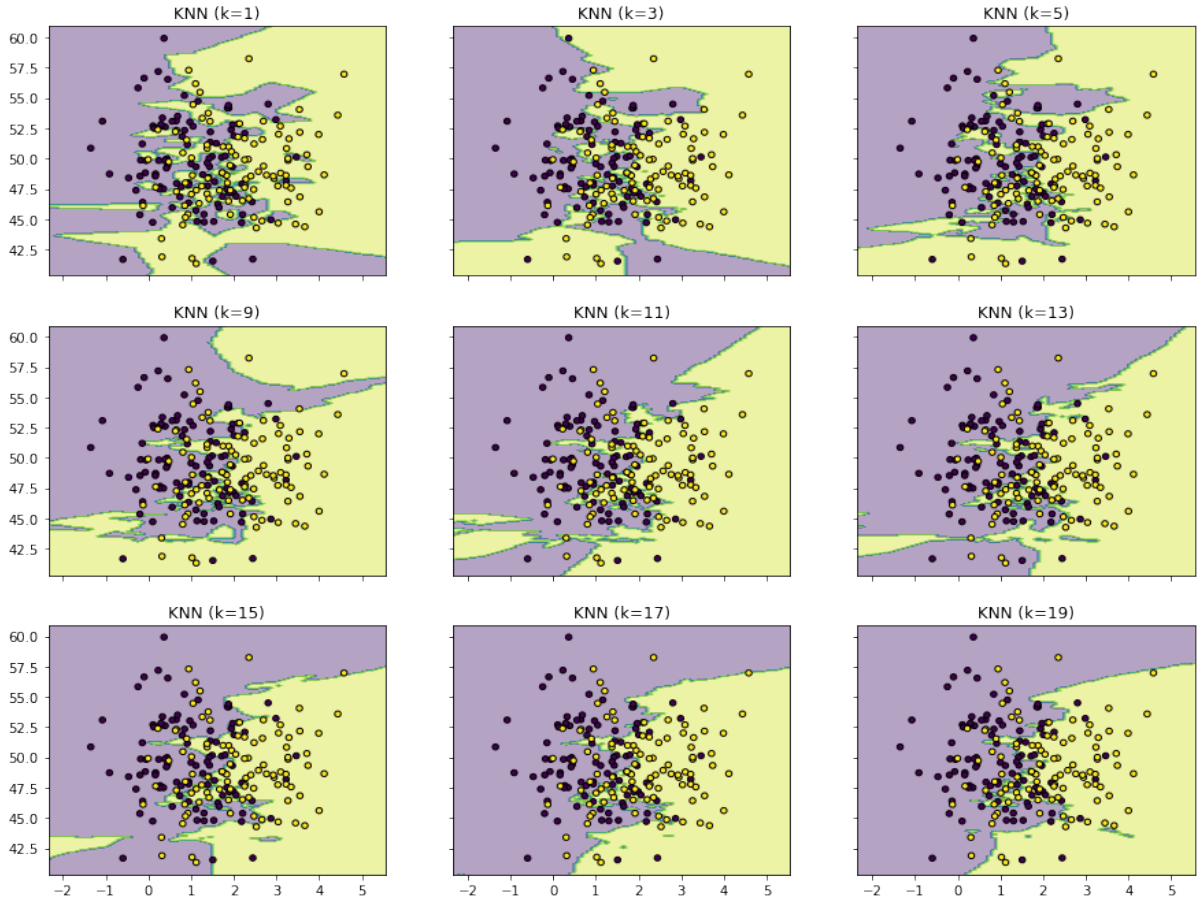
sklearn 类库中的 StandardScaler 类是一个用于特征缩放的转换器。

```
from sklearn.preprocessing import StandardScaler
ss = StandardScaler()
x_train_scaled = ss.fit_transform(x_train)

x_test_scaled = ss.transform(x_test)
```

4.3.1 KNN 的决策边界

```
import matplotlib.pyplot as plt
import numpy as np
from itertools import product
from sklearn.neighbors import KNeighborsClassifier

# 生成一些随机样本
n_points = 100
X1 = np.random.multivariate_normal([1,50], [[1,0],[0,10]], n_points)
X2 = np.random.multivariate_normal([2,50], [[1,0],[0,10]], n_points)
X = np.concatenate([X1,X2])
y = np.array([0]*n_points + [1]*n_points)
print (X.shape, y.shape)

# KNN模型的训练过程
clfs = []
neighbors = [1,3,5,9,11,13,15,17,19]
for i in range(len(neighbors)):
    clfs.append(KNeighborsClassifier(n_neighbors=neighbors[i]).fit(X,y))

# 可视化结果
x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.1),
                     np.arange(y_min, y_max, 0.1))

f, axarr = plt.subplots(3,3, sharex='col', sharey='row', figsize=(15, 12))
for idx, clf, tt in zip(product([0, 1, 2], [0, 1, 2]),
                        clfs, ['KNN (k=%d)'%k for k in neighbors]):
    Z = clf.predict(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)

    axarr[idx[0], idx[1]].contourf(xx, yy, Z, alpha=0.4)
    axarr[idx[0], idx[1]].scatter(X[:, 0], X[:, 1], c=y,
```

```
                                s=20, edgecolor='k')
    axarr[idx[0], idx[1]].set_title(tt)

plt.show()
```



第 4 章 练习

1. 使用 k-近邻算法改进约会网站的配对效果

   我的朋友海伦一直使用在线约会网站寻找适合自己的约会对象。尽管约会网站会推荐不同的人选，但她并不是喜欢每一个人。经过一番总结，她发现曾交往过三种类型的人：

   - 不喜欢的人
   - 魅力一般的人

   尽管发现了上述规律，但海伦依然无法将约会网站推荐的匹配对象归入恰当的分类。她觉得可以在周一到周五约会那些魅力一般的人，而周末则更喜欢与那些极具魅力的人为伴。海伦希望我们的分类软件可以更好地帮助她将匹配对象划分到确切的分类中。此外海伦还收集了一些约会网站未曾记录的数据信息，她认为这些数据更有助于匹配对象的归类。

   (a). 收集数据：提供文本文件。

   (b). 准备数据：使用 Python 解析文本文件。

   (c). 分析数据：使用 Matplotlib 画二维扩散图。

(d). 训练算法：此步骤不适用于 k-近邻算法。

(e). 测试算法：使用海伦提供的部分数据作为测试样本。测试样本和非测试样本的区别在于：测试样本是已经完成分类的数据，如果预测分类与实际类别不同，则标记为一个错误。

(f). 使用算法：产生简单的命令行程序，然后海伦可以输入一些特征数据以判断对方是否为自己喜欢的类型。

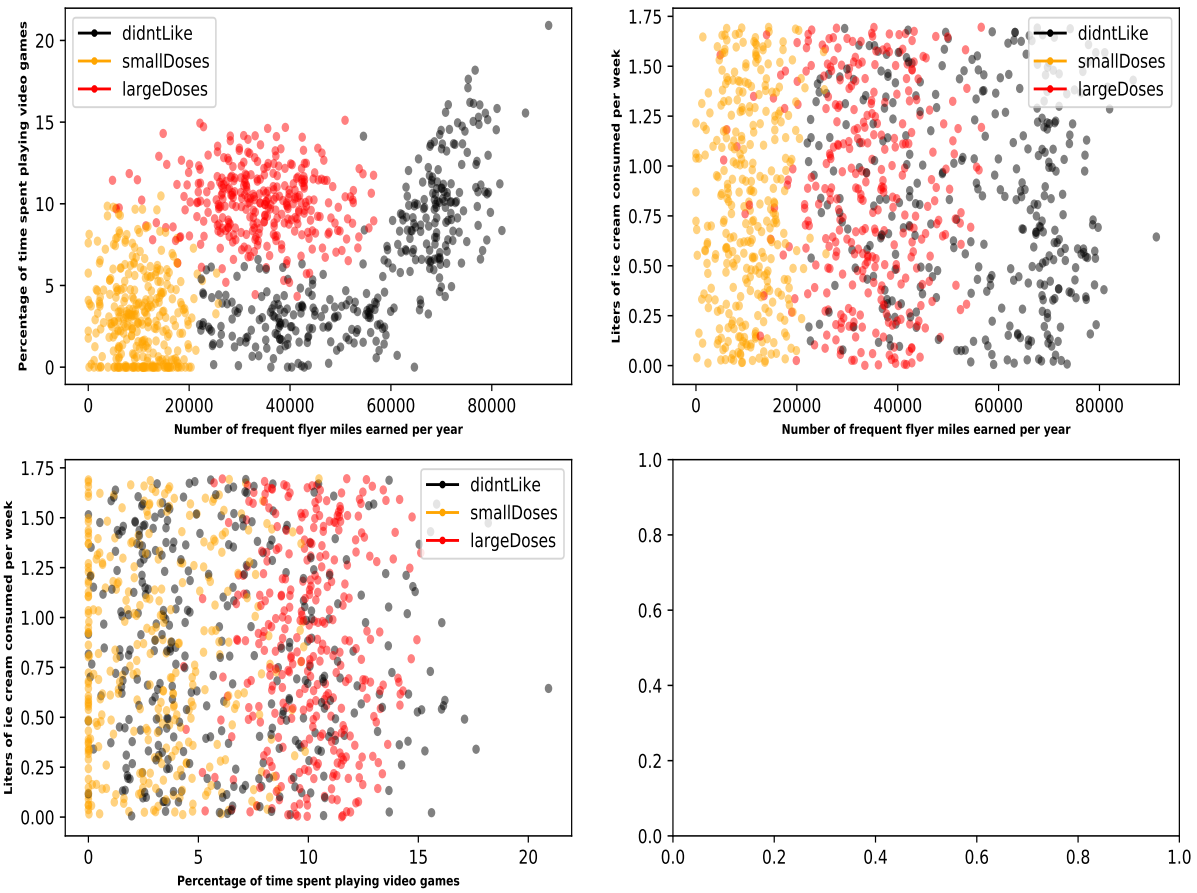
准备数据：从文本文件中解析数据

海伦收集约会数据已经有了一段时间，她把这些数据存放在文本文件datingTestSet.txt 中，每个样本数据占据一行，总共有 1000 行。海伦的样本主要包含以下 3 种特征：

- 每年获得的飞行常客里程数
- 玩视频游戏所耗时间百分比
- 每周消费的冰淇淋公升数

```
import numpy as np

"""
Parameters:
    filename - 文件名
Returns:
    returnMat - 特征矩阵
    classLabelVector - 分类Label向量
"""
# 函数说明:打开并解析文件，对数据进行分类：1代表不喜欢,2代表魅力一般,3代表极具魅力
def file2matrix(filename):
    #打开文件
    fr = open(filename)
    #读取文件所有内容
    arrayOLines = fr.readlines()
    #得到文件行数
    numberOfLines = len(arrayOLines)
    #返回的NumPy矩阵,解析完成的数据:numberOfLines行,3列
    returnMat = np.zeros((numberOfLines,3))
    #返回的分类标签向量
    classLabelVector = []
    #行的索引值
    index = 0
    for line in arrayOLines:
        #s.strip(rm)，当rm空时,默认删除空白符(包括'\n','\r','\t',' ')
        line = line.strip()
        #使用s.split(str="",num=string,cout(str))将字符串根据'\t'分隔符进行切片。
        listFromLine = line.split('\t')
        #将数据前三列提取出来,存放到returnMat的NumPy矩阵中,也就是特征矩阵
        returnMat[index,:] = listFromLine[0:3]
        #根据文本中标记的喜欢的程度进行分类,1代表不喜欢,2代表魅力一般,3代表极具魅力
        if listFromLine[-1] == 'didntLike':
            classLabelVector.append(1)
```

```
        elif listFromLine[-1] == 'smallDoses':
            classLabelVector.append(2)
        elif listFromLine[-1] == 'largeDoses':
            classLabelVector.append(3)
        index += 1
    return returnMat, classLabelVector
```



准备数据：归一化数值

```
"""
Parameters:
    dataSet - 特征矩阵
Returns:
    normDataSet - 归一化后的特征矩阵
    ranges - 数据范围
    minVals - 数据最小值
"""
# 函数说明:对数据进行归一化
def autoNorm(dataSet):
    #获得数据的最小值
    minVals = dataSet.min(0)
    maxVals = dataSet.max(0)
    #最大值和最小值的范围
    ranges = maxVals - minVals
```

```
    #shape(dataSet)返回dataSet的矩阵行列数
    normDataSet = np.zeros(np.shape(dataSet))
    #返回dataSet的行数
    m = dataSet.shape[0]
    #原始值减去最小值
    normDataSet = dataSet - np.tile(minVals, (m, 1))
    #除以最大和最小值的差,得到归一化数据
    normDataSet = normDataSet / np.tile(ranges, (m, 1))
    #返回归一化数据结果,数据范围,最小值
    return normDataSet, ranges, minVals
```

测试算法：作为完整程序验证分类器

```
# 函数说明:分类器测试函数
def datingClassTest():

    # 请编写代码

    # 打开的文件名

    #将返回的特征矩阵和分类向量分别存储到datingDataMat和datingLabels中

    #取所有数据的百分之十

    #数据归一化,返回归一化后的矩阵,数据范围,数据最小值

    #获得normMat的行数

    #百分之十的测试数据的个数

    #分类错误计数

    print("错误率:%f%%" %(errorCount/float(numTestVecs)*100))
```

2. KNN 函数来实现鸢尾花分类

```
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
import numpy as np

iris = datasets.load_iris()
X = iris.data
y = iris.target
print (X, y)

## 画一下数据的分布（前两个维度）

code here
```

```
## # 把数据分成训练数据和测试数据

code here

## # 构建KNN模型， K值为3、 并做训练

code here

#计算准确率
code here
```

3. 从零开始自己写一个 KNN 算法

```
from sklearn import datasets
from collections import Counter # 为了做投票
from sklearn.model_selection import train_test_split
import numpy as np

# 导入iris数据
iris = datasets.load_iris()
X = iris.data
y = iris.target
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=2003)

def euc_dis(instance1, instance2):
    """
    计算两个样本instance1和instance2之间的欧式距离
    instance1: 第一个样本， array型
    instance2: 第二个样本， array型
    """
    # TODO
    dist = code here
    return dist

def knn_classify(X, y, testInstance, k):
    """
    给定一个测试数据testInstance, 通过KNN算法来预测它的标签。
    X: 训练数据的特征
    y: 训练数据的标签
    testInstance: 测试数据，这里假定一个测试数据 array型
    k: 选择多少个neighbors?
    """
    # TODO 返回testInstance的预测标签 = {0,1,2}
    code here

    return

# 预测结果

code here
```