

摘要

本文将从k-近邻算法的思想开始讲起，使用python3一步一步编写代码进行实战训练。并且，我也提供了相应的数据集，对代码进行了详细的注释。除此之外，本文也对sklea算法的方法进行了讲解。实战实例：电影类别分类、约会网站配对效果判定、手写数字识别。



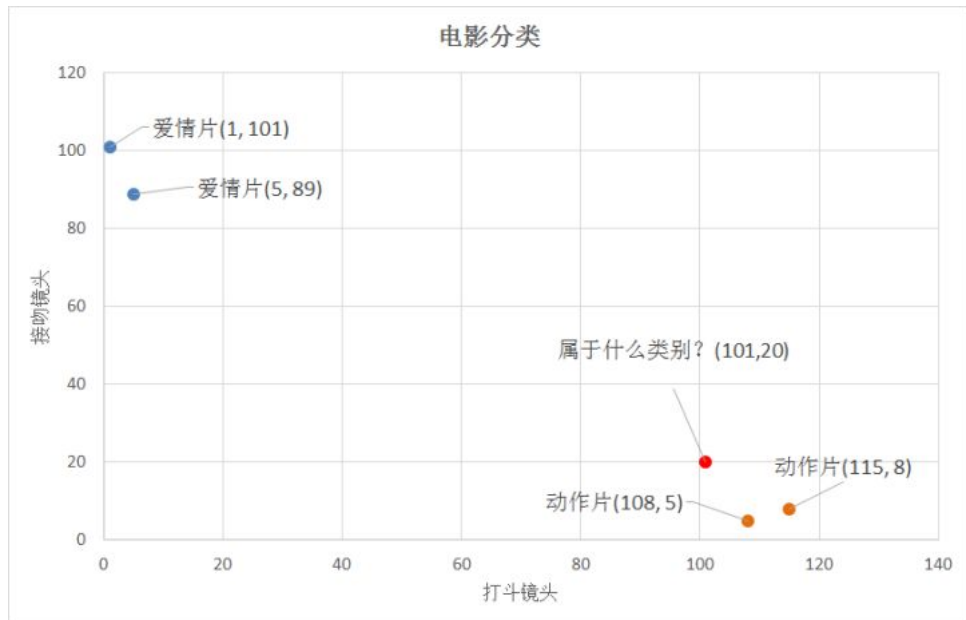
一、简单k-近邻算法

本文将从k-近邻算法的思想开始讲起，使用python3一步一步编写代码进行实战训练。并且，我也提供了相应的数据集，对代码进行了详细的注释。除也对sklearn实现k-近邻算法的方法进行了讲解。实战实例：电影类别分类、约会网站配对效果判定、手写数字识别。

本文出现的所有代码和数据集，均可在我的github上下载，欢迎Follow、Star：[Github代码地址](#)

1、k-近邻法简介

k近邻法(k-nearest neighbor, k-NN)是1967年由Cover T和Hart P提出的一种基本分类与回归方法。它的工作原理是：存在一个样本数据集合，也称作训练集，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个数据与所属分类的对应关系。输入没有标签的新数据后，将新的数据的每个特征与样本集的特征进行比较，然后算法提取样本最相似数据(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，k一般取1，3，5，7，9，11，13，15，17，19，21，23，25，27，29，31，33，35，37，39，41，43，45，47，49，51，53，55，57，59，61，63，65，67，69，71，73，75，77，79，81，83，85，87，89，91，93，95，97，99，101，103，105，107，109，111，113，115，117，119，121，123，125，127，129，131，133，135，137，139，141，143，145，147，149，151，153，155，157，159，161，163，165，167，169，171，173，175，177，179，181，183，185，187，189，191，193，195，197，199，201，203，205，207，209，211，213，215，217，219，221，223，225，227，229，231，233，235，237，239，241，243，245，247，249，251，253，255，257，259，261，263，265，267，269，271，273，275，277，279，281，283，285，287，289，291，293，295，297，299，301，303，305，307，309，311，313，315，317，319，321，323，325，327，329，331，333，335，337，339，341，343，345，347，349，351，353，355，357，359，361，363，365，367，369，371，373，375，377，379，381，383，385，387，389，391，393，395，397，399，401，403，405，407，409，411，413，415，417，419，421，423，425，427，429，431，433，435，437，439，441，443，445，447，449，451，453，455，457，459，461，463，465，467，469，471，473，475，477，479，481，483，485，487，489，491，493，495，497，499，501，503，505，507，509，511，513，515，517，519，521，523，525，527，529，531，533，535，537，539，541，543，545，547，549，551，553，555，557，559，561，563，565，567，569，571，573，575，577，579，581，583，585，587，589，591，593，595，597，599，601，603，605，607，609，611，613，615，617，619，621，623，625，627，629，631，633，635，637，639，641，643，645，647，649，651，653，655，657，659，661，663，665，667，669，671，673，675，677，679，681，683，685，687，689，691，693，695，697，699，701，703，705，707，709，711，713，715，717，719，721，723，725，727，729，731，733，735，737，739，741，743，745，747，749，751，753，755，757，759，761，763，765，767，769，771，773，775，777，779，781，783，785，787，789，791，793，795，797，799，801，803，805，807，809，811，813，815，817，819，821，823，825，827，829，831，833，835，837，839，841，843，845，847，849，851，853，855，857，859，861，863，865，867，869，871，873，875，877，879，881，883，885，887，889，891，893，895，897，899，901，903，905，907，909，911，913，915，917，919，921，923，925，927，929，931，933，935，937，939，941，943，945，947，949，951，953，955，957，959，961，963，965，967，969，971，973，975，977，979，981，983，985，987，989，991，993，995，997，999，1001，1003，1005，1007，1009，1011，1013，1015，1017，1019，1021，1023，1025，1027，1029，1031，1033，1035，1037，1039，1041，1043，1045，1047，1049，1051，1053，1055，1057，1059，1061，1063，1065，1067，1069，1071，1073，1075，1077，1079，1081，1083，1085，1087，1089，1091，1093，1095，1097，1099，1101，1103，1105，1107，1109，1111，1113，1115，1117，1119，1121，1123，1125，1127，1129，1131，1133，1135，1137，1139，1141，1143，1145，1147，1149，1151，1153，1155，1157，1159，1161，1163，1165，1167，1169，1171，1173，1175，1177，1179，1181，1183，1185，1187，1189，1191，1193，1195，1197，1199，1201，1203，1205，1207，1209，1211，1213，1215，1217，1219，1221，1223，1225，1227，1229，1231，1233，1235，1237，1239，1241，1243，1245，1247，1249，1251，1253，1255，1257，1259，1261，1263，1265，1267，1269，1271，1273，1275，1277，1279，1281，1283，1285，1287，1289，1291，1293，1295，1297，1299，1301，1303，1305，1307，1309，1311，1313，1315，1317，1319，1321，1323，1325，1327，1329，1331，1333，1335，1337，1339，1341，1343，1345，1347，1349，1351，1353，1355，1357，1359，1361，1363，1365，1367，1369，1371，1373，1375，1377，1379，1381，1383，1385，1387，1389，1391，1393，1395，1397，1399，1401，1403，1405，1407，1409，1411，1413，1415，1417，1419，1421，1423，1425，1427，1429，1431，1433，1435，1437，1439，1441，1443，1445，1447，1449，1451，1453，1455，1457，1459，1461，1463，1465，1467，1469，1471，1473，1475，1477，1479，1481，1483，1485，1487，1489，1491，1493，1495，1497，1499，1501，1503，1505，1507，1509，1511，1513，1515，1517，1519，1521，1523，1525，1527，1529，1531，1533，1535，1537，1539，1541，1543，1545，1547，1549，1551，1553，1555，1557，1559，1561，1563，1565，1567，1569，1571，1573，1575，1577，1579，1581，1583，1585，1587，1589，1591，1593，1595，1597，1599，1601，1603，1605，1607，1609，1611，1613，1615，1617，1619，1621，1623，1625，1627，1629，1631，1633，1635，1637，1639，1641，1643，1645，1647，1649，1651，1653，1655，1657，1659，1661，1663，1665，1667，1669，1671，1673，1675，1677，1679，1681，1683，1685，1687，1689，1691，1693，1695，1697，1699，1701，1703，1705，1707，1709，1711，1713，1715，1717，1719，1721，1723，1725，1727，1729，1731，1733，1735，1737，1739，1741，1743，1745，1747，1749，1751，1753，1755，1757，1759，1761，1763，1765，1767，1769，1771，1773，1775，1777，1779，1781，1783，1785，1787，1789，1791，1793，1795，1797，1799，1801，1803，1805，1807，1809，1811，1813，1815，1817，1819，1821，1823，1825，1827，1829，1831，1833，1835，1837，1839，1841，1843，1845，1847，1849，1851，1853，1855，1857，1859，1861，1863，1865，1867，1869，1871，1873，1875，1877，1879，1881，1883，1885，1887，1889，1891，1893，1895，1897，1899，1901，1903，1905，1907，1909，1911，1913，1915，1917，1919，1921，1923，1925，1927，1929，1931，1933，1935，1937，1939，1941，1943，1945，1947，1949，1951，1953，1955，1957，1959，1961，1963，1965，1967，1969，1971，1973，1975，1977，1979，1981，1983，1985，1987，1989，1991，1993，1995，1997，1999，2001，2003，2005，2007，2009，2011，2013，2015，2017，2019，2021，2023，2025，2027，2029，2031，2033，2035，2037，2039，2041，2043，2045，2047，2049，2051，2053，2055，2057，2059，2061，2063，2065，2067，2069，2071，2073，2075，2077，2079，2081，2083，2085，2087，2089，2091，2093，2095，2097，2099，2101，2103，2105，2107，2109，2111，2113，2115，2117，2119，2121，2123，2125，2127，2129，2131，2133，2135，2137，2139，2141，2143，2145，2147，2149，2151，2153，2155，2157，2159，2161，2163，2165，2167，2169，2171，2173，2175，2177，2179，2181，2183，2185，2187，2189，2191，2193，2195，2197，2199，2201，2203，2205，2207，2209，2211，2213，2215，2217，2219，2221，2223，2225，2227，2229，2231，2233，2235，2237，2239，2241，2243，2245，2247，2249，2251，2253，2255，2257，2259，2261，2263，2265，2267，2269，2271，2273，2275，2277，2279，2281，2283，2285，2287，2289，2291，2293，2295，2297，2299，2301，2303，2305，2307，2309，2311，2313，2315，2317，2319，2321，2323，2325，2327，2329，2331，2333，2335，2337，2339，2341，2343，2345，2347，2349，2351，2353，2355，2357，2359，2361，2363，2365，2367，2369，2371，2373，2375，2377，2379，2381，2383，2385，2387，2389，2391，2393，2395，2397，2399，2401，2403，2405，2407，2409，2411，2413，2415，2417，2419，2421，2423，2425，2427，2429，2431，2433，2435，2437，2439，2441，2443，2445，2447，2449，2451，2453，2455，2457，2459，2461，2463，2465，2467，2469，2471，2473，2475，2477，2479，2481，2483，2485，2487，2489，2491，2493，2495，2497，2499，2501，2503，2505，2507，2509，2511，2513，2515，2517，2519，2521，2523，2525，2527，2529，2531，2533，2535，2537，2539，2541，2543，2545，2547，2549，2551，2553，2555，2557，2559，2561，2563，2565，2567，2569，2571，2573，2575，2577，2579，2581，2583，2585，2587，2589，2591，2593，2595，2597，2599，2601，2603，2605，2607，2609，2611，2613，2615，2617，2619，2621，2623，2625，2627，2629，2631，2633，2635，2637，2639，2641，2643，2645，2647，2649，2651，2653，2655，2657，2659，2661，2663，2665，2667，2669，2671，2673，2675，2677，2679，2681，2683，2685，2687，2689，2691，2693，2695，2697，2699，2701，2703，2705，2707，2709，2711，2713，2715，2717，2719，2721，2723，2725，2727，2729，2731，2733，2735，2737，2739，2741，2743，2745，2747，2749，2751，2753，2755，2757，2759，2761，2763，2765，2767，2769，2771，2773，2775，2777，2779，2781，2783，2785，2787，2789，2791，2793，2795，2797，2799，2801，2803，2805，2807，2809，2811，2813，2815，2817，2819，2821，2823，2825，2827，2829，2831，2833，2835，2837，2839，2841，2843，2845，2847，2849，2851，2853，2855，2857，2859，2861，2863，2865，2867，2869，2871，2873，2875，2877，2879，2881，2883，2885，2887，2889，2891，2893，2895，2897，2899，2901，2903，2905，2907，2909，2911，2913，2915，2917，2919，2921，2923，2925，2927，2929，2931，2933，2935，2937，2939，2941，2943，2945，2947，2949，2951，2953，2955，2957，2959，2961，2963，2965，2967，2969，2971，2973，2975，2977，2979，2981，2983，2985，2987，2989，2991，2993，2995，2997，2999，3001，3003，3005，3007，3009，3011，3013，3015，3017，3019，3021，3023，3025，3027，3029，3031，3033，3035，3037，3039，3041，3043，3045，3047，3049，3051，3053，3055，3057，3059，3061，3063，3065，3067，3069，3071，3073，3075，3077，3079，3081，3083，3085，3087，3089，3091，3093，3095，3097，3099，3101，3103，3105，3107，3109，3111，3113，3115，3117，3119，3121，3123，3125，3127，3129，3131，3133，3135，3137，3139，3141，3143，3145，3147，3149，3151，3153，3155，3157，3159，3161，3163，3165，3167，3169，3171，3173，3175，3177，3179，3181，3183，3185，3187，3189，3191，3193，3195，3197，3199，3201，3203，3205，3207，3209，3211，3213，3215，3217，3219，3221，3223，3225，3227，3229，3231，3233，3235，3237，3239，3241，3243，3245，3247，3249，3251，3253，3255，3257，3259，3261，3263，3265，3267，3269，3271，3273，3275，3277，3279，3281，3283，3285，3287，3289，3291，3293，3295，3297，3299，3301，3303，3305，3307，3309，3311，3313，3315，3317，3319，3321，3323，3325，3327，3329，3331，3333，3335，3337，3339，3341，3343，3345，3347，3349，3351，3353，3355，3357，3359，3361，3363，3365，3367，3369，3371，3373，3375，3377，3379，3381，3383，3385，3387，3389，3391，3393，3395，3397，3399，3401，3403，3405，3407，3409，3411，3413，3415，3417，3419，3421，3423，3425，3427，3429，3431，3433，3435，3437，3439，3441，3443，3445，3447，3449，3451，3453，3455，3457，3459，3461，3463，3465，3467，3469，3471，3473，3475，3477，3479，3481，3483，3485，3487，3489，3491，3493，3495，3497，3499，3501，3503，3505，3507，3509，3511，3513，3515，3517，3519，3521，3523，3525，3527，3529，3531，3533，3535，3537，3539，3541，3543，3545，3547，3549，3551，3553，3555，3557，3559，3561，3563，3565，3567，3569，3571，3573，3575，3577，3579，3581，3583，3585，3587，3589，3591，3593，3595，3597，3599，3601，3603，3605，3607，3609，3611，3613，3615，3617，3619，3621，3623，3625，3627，3629，3631，3633，3635，3637，3639，3641，3643，3645，3647，3649，3651，3653，3655，3657，3659，3661，3663，3665，3667，3669，3671，3673，3675，3677，3679，3681，3683，3685，3687，3689，3691，3693，3695，3697，3699，3701，3703，3705，3707，3709，3711，3713，3715，3717，3719，3721，3723，3725，3727，3729，3731，3733，3735，3737，3739，3741，3743，3745，3747，3749，3751，3753，3755，3757，3759，3761，3763，3765，3767，3769，3771，3773，3775，3777，3779，3781，3783，3785，3787，3789，3791，3793，3795，3797，3799，3801，3803，3805，3807，3809，3811，3813，3815，3817，3819，3821，3823，3825，3827，3829，3831，3833，3835，3837，3839，3841，3843，3845，3847，3849，3851，3853，3855，3857，3859，3861，3863，3865，3867，3869，3871，3873，3875，3877，3879，3881，3883，3885，3887，3889，3891，3893，3895，3897，3899，3901，3903，3905，3907，3909，3911，3913，3915，3917，3919，3921，3923，3925，3927，3929，3931，3933，3935，3937，3939，3941，3943，3945，3947，3949，3951，3953，3955，3957，3959，3961，3963，3965，3967，3969，3971，3973，3975，3977，3979，3981，3983，3985，3987，3989，3991，3993，3995，3997，3999，4001，4003，4005，4007，4009，4011，4013，4015，4017，4019，4021，4023，4025，4027，4029，4031，4033，4035，4037，4039，4041，4043，4045，4047，4049，4051，4053，4055，4057，4059，4061，4063，4065，4067，4069，4071，4073，4075，4077，4079，4081，4083，4085，4087，4089，4091，4093，4095，4097，4099，4101，4103，4105，4107，4109，4111，4113，4115，4117，4119，4121，4123，4125，4127，4129，4131，4133，4135，4137，4139，4141，4143，4145，4147，4149，4151，4153，4155，4157，4159，4161，4163，4165，4167，4169，4171，4173，4175，4177，4179，4181，4183，4185，4187，4189，4191，4193，4195，4197，4199，4201，4203，4205，4207，4209，4211，4213，4215，4217，4219，4221，4223，4225，4227，4229，4231，4233，4235，4237，4239，4241，4243，4245，4247，4249，4251，4253，4255，4257，4259，4261，4263，4265，4267，4269，4271，4273，4275，4277，4279，4281，4283，4285，4287，4289，4291，4293，4295，4297，4299，4301，4303，4305，4307，4309，4311，4313，4315，4317，4319，4321，4323，4325，4327，4329，4331，4333，4335，4337，4339，4341，4343，4345，4347，4349，4351，4353，4355，4357，4359，4361，4363，4365，4367，4369，4371，4373，4375，4377，4379，4381，4383，4385，4387，4389，4391，4393，4395，4397，4399，4401，4403，4405，4407，4409，4411，4413，4415，4417，4419，4421，4423，4425，4427，4429，4431，4433，4435，4437，4439，4441，4443，4445，4447，4449，4451，4453，4455，4457，4459，4461，4463，4465，4467，4469，4471，4473，4475，4477，4479，4481，4483，4485，4487，4489，4491，4493，4495，4497，4499，4501，4503，4505，4507，4509，4511，4513，4515，4517，4519，4521，4523，4525，4527，4529，4531，4533，4535，4537，4539，4541，4543，4545，4547，4549，4551，4553，4555，4557，4559，4561，4563，4565，4567，4569，4571，4573，4575，4577，4579，4581，4583，4585，4587，4589，4591，4593，4595，4597，4599，4601，4603，4605，4607，4609，4611，4613，4615，4617，4619，4621，4623，4625，4627，4629，4631，4633，4635，4637，4639，4641，4643，4645，4647，4649，4651，4653，4655，4657，4659，4661，4663，4665，4667，4669，4671，4673，4675，4677，4679，4681，4683，4685，4687，4689，4691，4693，4695，4697，4699，4701，4703，4705，4707，4709，4711，4713，4715，4717，4719，4721，4723，4725，4727，4729，4731，4733，4735，4737，4739，4741，4743，4745，4747，4749，4751，4753，4755，4757，4759，4761，4763，4765，4767，4769，4771，4773，4775，4777，4779，4781，4783，4785，4787，4789，4791，4793，4795，4797，4799，4801，4803，4805，4807，4809，4811，4813，4815，4817，4819，4821，4823，4825，4827，4829，4831，4833，4835，4837，4839，4841，4843，4845，4847，4849，4851，4853，4855，4857，4859，4861，4863，4865，4867，4869，4871，4873，4875，4877，4879，4881，4883，4885，4887，4889，4891，4893，4895，4897，4899，4901，4903，4905，4907，4909，4911，4913，4915，4917，4919，4921，4923，4925，4927，4929，4931，4933，4935，4937，4939，4941，4



我们可以从散点图大致推断，这个红色圆点标记的电影可能属于动作片，因为距离已知的那两个动作片的圆点更近。k-近邻算法用什么方法进行判断？距离度量。这个电影分类的例子有2个特征，也就是在2维实数向量空间，可以使用我们高中学过的两点距离公式计算距离，如图1.2所示。

$$|AB| = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

通过计算，我们可以得到如下结果：

- (101,20)->动作片(108,5)的距离约为16.55
- (101,20)->动作片(115,8)的距离约为18.44
- (101,20)->爱情片(5,89)的距离约为118.22
- (101,20)->爱情片(1,101)的距离约为128.69

通过计算可知，红色圆点标记的电影到动作片 (108,5)的距离最近，为16.55。如果算法直接根据这个结果，判断该红色圆点标记的电影为动作片，这叫近邻算法，而非k-近邻算法。那么k-近邻算法是什么呢？k-近邻算法步骤如下：

1. 计算已知类别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个点所在类别的出现频率；
5. 返回前k个点所出现频率最高的类别作为当前点的预测分类。

比如，现在我这个k值取3，那么在电影例子中，按距离依次排序的三个点分别是动作片(108,5)、动作片(115,8)、爱情片(5,89)。在这三个点中，动作片三分之二，爱情片出现的频率为三分之一，所以该红色圆点标记的电影为动作片。这个判别过程就是k-近邻算法。

3、Python3代码实现

我们已经知道了k-近邻算法的原理，那么接下来就是使用Python3实现该算法，依然以电影分类为例。

(1)准备数据集

对于表1.1中的数据，我们可以使用numpy直接创建，代码如下：

```
1 # -*- coding: UTF-8 -*-
2 import numpy as np
3
4 """
5 函数说明:创建数据集
6
7 Parameters:
8     无
9 Returns:
10     group - 数据集
11     labels - 分类标签
12 Modify:
13     2017-07-13
14 """
15 def createDataSet():
16     #四组二维特征
17     group = np.array([[1,101],[5,89],[108,5],[115,8]])
18     #四组特征的标签
19     labels = ['爱情片','爱情片','动作片','动作片']
20     return group, labels
21 if __name__ == '__main__':
```

```

22 #创建数据集
23 group, labels = createDataSet()
24 #打印数据集
25 print(group)
26 print(labels)

```

运行结果，如图1.3所示：

```

65
66 if __name__ == '__main__':
67     #创建数据集
68     group, labels = createDataSet()
69     #打印数据集
70     print(group)
71     print(labels)

```

```

[[ 1 101]
 [ 5 89]
 [108 5]
 [115 8]]
['爱情片', '爱情片', '动作片', '动作片']
[Finished in 0.5s]

```

图1.3 运行结果

(2)k-近邻算法

根据两点距离公式，计算距离，选择距离最小的前k个点，并返回分类结果。

```

1 # -*- coding: UTF-8 -*-
2 import numpy as np
3 import operator
4
5 """
6 函数说明: 创建数据集
7
8 Parameters:
9     无
10 Returns:
11     group - 数据集
12     labels - 分类标签
13 Modify:
14     2017-07-13
15 """
16 def createDataSet():
17     #四组二维特征
18     group = np.array([[1,101],[5,89],[108,5],[115,8]])
19     #四组特征的标签
20     labels = ['爱情片', '爱情片', '动作片', '动作片']
21     return group, labels
22
23 """
24 函数说明: kNN算法, 分类器
25
26 Parameters:
27     inX - 用于分类的数据(测试集)
28     dataSet - 用于训练的数据(训练集)
29     labels - 分类标签
30     k - kNN算法参数, 选择距离最小的k个点
31 Returns:
32     sortedClassCount[0][0] - 分类结果
33
34 Modify:
35     2017-07-13
36 """
37 def classify0(inX, dataSet, labels, k):
38     #numpy函数shape[0]返回dataSet的行数
39     dataSetSize = dataSet.shape[0]
40     #在列向量方向上重复inX共1次(横向), 行向量方向上重复inX共dataSetSize次(纵向)
41     diffMat = np.tile(inX, (dataSetSize, 1)) - dataSet
42     #二维特征相减后平方
43     sqDiffMat = diffMat**2
44     #sum()所有元素相加, sum(0)列相加, sum(1)行相加
45     sqDistances = sqDiffMat.sum(axis=1)
46     #开方, 计算出距离
47     distances = sqDistances**0.5
48     #返回distances中元素从小到大排序后的索引值
49     sortedDistIndices = distances.argsort()
50     #定一个记录类别次数的字典
51     classCount = {}
52     for i in range(k):
53         #取出前k个元素的类别
54         voteIlabel = labels[sortedDistIndices[i]]
55         #dict.get(key, default=None), 字典的get()方法, 返回指定键的值, 如果值不在字典中返回默认值。
56         #计算类别次数
57         classCount[voteIlabel] = classCount.get(voteIlabel, 0) + 1
58     #python3中用items()替换python2中的iteritems()
59     #key=operator.itemgetter(1)根据字典的值进行排序
60     #key=operator.itemgetter(0)根据字典的键进行排序
61     #reverse降序排序字典
62     sortedClassCount = sorted(classCount.items(), key=operator.itemgetter(1), reverse=True)
63     #返回次数最多的类别, 即所要分类的类别
64     return sortedClassCount[0][0]
65

```

```

66 if __name__ == '__main__':
67     #创建数据集
68     group, labels = createDataSet()
69     #测试集
70     test = [101,20]
71     #kNN分类
72     test_class = classify0(test, group, labels, 3)
73     #打印分类结果
74     print(test_class)

```

运行结果，如图1.4所示：



图1.4 运行结果

可以看到，分类结果根据我们的“经验”，是正确的，尽管这种分类比较耗时，用时1.4s。

到这里，也许有人早已经发现，电影例子中的特征是2维的，这样的距离度量可以用两点距离公式计算，但是如果是更高维的呢？对，没错。我们可以用欧几里德度量（也称欧几里德度量），如图1.5所示。我们高中所学的两点距离公式就是欧氏距离在二维空间上的公式，也就是欧氏距离的n的值为2的情况。

$$\begin{aligned}
 d(\mathbf{p}, \mathbf{q}) &= d(\mathbf{q}, \mathbf{p}) = \sqrt{(q_1 - p_1)^2 + (q_2 - p_2)^2 + \cdots + (q_n - p_n)^2} \\
 &= \sqrt{\sum_{i=1}^n (q_i - p_i)^2}.
 \end{aligned}$$

图1.5 欧氏距离公式

看到这里，有人可能会问：“分类器何种情况下会出错？”或者“答案是否总是正确的？”答案是否定的，分类器并不会得到百分百正确的结果，我们可以检测分类器的正确率。此外分类器的性能也会受到多种因素的影响，如分类器设置和数据集等。不同的算法在不同数据集上的表现可能完全不同。为了测试分类器的性能，我们可以使用已知答案的数据，当然答案不能告诉分类器，检验分类器给出的结果是否符合预期结果。通过大量的测试数据，我们可以得到分类器的性能。分类器的性能可以通过给出错误结果的次数除以测试执行的总数。错误率是常用的评估方法，主要用于评估分类器在某个数据集上的执行效果。完美分类器的错误率为0，最差分类器的错误率为1.0。同时，我们也不难发现，k-近邻算法没有进行数据的训练，直接使用未知的数据与已知的数据进行比较，得到结果。因此，可以说k-近邻算法不具有训练过程。

二、k-近邻算法实战之约会网站配对效果判定

上一小结学习了简单的k-近邻算法的实现方法，但是这并不是完整的k-近邻算法流程，k-近邻算法的一般流程：

1. 收集数据：可以使用爬虫进行数据的收集，也可以使用第三方提供的免费或收费的数据。一般来讲，数据放在txt文本文件中，按照一定的格式进行解析及处理。
2. 准备数据：使用Python解析、预处理数据。
3. 分析数据：可以使用很多方法对数据进行分析，例如使用Matplotlib将数据可视化。
4. 测试算法：计算错误率。
5. 使用算法：错误率在可接受范围内，就可以运行k-近邻算法进行分类。

已经了解了k-近邻算法的一般流程，下面开始进入实战内容。

1、实战背景

海伦女士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候选，但她并不是喜欢每一个人。经过一番总结，她发现自己可以进行如下分类：

1. 不喜欢的人
2. 魅力一般的人

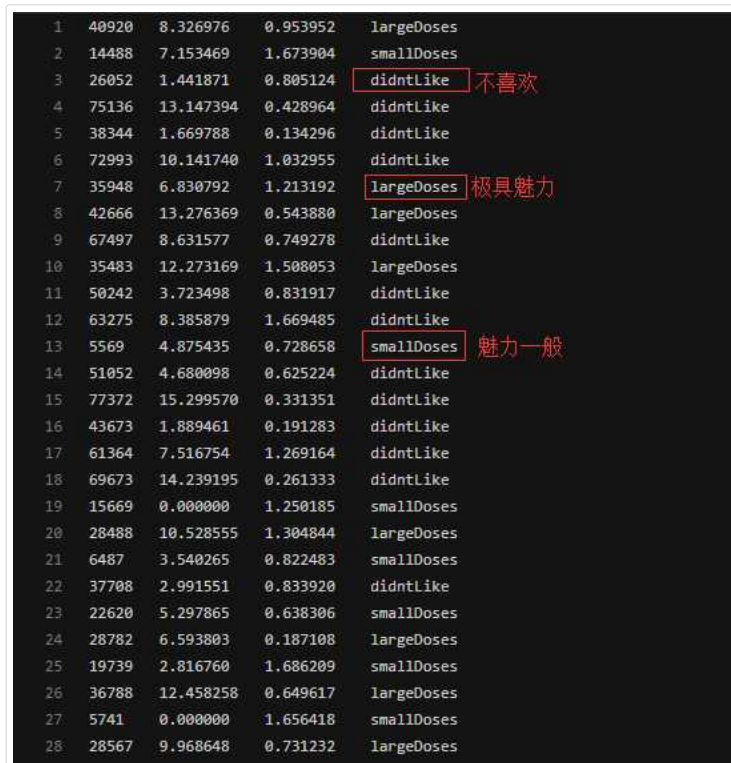
3. 极具魅力的人

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。datingTestSet.txt数据集下载

海伦收集的样本数据主要包含以下3种特征：

1. 每年获得的飞行常客里程数
2. 玩视频游戏所消耗时间百分比
3. 每周消费的冰淇淋公升数

这里不得不吐槽一句，海伦是个小吃货啊，冰淇淋公斤数都影响自己择偶标准。打开txt文本文件，数据格式如图2.1所示。



1	40920	8.326976	0.953952	largeDoses
2	14488	7.153469	1.673904	smallDoses
3	26052	1.441871	0.805124	didn'tLike
4	75136	13.147394	0.428964	didn'tLike
5	38344	1.669788	0.134296	didn'tLike
6	72993	10.141740	1.032955	didn'tLike
7	35948	6.830792	1.213192	largeDoses
8	42666	13.276369	0.543880	largeDoses
9	67497	8.631577	0.749278	didn'tLike
10	35483	12.273169	1.508053	largeDoses
11	50242	3.723498	0.831917	didn'tLike
12	63275	8.385879	1.669485	didn'tLike
13	5569	4.875435	0.728658	smallDoses
14	51052	4.680098	0.625224	didn'tLike
15	77372	15.299570	0.331351	didn'tLike
16	43673	1.889461	0.191283	didn'tLike
17	61364	7.516754	1.269164	didn'tLike
18	69673	14.239195	0.261333	didn'tLike
19	15669	0.000000	1.250185	smallDoses
20	28488	10.528555	1.304844	largeDoses
21	6487	3.540265	0.822483	smallDoses
22	37708	2.991551	0.833920	didn'tLike
23	22620	5.297865	0.638306	smallDoses
24	28782	6.593803	0.187108	largeDoses
25	19739	2.816760	1.686209	smallDoses
26	36788	12.458258	0.649617	largeDoses
27	5741	0.000000	1.656418	smallDoses
28	28567	9.968648	0.731232	largeDoses

图2.1 datingTestSet.txt格式

2、准备数据：数据解析

在将上述特征数据输入到分类器前，必须将待处理的数据的格式改变为分类器可以接收的格式。分类器接收的数据是什么格式的？从上小结已经知道：两部分，即特征矩阵和对应的分类标签向量。在kNN_test02.py文件中创建名为file2matrix的函数，以此来处理输入格式问题。将datingTestSet.txt放到与k相同目录下，编写代码如下：

```

1  #-*- coding: UTF-8 -*-
2  import numpy as np
3  """
4  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
5
6  Parameters:
7      filename - 文件名
8  Returns:
9      returnMat - 特征矩阵
10     classLabelVector - 分类Label向量
11
12  Modify:
13      2017-03-24
14  """
15  def file2matrix(filename):
16      #打开文件
17      fr = open(filename)
18      #读取文件所有内容
19      array0Lines = fr.readlines()
20      #得到文件行数
21      numberOfLines = len(array0Lines)
22      #返回的NumPy矩阵, 解析完成的数据: numberOfLines行, 3列
23      returnMat = np.zeros((numberOfLines, 3))
24      #返回的分类标签向量
25      classLabelVector = []
26      #行的索引值
27      index = 0
28      for line in array0Lines:
29          #s.strip(rm), 当rm为空时, 默认删除空白符(包括'\n', '\r', '\t', ' ')
30          line = line.strip()
31          #使用s.split(str="", num=string.count(str))将字符串根据'分隔符'进行切片。
32          listFromLine = line.split('\t')
33          #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
34          returnMat[index, :] = listFromLine[0:3]

```



```

35 #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
36 if listFromLine[-1] == 'didntLike':
37     classLabelVector.append(1)
38 elif listFromLine[-1] == 'smallDoses':
39     classLabelVector.append(2)
40 elif listFromLine[-1] == 'largeDoses':
41     classLabelVector.append(3)
42 index += 1
43 return returnMat, classLabelVector
44
45 """
46 函数说明:main函数
47
48 Parameters:
49 无
50 Returns:
51 无
52
53 Modify:
54 2017-03-24
55 """
56 if __name__ == '__main__':
57     #打开的文件名
58     filename = "datingTestSet.txt"
59     #打开并处理数据
60     datingDataMat, datingLabels = file2matrix(filename)
61     print(datingDataMat)
62     print(datingLabels)

```

运行上述代码，得到的数据解析结果如图2.2所示。



可以看到，我们已经顺利导入数据，并对数据进行解析，格式化为分类器需要的数据格式。接着我们需要了解数据的真正含义。可以通过友好、直观的方式观察数据。

3、分析数据：数据可视化

在kNN_test02.py文件中编写名为showdatas的函数，用来将数据可视化。编写代码如下：

```

1 # -*- coding: UTF-8 -*-
2 from matplotlib.font_manager import FontProperties
3 import matplotlib.lines as mlines
4 import matplotlib.pyplot as plt
5 import numpy as np
6
7 """
8 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
9
10 Parameters:
11     filename - 文件名
12 Returns:
13     returnMat - 特征矩阵
14     classLabelVector - 分类Label向量
15
16 Modify:
17     2017-03-24
18 """
19 def file2matrix(filename):
20     #打开文件
21     fr = open(filename)
22     #读取文件所有内容
23     array0Lines = fr.readlines()
24     #得到文件行数
25     numberOfLines = len(array0Lines)
26     #返回的NumPy矩阵,解析完成的数据:numberOfLines行,3列
27     returnMat = np.zeros((numberOfLines,3))
28     #返回的分类标签向量
29     classLabelVector = []
30     #行的索引值
31     index = 0

```

```

32 for line in arrayOLines:
33     #s.strip(rm), 当rm空时,默认删除空白符(包括'\n','\r','\t',' ')
34     line = line.strip()
35     #使用s.split(str="",num=string,cout(str))将字符串根据'\t'分隔符进行切片。
36     listFromLine = line.split('\t')
37     #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
38     returnMat[index,:] = listFromLine[0:3]
39     #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
40     if listFromLine[-1] == 'didntLike':
41         classLabelVector.append(1)
42     elif listFromLine[-1] == 'smallDoses':
43         classLabelVector.append(2)
44     elif listFromLine[-1] == 'largeDoses':
45         classLabelVector.append(3)
46     index += 1
47 return returnMat, classLabelVector
48
49 """
50 函数说明:可视化数据
51
52 Parameters:
53     datingDataMat - 特征矩阵
54     datingLabels - 分类Label
55 Returns:
56     无
57 Modify:
58     2017-03-24
59 """
60 def showdatas(datingDataMat, datingLabels):
61     #设置汉字格式
62     font = FontProperties(fname=r"c:\windows\fonts\simsun.ttc", size=14)
63     #将fig画布分隔成1行1列,不共享x轴和y轴,fig画布的大小为(13,8)
64     #当nrow=2,nclos=2时,代表fig画布被分为四个区域,axs[0][0]表示第一行第一个区域
65     fig, axs = plt.subplots(nrows=2, ncols=2, sharex=False, sharey=False, figsize=(13,8))
66
67     numberOfLabels = len(datingLabels)
68     LabelsColors = []
69     for i in datingLabels:
70         if i == 1:
71             LabelsColors.append('black')
72         if i == 2:
73             LabelsColors.append('orange')
74         if i == 3:
75             LabelsColors.append('red')
76
77     #画出散点图,以datingDataMat矩阵的第一(飞行常客例程)、第二列(玩游戏)数据画散点数据,散点大小为15,透明度为0.5
78     axs[0][0].scatter(x=datingDataMat[:,0], y=datingDataMat[:,1], color=LabelsColors,s=15, alpha=.5)
79     #设置标题,x轴label,y轴label
80     axs0_title_text = axs[0][0].set_title(u'每年获得的飞行常客里程数与玩视频游戏所消耗时间占比', FontProperties=font)
81     axs0_xlabel_text = axs[0][0].set_xlabel(u'每年获得的飞行常客里程数', FontProperties=font)
82     axs0_ylabel_text = axs[0][0].set_ylabel(u'玩视频游戏所消耗时间占', FontProperties=font)
83     plt.setp(axs0_title_text, size=9, weight='bold', color='red')
84     plt.setp(axs0_xlabel_text, size=7, weight='bold', color='black')
85     plt.setp(axs0_ylabel_text, size=7, weight='bold', color='black')
86
87     #画出散点图,以datingDataMat矩阵的第一(飞行常客例程)、第三列(冰激凌)数据画散点数据,散点大小为15,透明度为0.5
88     axs[0][1].scatter(x=datingDataMat[:,0], y=datingDataMat[:,2], color=LabelsColors,s=15, alpha=.5)
89     #设置标题,x轴label,y轴label
90     axs1_title_text = axs[0][1].set_title(u'每年获得的飞行常客里程数与每周消费的冰激淋公升数', FontProperties=font)
91     axs1_xlabel_text = axs[0][1].set_xlabel(u'每年获得的飞行常客里程数', FontProperties=font)
92     axs1_ylabel_text = axs[0][1].set_ylabel(u'每周消费的冰激淋公升数', FontProperties=font)
93     plt.setp(axs1_title_text, size=9, weight='bold', color='red')
94     plt.setp(axs1_xlabel_text, size=7, weight='bold', color='black')
95     plt.setp(axs1_ylabel_text, size=7, weight='bold', color='black')
96
97     #画出散点图,以datingDataMat矩阵的第二(玩游戏)、第三列(冰激凌)数据画散点数据,散点大小为15,透明度为0.5
98     axs[1][0].scatter(x=datingDataMat[:,1], y=datingDataMat[:,2], color=LabelsColors,s=15, alpha=.5)
99     #设置标题,x轴label,y轴label
100    axs2_title_text = axs[1][0].set_title(u'玩视频游戏所消耗时间占比与每周消费的冰激淋公升数', FontProperties=font)
101    axs2_xlabel_text = axs[1][0].set_xlabel(u'玩视频游戏所消耗时间占比', FontProperties=font)
102    axs2_ylabel_text = axs[1][0].set_ylabel(u'每周消费的冰激淋公升数', FontProperties=font)
103    plt.setp(axs2_title_text, size=9, weight='bold', color='red')
104    plt.setp(axs2_xlabel_text, size=7, weight='bold', color='black')
105    plt.setp(axs2_ylabel_text, size=7, weight='bold', color='black')
106
107    #设置图例
108    didntLike = mlines.Line2D([], [], color='black', marker='.',
109                               markersize=6, label='didntLike')
110    smallDoses = mlines.Line2D([], [], color='orange', marker='.',
111                                markersize=6, label='smallDoses')
112    largeDoses = mlines.Line2D([], [], color='red', marker='.',
113                                markersize=6, label='largeDoses')
114
115    #添加图例
116    axs[0][0].legend(handles=[didntLike, smallDoses, largeDoses])
117    axs[0][1].legend(handles=[didntLike, smallDoses, largeDoses])
118    axs[1][0].legend(handles=[didntLike, smallDoses, largeDoses])
119
120    #显示图片
121    plt.show()
122
123 """
124 函数说明:main函数
125
126 Parameters:
127     无
128 Returns:
129     无
130 Modify:
131     2017-03-24
132 """
133 if __name__ == '__main__':
134     #打开的文件名
135     filename = "datingTestSet.txt"
136     #打开并处理数据
137     datingDataMat, datingLabels = file2matrix(filename)
138     showdatas(datingDataMat, datingLabels)

```

运行上述代码，可以看到可视化结果如图2.3所示。

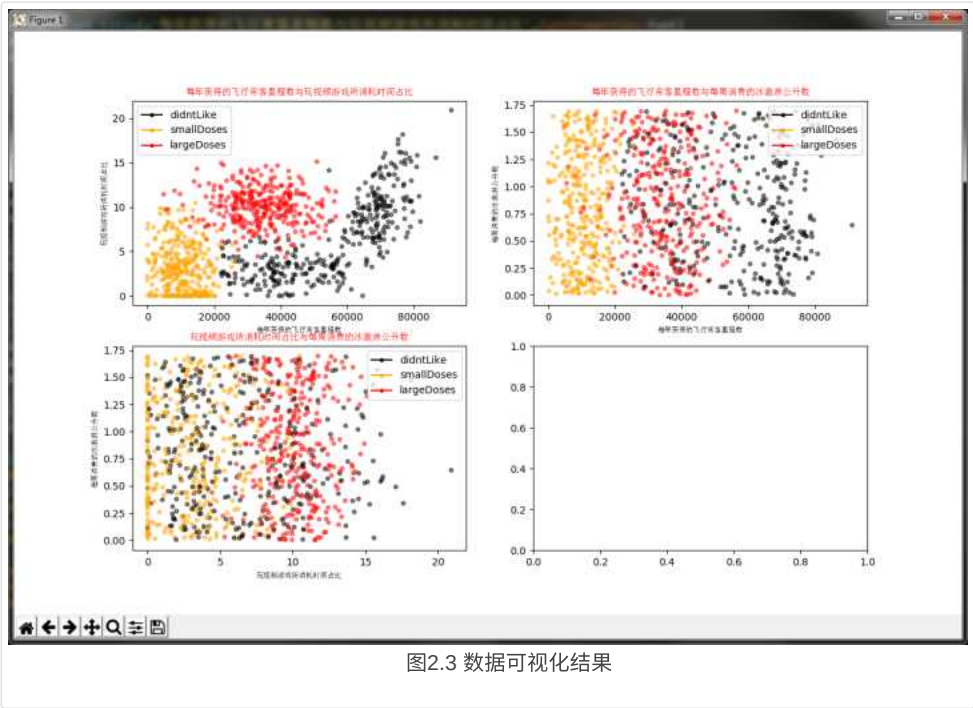


图2.3 数据可视化结果

通过数据可以很直观地发现数据的规律，比如以玩游戏所消耗时间占比与每年获得的飞行常客里程数，只考虑这二维的特征信息，给我的感觉就是海量的男人。为什么这么说呢？每年获得的飞行常客里程数表明，海伦喜欢能享受飞行常客奖励计划的男人，但是不能经常坐飞机，疲于奔波，满世界飞。[也要玩视频游戏，并且占一定时间比例。能到处飞，又能经常玩游戏的男人是什么样的男人？很显然，有生活质量，并且生活悠闲的人。我的分析，仅仅数据总结的个人看法。我想，每个人的感受应该也是不尽相同。

4、准备数据：数据归一化

表2.1给出了四组样本，如果想要计算样本3和样本4之间的距离，可以使用欧拉公式计算。

样本	玩游戏所耗时间百分比	每年获得的飞行常客里程数	每周消费的冰淇淋公斤数	样本分类
1	0.8	400	0.5	1
2	12	134000	0.9	3
3	0	20000	1.1	2
4	67	32000	0.1	2

表2.1 约会网站样本数据

计算方法如图2.4所示。

$$\sqrt{(0 - 67)^2 + (20000 - 32000)^2 + (1.1 - 0.1)^2}$$

图2.4 计算公式

我们很容易发现，上方方程中数字差值最大的属性对计算结果的影响最大，也就是说，每年获取的飞行常客里程数对于计算结果的影响将远远大于表特征-玩视频游戏所耗时间占比和每周消费冰淇淋公斤数的影响。而产生这种现象的唯一原因，仅仅是因为飞行常客里程数远大于其他特征值。但海伦认为同等重要的，因此作为三个等权重的特征之一，飞行常客里程数并不应该如此严重地影响到计算结果。

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值转化为 0 到 1 区间内的值：

newValue = (oldValue - min) / (max - min)

其中min和max分别是数据集中的最小特征值和最大特征值。虽然改变数值取值范围增加了分类器的复杂度，但为了得到准确结果，我们必须这样做。kNN_test02.py文件中编写名为autoNorm的函数，用该函数自动将数据归一化。代码如下：

```
1 # -*- coding: UTF-8 -*-
2 import numpy as np
3
4 """
5 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
6
7 Parameters:
8     filename - 文件名
9 Returns:
```

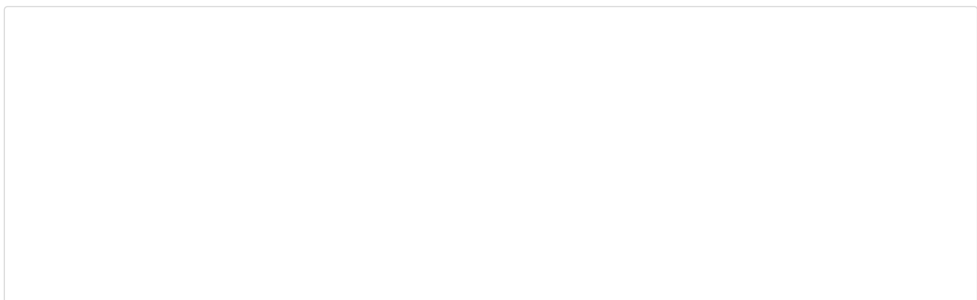


```

10     returnMat - 特征矩阵
11     classLabelVector - 分类Label向量
12
13 Modify:
14     2017-03-24
15 """
16 def file2matrix(filename):
17     #打开文件
18     fr = open(filename)
19     #读取文件所有内容
20     array0Lines = fr.readlines()
21     #得到文件行数
22     numberOfLines = len(array0Lines)
23     #返回的NumPy矩阵,解析完成的数据:numberOfLines行,3列
24     returnMat = np.zeros((numberOfLines,3))
25     #返回的分类标签向量
26     classLabelVector = []
27     #行的索引值
28     index = 0
29     for line in array0Lines:
30         #s.strip(rm),当rm空时,默认删除空白符(包括'\n','\r','\t',' ')
31         line = line.strip()
32         #使用s.split(str="",num=string,cout(str))将字符串根据'\t'分隔符进行切片。
33         listFromLine = line.split('\t')
34         #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
35         returnMat[index,:] = listFromLine[0:3]
36         #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
37         if listFromLine[-1] == 'didntLike':
38             classLabelVector.append(1)
39         elif listFromLine[-1] == 'smallDoses':
40             classLabelVector.append(2)
41         elif listFromLine[-1] == 'largeDoses':
42             classLabelVector.append(3)
43         index += 1
44     return returnMat, classLabelVector
45
46 """
47 函数说明:对数据进行归一化
48
49 Parameters:
50     dataSet - 特征矩阵
51 Returns:
52     normDataSet - 归一化后的特征矩阵
53     ranges - 数据范围
54     minVals - 数据最小值
55
56 Modify:
57     2017-03-24
58 """
59 def autoNorm(dataSet):
60     #获得数据的最小值
61     minVals = dataSet.min(0)
62     maxVals = dataSet.max(0)
63     #最大值和最小值的范围
64     ranges = maxVals - minVals
65     #shape(dataSet)返回dataSet的矩阵行列数
66     normDataSet = np.zeros(np.shape(dataSet))
67     #返回dataSet的行数
68     m = dataSet.shape[0]
69     #原始值减去最小值
70     normDataSet = dataSet - np.tile(minVals, (m, 1))
71     #除以最大和最小值的差,得到归一化数据
72     normDataSet = normDataSet / np.tile(ranges, (m, 1))
73     #返回归一化数据结果,数据范围,最小值
74     return normDataSet, ranges, minVals
75
76 """
77 函数说明:main函数
78
79 Parameters:
80     无
81 Returns:
82     无
83
84 Modify:
85     2017-03-24
86 """
87 if __name__ == '__main__':
88     #打开的文件名
89     filename = "datingTestSet.txt"
90     #打开并处理数据
91     datingDataMat, datingLabels = file2matrix(filename)
92     normDataSet, ranges, minVals = autoNorm(datingDataMat)
93     print(normDataSet)
94     print(ranges)
95     print(minVals)

```

运行上述代码，得到结果如图2.4所示。



```

277 Modify:
278     2017-03-24
279     """
280 if __name__ == '__main__':
281     #打开的文件名
282     filename = "datingTestSet.txt"
283     #打开并处理数据
284     datingDataMat, datingLabels = file2matrix(filename)
285     normDataSet, ranges, minVals = autoNorm(datingDataMat)
286     print(normDataSet)
287     print(ranges)
288     print(minVals)

```

```

[[ 0.44832535  0.39805139  0.56233353]
 [ 0.15873259  0.34195467  0.98724416]
 [ 0.28542943  0.06892523  0.47449629]
 ...,
 [ 0.29115949  0.50910294  0.51079493]
 [ 0.52711097  0.43665451  0.4290048 ]
 [ 0.47940793  0.3768091  0.78571804]]
[ 9.12730000e+04  2.09193490e+01  1.69436100e+00]
[ 0.  0.  0.001156]
[Finished in 6.1s]

```

图2.4 归一化函数运行结果

从图2.4的运行结果可以看到，我们已经顺利将数据归一化了，并且求出了数据的取值范围和数据的最小值，这两个值是在分类的时候需要用到的，直来，也算是对数据预处理了。

5、测试算法：验证分类器

机器学习算法一个很重要的工作就是评估算法的正确率，通常我们只提供已有数据的90%作为训练样本来训练分类器，而使用其余的10%数据去测试：类器的正确率。需要注意的是，10%的测试数据应该是随机选择的，由于海伦提供的数据并没有按照特定目的来排序，所以我们可以随意选择10%数据而：性。

为了测试分类器效果，在kNN_test02.py文件中创建函数datingClassTest，编写代码如下：

```

1  #-*- coding: UTF-8 -*-
2  import numpy as np
3  import operator
4
5  """
6  函数说明:kNN算法,分类器
7
8  Parameters:
9      inX - 用于分类的数据(测试集)
10     dataSet - 用于训练的数据(训练集)
11     labels - 分类标签
12     k - kNN算法参数,选择距离最小的k个点
13 Returns:
14     sortedClassCount[0][0] - 分类结果
15
16 Modify:
17     2017-03-24
18     """
19 def classify0(inX, dataSet, labels, k):
20     #numpy函数shape[0]返回dataSet的行数
21     dataSetSize = dataSet.shape[0]
22     #在列向量方向上重复inX共1次(纵向),行向量方向上重复inX共dataSetSize次(纵向)
23     diffMat = np.tile(inX, (dataSetSize, 1)) - dataSet
24     #二维特征相减后平方
25     sqDiffMat = diffMat**2
26     #sum()所有元素相加,sum(0)列相加,sum(1)行相加
27     sqDistances = sqDiffMat.sum(axis=1)
28     #开方,计算出距离
29     distances = sqDistances**0.5
30     #返回distances中元素从小到大排序后的索引值
31     sortedDistIndices = distances.argsort()
32     #定一个记录类别次数的字典
33     classCount = {}
34     for i in range(k):
35         #取出前k个元素的类别
36         voteIlabel = labels[sortedDistIndices[i]]
37         #dict.get(key,default=None),字典的get()方法,返回指定键的值,如果值不在字典中返回默认值。
38         #计算类别次数
39         classCount[voteIlabel] = classCount.get(voteIlabel,0) + 1
40     #python3中用items()替换python2中的iteritems()
41     #key=operator.itemgetter(1)根据字典的值进行排序
42     #key=operator.itemgetter(0)根据字典的键进行排序
43     #reverse降序排序字典
44     sortedClassCount = sorted(classCount.items(),key=operator.itemgetter(1),reverse=True)
45     #返回次数最多的类别,即所要分类的类别
46     return sortedClassCount[0][0]
47
48 """
49 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
50
51 Parameters:

```

```

52     filename - 文件名
53 Returns:
54     returnMat - 特征矩阵
55     classLabelVector - 分类Label向量
56
57 Modify:
58     2017-03-24
59 """
60 def file2matrix(filename):
61     #打开文件
62     fr = open(filename)
63     #读取文件所有内容
64     array0Lines = fr.readlines()
65     #得到文件行数
66     numberOfLines = len(array0Lines)
67     #返回的NumPy矩阵,解析完成的数据:numberOfLines行,3列
68     returnMat = np.zeros((numberOfLines,3))
69     #返回的分类标签向量
70     classLabelVector = []
71     #行的索引值
72     index = 0
73     for line in array0Lines:
74         #s.strip(rm),当rm空时,默认删除空白符(包括'\n','\r','\t',' ')
75         line = line.strip()
76         #使用s.split(str="",num=string.count(str))将字符串根据'\t'分隔符进行切片。
77         listFromLine = line.split('\t')
78         #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
79         returnMat[index,:] = listFromLine[0:3]
80         #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
81         if listFromLine[-1] == 'didntLike':
82             classLabelVector.append(1)
83         elif listFromLine[-1] == 'smallDoses':
84             classLabelVector.append(2)
85         elif listFromLine[-1] == 'largeDoses':
86             classLabelVector.append(3)
87         index += 1
88     return returnMat, classLabelVector
89
90 """
91 函数说明:对数据进行归一化
92
93 Parameters:
94     dataSet - 特征矩阵
95 Returns:
96     normDataSet - 归一化后的特征矩阵
97     ranges - 数据范围
98     minVals - 数据最小值
99
100 Modify:
101     2017-03-24
102 """
103 def autoNorm(dataSet):
104     #获得数据的最小值
105     minVals = dataSet.min(0)
106     maxVals = dataSet.max(0)
107     #最大值和最小值的范围
108     ranges = maxVals - minVals
109     #shape(dataSet)返回dataSet的矩阵行列数
110     normDataSet = np.zeros(np.shape(dataSet))
111     #返回dataSet的行数
112     m = dataSet.shape[0]
113     #原始值减去最小值
114     normDataSet = dataSet - np.tile(minVals, (m, 1))
115     #除以最大和最小值的差,得到归一化数据
116     normDataSet = normDataSet / np.tile(ranges, (m, 1))
117     #返回归一化数据结果,数据范围,最小值
118     return normDataSet, ranges, minVals
119
120 """
121 函数说明:分类器测试函数
122
123 Parameters:
124     无
125 Returns:
126     normDataSet - 归一化后的特征矩阵
127     ranges - 数据范围
128     minVals - 数据最小值
129
130
131 Modify:
132     2017-03-24
133 """
134 def datingClassTest():
135     #打开的文件名
136     filename = "datingTestSet.txt"
137     #将返回的特征矩阵和分类向量分别存储到datingDataMat和datingLabels中
138     datingDataMat, datingLabels = file2matrix(filename)
139     #取所有数据的百分之十
140     hoRatio = 0.10
141     #数据归一化,返回归一化后的矩阵,数据范围,数据最小值
142     normMat, ranges, minVals = autoNorm(datingDataMat)
143     #获得normMat的行数
144     m = normMat.shape[0]
145     #百分之十的测试数据的个数
146     numTestVecs = int(m * hoRatio)
147     #分类错误计数
148     errorCount = 0.0
149
150     for i in range(numTestVecs):
151         #前numTestVecs个数据作为测试集,后m-numTestVecs个数据作为训练集
152         classifierResult = classify0(normMat[i:], normMat[numTestVecs:m,:],
153                                     datingLabels[numTestVecs:m], 4)
154         print("分类结果:%d\t真实类别:%d" % (classifierResult, datingLabels[i]))
155         if classifierResult != datingLabels[i]:

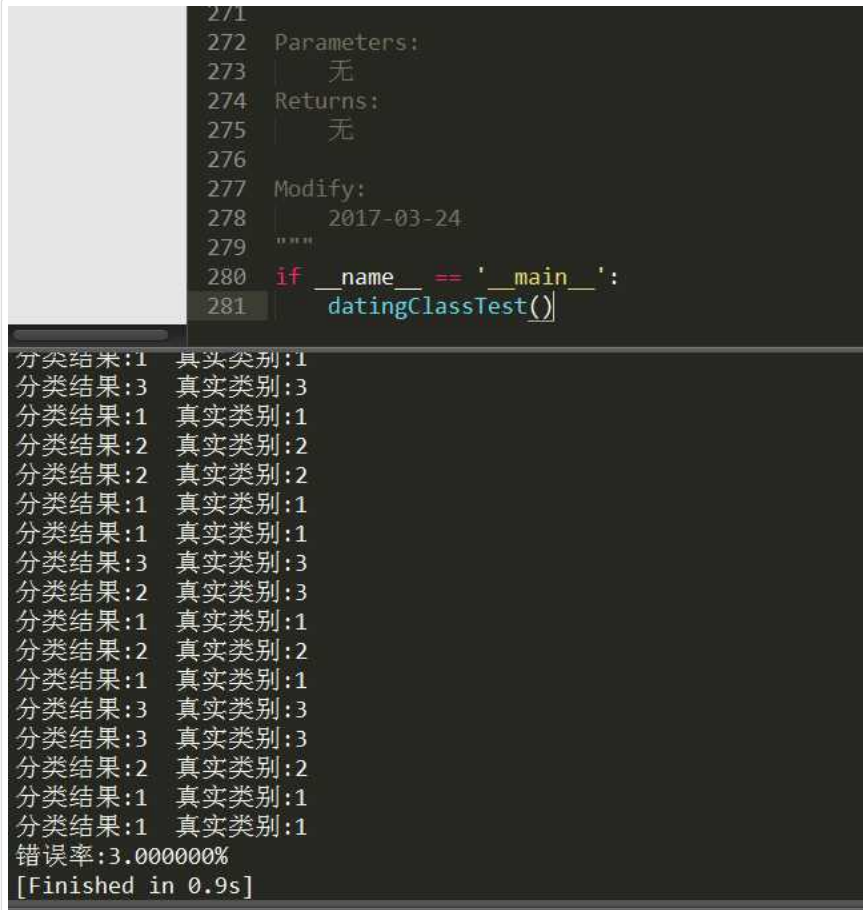
```

```

156         errorCount += 1.0
157     print("错误率:%f%%" %(errorCount/float(numTestVecs)*100))
158
159     """
160     函数说明:main函数
161
162     Parameters:
163     无
164     Returns:
165     无
166
167     Modify:
168     2017-03-24
169     """
170     if __name__ == '__main__':
171         datingClassTest()

```

运行上述代码，得到结果如图2.5所示。



```

271
272 Parameters:
273 无
274 Returns:
275 无
276
277 Modify:
278 2017-03-24
279 """
280 if __name__ == '__main__':
281     datingClassTest()

```

分类结果:1 真实类别:1
分类结果:3 真实类别:3
分类结果:1 真实类别:1
分类结果:2 真实类别:2
分类结果:2 真实类别:2
分类结果:1 真实类别:1
分类结果:1 真实类别:1
分类结果:3 真实类别:3
分类结果:2 真实类别:3
分类结果:1 真实类别:1
分类结果:2 真实类别:2
分类结果:1 真实类别:1
分类结果:3 真实类别:3
分类结果:3 真实类别:3
分类结果:2 真实类别:2
分类结果:1 真实类别:1
分类结果:1 真实类别:1
错误率:3.000000%
[Finished in 0.9s]

图2.5 验证分类器结果

从图2.5验证分类器结果中可以看出，错误率是3%，这是一个相当不错的结果。我们可以改变函数datingClassTest内变量hoRatio和分类器k的值，检测着变量值的变化而增加。依赖于分类算法、数据集和程序设置，分类器的输出结果可能有很大的不同。

6、使用算法：构建完整可用系统

我们可以给海伦一个小段程序，通过该程序海伦会在约会网站上找到某个人并输入他的信息。程序会给出她对男方喜欢程度的预测值。

在kNN_test02.py文件中创建函数classifyPerson，代码如下：

```

1  # -*- coding: UTF-8 -*-
2  import numpy as np
3  import operator
4
5  """
6  函数说明:kNN算法,分类器
7
8  Parameters:
9      inX - 用于分类的数据(测试集)
10     dataSet - 用于训练的数据(训练集)
11     labels - 分类标签
12     k - kNN算法参数,选择距离最小的k个点
13 Returns:
14     sortedClassCount[0][0] - 分类结果
15
16 Modify:
17     2017-03-24
18 """
19 def classify0(inX, dataSet, labels, k):
20     #numpy函数shape[0]返回dataSet的行数
21     dataSetSize = dataSet.shape[0]
22     #在列向量方向上重复inX共1次(横向),行向量方向上重复inX共dataSetSize次(纵向)

```

```

23 diffMat = np.tile(inX, (dataSetSize, 1)) - dataSet
24 #二维特征相减后平方
25 sqDiffMat = diffMat**2
26 #sum()所有元素相加,sum(0)列相加,sum(1)行相加
27 sqDistances = sqDiffMat.sum(axis=1)
28 #开方,计算出距离
29 distances = sqDistances**0.5
30 #返回distances中元素从小到大排序后的索引值
31 sortedDistIndices = distances.argsort()
32 #定一个记录类别次数的字典
33 classCount = {}
34 for i in range(k):
35     #取出前k个元素的类别
36     voteLabel = labels[sortedDistIndices[i]]
37     #dict.get(key,default=None),字典的get()方法,返回指定键的值,如果值不在字典中返回默认值。
38     #计算类别次数
39     classCount[voteLabel] = classCount.get(voteLabel,0) + 1
40 #python3中用items()替换python2中的iteritems()
41 #key=operator.itemgetter(1)根据字典的值进行排序
42 #key=operator.itemgetter(0)根据字典的键进行排序
43 #reverse降序排序字典
44 sortedClassCount = sorted(classCount.items(),key=operator.itemgetter(1),reverse=True)
45 #返回次数最多的类别,即所要分类的类别
46 return sortedClassCount[0][0]
47
48
49 """
50 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
51
52 Parameters:
53     filename - 文件名
54 Returns:
55     returnMat - 特征矩阵
56     classLabelVector - 分类Label向量
57
58 Modify:
59     2017-03-24
60 """
61 def file2matrix(filename):
62     #打开文件
63     fr = open(filename)
64     #读取文件所有内容
65     array0Lines = fr.readlines()
66     #得到文件行数
67     numberOfLines = len(array0Lines)
68     #返回的NumPy矩阵,解析完成的数据:numberOfLines行,3列
69     returnMat = np.zeros((numberOfLines,3))
70     #返回的分类标签向量
71     classLabelVector = []
72     #行的索引值
73     index = 0
74     for line in array0Lines:
75         #s.strip(rm),当rm空时,默认删除空白符(包括'\n','\r','\t',' ')
76         line = line.strip()
77         #使用s.split(str="",num=string,cout(str))将字符串根据'\t'分隔符进行切片。
78         listFromLine = line.split('\t')
79         #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
80         returnMat[index,:] = listFromLine[0:3]
81         #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
82         if listFromLine[-1] == 'didntLike':
83             classLabelVector.append(1)
84         elif listFromLine[-1] == 'smallDoses':
85             classLabelVector.append(2)
86         elif listFromLine[-1] == 'largeDoses':
87             classLabelVector.append(3)
88         index += 1
89     return returnMat, classLabelVector
90
91 """
92 函数说明:对数据进行归一化
93
94 Parameters:
95     dataSet - 特征矩阵
96 Returns:
97     normDataSet - 归一化后的特征矩阵
98     ranges - 数据范围
99     minVals - 数据最小值
100
101 Modify:
102     2017-03-24
103 """
104 def autoNorm(dataSet):
105     #获得数据的最小值
106     minVals = dataSet.min(0)
107     maxVals = dataSet.max(0)
108     #最大值和最小值的范围
109     ranges = maxVals - minVals
110     #shape(dataSet)返回dataSet的矩阵行列数
111     normDataSet = np.zeros(np.shape(dataSet))
112     #返回dataSet的行数
113     m = dataSet.shape[0]
114     #原始值减去最小值
115     normDataSet = dataSet - np.tile(minVals, (m, 1))
116     #除以最大和最小值的差,得到归一化数据
117     normDataSet = normDataSet / np.tile(ranges, (m, 1))
118     #返回归一化数据结果,数据范围,最小值
119     return normDataSet, ranges, minVals
120
121 """
122 函数说明:通过输入一个人的三维特征,进行分类输出
123
124 Parameters:
125     无
126 Returns:

```



```
127     无
128
129     Modify:
130         2017-03-24
131     """
132     def classifyPerson():
133         #输出结果
134         resultList = ['讨厌', '有些喜欢', '非常喜欢']
135         #三维特征用户输入
136         precentTats = float(input("玩视频游戏所耗时间百分比:"))
137         ffMiles = float(input("每年获得的飞行常客里程数:"))
138         iceCream = float(input("每周消费的冰激淋公升数:"))
139         #打开的文件名
140         filename = "datingTestSet.txt"
141         #打开并处理数据
142         datingDataMat, datingLabels = file2matrix(filename)
143         #训练集归一化
144         normMat, ranges, minVals = autoNorm(datingDataMat)
145         #生成NumPy数组,测试集
146         inArr = np.array([ffMiles, precentTats, iceCream])
147         #测试集归一化
148         norminArr = (inArr - minVals) / ranges
149         #返回分类结果
150         classifierResult = classify0(norminArr, normMat, datingLabels, 3)
151         #打印结果
152         print("你可能%s这个人" % (resultList[classifierResult-1]))
153
154     """
155     函数说明:main函数
156
157     Parameters:
158         无
159     Returns:
160         无
161
162     Modify:
163         2017-03-24
164     """
165     if __name__ == '__main__':
166         classifyPerson()
```

在cmd中，运行程序，并输入数据(12,44000,0.5)，预测结果是"你可能有些喜欢这个人"，也就是这个人魅力一般。一共有三个档次：讨厌、有些喜欢、应着不喜欢的人、魅力一般的人、极具魅力的人。结果如图2.6所示。



三、k-近邻算法实战之sklearn手写数字识别

1、实战背景

对于需要识别的数字已经使用图形处理软件，处理成具有相同的色彩和大小：宽高是32像素x32像素。尽管采用本文格式存储图像不能有效地利用内存，为了方便理解，我们将图片转换为文本格式，数字的文本格式如图3.1所示。



图3.1 数字的文本格式

与此同时，这些文本格式存储的数字的文件命名也很有特点，格式为：数字的值_该数字的样本序号，如图3.2所示。



4_174.txt	2017/3/28 14:33	TXT 文件	2 KB
4_175.txt	2017/3/28 14:33	TXT 文件	2 KB
4_176.txt	2017/3/28 14:33	TXT 文件	2 KB
4_177.txt	2017/3/28 14:33	TXT 文件	2 KB
4_178.txt	2017/3/28 14:33	TXT 文件	2 KB
4_179.txt	2017/3/28 14:33	TXT 文件	2 KB
4_180.txt	2017/3/28 14:33	TXT 文件	2 KB
4_181.txt	2017/3/28 14:33	TXT 文件	2 KB
4_182.txt	2017/3/28 14:33	TXT 文件	2 KB
4_183.txt	2017/3/28 14:33	TXT 文件	2 KB
4_184.txt	2017/3/28 14:33	TXT 文件	2 KB
4_185.txt	2017/3/28 14:33	TXT 文件	2 KB
5_0.txt	2017/3/28 14:33	TXT 文件	2 KB
5_1.txt	2017/3/28 14:33	TXT 文件	2 KB
5_2.txt	2017/3/28 14:33	TXT 文件	2 KB
5_3.txt	2017/3/28 14:33	TXT 文件	2 KB
5_4.txt	2017/3/28 14:33	TXT 文件	2 KB
5_5.txt	2017/3/28 14:33	TXT 文件	2 KB
5_6.txt	2017/3/28 14:33	TXT 文件	2 KB
5_7.txt	2017/3/28 14:33	TXT 文件	2 KB
5_8.txt	2017/3/28 14:33	TXT 文件	2 KB
5_9.txt	2017/3/28 14:33	TXT 文件	2 KB
5_10.txt	2017/3/28 14:33	TXT 文件	2 KB
5_11.txt	2017/3/28 14:33	TXT 文件	2 KB
5_12.txt	2017/3/28 14:33	TXT 文件	2 KB
5_13.txt	2017/3/28 14:33	TXT 文件	2 KB

图3.2 文本数字的存储格式

对于这样已经整理好的文本，我们可以直接使用Python处理，进行数字预测。数据集分为训练集和测试集，使用上小结的方法，自己设计k-近邻算法：现分类。数据集和实现代码下载地址：[数据集下载](#)

这里不再讲解自己用Python写的k-邻域分类器的方法，因为这不是本小节的重点。接下来，我们将使用强大的第三方Python科学计算库Sklearn构建手

2、sklearn简介

Scikit learn 也简称sklearn，是机器学习领域当中最知名的python模块之一。sklearn包含了很多机器学习的方式：

- Classification 分类
- Regression 回归
- Clustering 非监督分类
- Dimensionality reduction 数据降维
- Model Selection 模型选择
- Preprocessing 数据与处理

使用sklearn可以很方便地让我们实现一个机器学习算法。一个复杂度算法的实现，使用sklearn可能只需要调用几行API即可。所以学习sklearn，可以特定任务的实现周期。

3、sklearn安装

在安装sklearn之前，需要安装两个库，即numpy+mkl和scipy。不要使用pip3直接进行安装，因为pip3默安装的是numpy，而不是numpy+mkl。第三方<http://www.lfd.uci.edu/~gohlke/pythonlibs/>

这个网站的使用方法，我在之前的文章里有讲过：<http://blog.csdn.net/c406495762/article/details/60156205>

找到对应python版本的numpy+mkl和scipy，下载安装即可，如图3.3和图3.4所示。

NumPy, a fundamental package needed for scientific computing with Python.
Numpy+MKL is linked to the Intel® Math Kernel Library and includes required DLLs in the numpy.core directory.

[numpy-1.13.1+mkl-cp27-cp27m-win32.whl](#)
[numpy-1.13.1+mkl-cp27-cp27m-win_amd64.whl](#)
[numpy-1.13.1+mkl-cp34-cp34m-win32.whl](#)
[numpy-1.13.1+mkl-cp34-cp34m-win_amd64.whl](#)
[numpy-1.13.1+mkl-cp35-cp35m-win32.whl](#)
[numpy-1.13.1+mkl-cp35-cp35m-win_amd64.whl](#)
[numpy-1.13.1+mkl-cp36-cp36m-win32.whl](#)
[numpy-1.13.1+mkl-cp36-cp36m-win_amd64.whl](#)

图3.3 numpy+mkl

SciPy is software for mathematics, science, and engineering. Install numpy+mkl before installing scipy.

[scipy-0.19.1-cp27-cp27m-win32.whl](#)
[scipy-0.19.1-cp27-cp27m-win_amd64.whl](#)
[scipy-0.19.1-cp34-cp34m-win32.whl](#)
[scipy-0.19.1-cp34-cp34m-win_amd64.whl](#)
[scipy-0.19.1-cp35-cp35m-win32.whl](#)
[scipy-0.19.1-cp35-cp35m-win_amd64.whl](#)
[scipy-0.19.1-cp36-cp36m-win32.whl](#)
[scipy-0.19.1-cp36-cp36m-win_amd64.whl](#)

图3.4 scipy

使用pip3安装好这两个whl文件后，使用如下指令安装sklearn。

```
1 pip3 install -U scikit-learn
```

4、sklearn实现k-近邻算法简介

官网英文文档：[点我查看](#)

sklearn.neighbors模块实现了k-近邻算法，内容如图3.5所示。

User guide: See the Nearest Neighbors section for further details.

<code>neighbors.NearestNeighbors</code> ([n_neighbors, ...])	Unsupervised learner for implementing neighbor searches.
<code>neighbors.KNeighborsClassifier</code> ([...])	Classifier implementing the k-nearest neighbors vote.
<code>neighbors.RadiusNeighborsClassifier</code> ([...])	Classifier implementing a vote among neighbors within a given radius.
<code>neighbors.KNeighborsRegressor</code> ([n_neighbors, ...])	Regression based on k-nearest neighbors.
<code>neighbors.RadiusNeighborsRegressor</code> ([radius, ...])	Regression based on neighbors within a fixed radius.
<code>neighbors.NearestCentroid</code> ([metric, ...])	Nearest centroid classifier.
<code>neighbors.BallTree</code>	BallTree for fast generalized N-point problems
<code>neighbors.KDTree</code>	KDTree for fast generalized N-point problems
<code>neighbors.LSHForest</code> ([n_estimators, radius, ...])	Performs approximate nearest neighbor search using LSH forest.
<code>neighbors.DistanceMetric</code>	DistanceMetric class
<code>neighbors.KernelDensity</code> ([bandwidth, ...])	Kernel Density Estimation
<code>neighbors.kneighbors_graph</code> (X, n_neighbors[, ...])	Computes the (weighted) graph of k-Neighbors for points in X
<code>neighbors.radius_neighbors_graph</code> (X, radius)	Computes the (weighted) graph of Neighbors for points in X

图3.5 sklearn.neighbors

我们使用sklearn.neighbors.KNeighborsClassifier就可以是实现上小结，我们实现的k-近邻算法。KNeighborsClassifier函数一共有8个参数，如图3.6所示。

Methods

<code>fit</code> (X, y)	Fit the model using X as training data and y as target values
<code>get_params</code> ([deep])	Get parameters for this estimator.
<code>kneighbors</code> (X, n_neighbors, return_distance)	Finds the K-neighbors of a point.
<code>kneighbors_graph</code> (X, n_neighbors, mode)	Computes the (weighted) graph of k-Neighbors for points in X
<code>predict</code> (X)	Predict the class labels for the provided data
<code>predict_proba</code> (X)	Return probability estimates for the test data X.
<code>score</code> (X, y[, sample_weight])	Returns the mean accuracy on the given test data and labels.
<code>set_params</code> ({"params"})	Set the parameters of this estimator.

图3.6 KNeighborsClassifier

KNeighborsClassifier参数说明：

- `n_neighbors`：默认为5，就是k-NN的k的值，选取最近的k个点。
- `weights`：默认是uniform，参数可以是uniform、distance，也可以是用户自己定义的函数。uniform是均等的权重，就说所有的邻近点的权重都是相等的；distance是不均等的权重，距离近的点比距离远的点的影响大。用户自定义的函数，接收距离的数组，返回一组维数相同的权重。
- `algorithm`：快速k近邻搜索算法，默认参数为auto，可以理解为算法自己决定合适的搜索算法。除此之外，用户也可以自己指定搜索算法ball_tree、brute方法进行搜索，brute是蛮力搜索，也就是线性扫描，当训练集很大时，计算非常耗时。kd_tree，构造kd树存储数据以便对其进行快速检索的；kd树也就是数据结构中的二叉树。以中值切分构造的树，每个结点是一个超矩形，在维数小于20时效率高。ball tree是为了克服kd树高维失效的问题，其构造过程是以质心C和半径r分割样本空间，每个节点是一个超球体。
- `leaf_size`：默认是30，这个是构造的kd树和ball树的大小。这个值的设置会影响树构建的速度和搜索速度，同样也影响着存储树所需的内存大小。需要平衡搜索效率和内存消耗的性质选择最优的大小。
- `metric`：用于距离度量，默认度量是minkowski，也就是p=2的欧氏距离(欧几里德度量)。
- `p`：距离度量公式。在上小结，我们使用欧氏距离公式进行距离度量。除此之外，还有其他的度量方法，例如曼哈顿距离。这个参数默认为2，也就使用欧氏距离公式进行距离度量。也可以设置为1，使用曼哈顿距离公式进行距离度量。
- `metric_params`：距离公式的其他关键参数，这个可以不管，使用默认的None即可。

- `n_jobs`：并行处理设置。默认为1，临近点搜索并行工作数。如果为-1，那么CPU的所有cores都用于并行工作。

`KNeighborsClassifier`提供了以一些方法供我们使用，如图3.7所示。

```

89
90 """
91 函数说明:main函数
92
93 Parameters:
94     无
95 Returns:
96     无
97
98 Modify:
99     2017-07-15
100 """
101 if __name__ == '__main__':
102     handwritingClassTest()
103
分类返回结果为9 真实结果为9
分类返回结果为9 真实结果为9
分类返回结果为9 真实结果为9
分类返回结果为9 真实结果为9
分类返回结果为9 真实结果为9
分类返回结果为9 真实结果为9
分类返回结果为9 真实结果为9
分类返回结果为9 真实结果为9
分类返回结果为9 真实结果为9
分类返回结果为9 真实结果为9
总共错了12个数据
错误率为1.268499%
[Finished in 7.4s]

```

图3.7 KNeighborsClassifier的方法

由于篇幅原因，每个函数的怎么用，就不具体讲解了。官方手册：[点我查看](#) 已经讲解的很详细了，各位可以查看这个手册进行学习，我们直接讲手写的实现。

5、sklearn小试牛刀

我们知道数字图片是32x32的二进制图像，为了方便计算，我们可以将32x32的二进制图像转换为1x1024的向量。对于sklearn的KNeighborsClassifier阵，不用一定转换为向量，不过为了跟自己写的k-近邻算法分类器对应上，这里也做了向量化处理。然后构建kNN分类器，利用分类器做预测。创建kNN_件，编写代码如下：

```

1  #-*- coding: UTF-8 -*-
2  import numpy as np
3  import operator
4  from os import listdir
5  from sklearn.neighbors import KNeighborsClassifier as kNN
6
7  """
8  函数说明:将32x32的二进制图像转换为1x1024向量。
9
10 Parameters:
11     filename - 文件名
12 Returns:
13     returnVect - 返回的二进制图像的1x1024向量
14
15 Modify:
16     2017-07-15
17 """
18 def img2vector(filename):
19     #创建1x1024零向量
20     returnVect = np.zeros((1, 1024))
21     #打开文件
22     fr = open(filename)
23     #按行读取
24     for i in range(32):
25         #读一行数据
26         lineStr = fr.readline()
27         #每一行的前32个元素依次添加到returnVect中
28         for j in range(32):
29             returnVect[0, 32*i+j] = int(lineStr[j])
30     #返回转换后的1x1024向量
31     return returnVect
32
33 """
34 函数说明:手写数字分类测试
35
36 Parameters:
37     无
38 Returns:
39     无
40

```

```

41 Modify:
42     2017-07-15
43 """
44 def handwritingClassTest():
45     #测试集的Labels
46     hwLabels = []
47     #返回trainingDigits目录下的文件名
48     trainingFileList = listdir('trainingDigits')
49     #返回文件夹下文件的个数
50     m = len(trainingFileList)
51     #初始化训练的Mat矩阵，测试集
52     trainingMat = np.zeros((m, 1024))
53     #从文件名中解析出训练集的类别
54     for i in range(m):
55         #获得文件的名字
56         fileNameStr = trainingFileList[i]
57         #获得分类的数字
58         classNumber = int(fileNameStr.split('_')[0])
59         #将获得的类别添加到hwLabels中
60         hwLabels.append(classNumber)
61         #将每一个文件的1x1024数据存储到trainingMat矩阵中
62         trainingMat[i,:] = img2vector('trainingDigits/%s' % (fileNameStr))
63     #构建kNN分类器
64     neigh = kNN(n_neighbors = 3, algorithm = 'auto')
65     #拟合模型， trainingMat为训练矩阵，hwLabels为对应的标签
66     neigh.fit(trainingMat, hwLabels)
67     #返回testDigits目录下的文件列表
68     testFileList = listdir('testDigits')
69     #错误检测计数
70     errorCount = 0.0
71     #测试数据的数量
72     mTest = len(testFileList)
73     #从文件中解析出测试集的类别并进行分类测试
74     for i in range(mTest):
75         #获得文件的名字
76         fileNameStr = testFileList[i]
77         #获得分类的数字
78         classNumber = int(fileNameStr.split('_')[0])
79         #获得测试集的1x1024向量,用于训练
80         vectorUnderTest = img2vector('testDigits/%s' % (fileNameStr))
81         #获得预测结果
82         # classifierResult = classify0(vectorUnderTest, trainingMat, hwLabels, 3)
83         classifierResult = neigh.predict(vectorUnderTest)
84         print("分类返回结果为%d\t真实结果为%d" % (classifierResult, classNumber))
85         if(classifierResult != classNumber):
86             errorCount += 1.0
87     print("总共错了%d个数据\n错误率为%f%%" % (errorCount, errorCount/mTest * 100))
88
89 """
90 函数说明:main函数
91 Parameters:
92     无
93 Returns:
94     无
95
96
97
98 Modify:
99     2017-07-15
100 """
101 if __name__ == '__main__':
102     handwritingClassTest()

```

运行上述代码，得到如图3.8所示的结果。




```
89 """
90 
91 函数说明:main函数
92 
93 Parameters:
94     无
95 Returns:
96     无
97 
98 Modify:
99     2017-07-15
100 """
101 if __name__ == '__main__':
102     handwritingClassTest()
103 
```

分类返回结果为9 真实结果为9
分类返回结果为9 真实结果为9
分类返回结果为9 真实结果为9
分类返回结果为9 真实结果为9
分类返回结果为9 真实结果为9
分类返回结果为9 真实结果为9
分类返回结果为9 真实结果为9
分类返回结果为9 真实结果为9
分类返回结果为9 真实结果为9
分类返回结果为9 真实结果为9
总共错了12个数据
错误率为1.268499%
[Finished in 7.4s]

图3.8 sklearn运行结果

上述代码使用的algorithm参数是auto，更改algorithm参数为brute，使用暴力搜索，你会发现，运行时间变长了，变为10s+。更改n_neighbors参数，不同的值，检测精度也是不同的。自己可以尝试更改这些参数的设置，加深对其函数的理解。

四、总结

1、kNN算法的优缺点

优点

- 简单好用，容易理解，精度高，理论成熟，既可以用来做分类也可以用来做回归；
- 可用于数值型数据和离散型数据；
- 训练时间复杂度为 $O(n)$ ；无数据输入假定；
- 对异常值不敏感

缺点

- 计算复杂性高；空间复杂性高；
- 样本不平衡问题（即有些类别的样本数量很多，而其它样本的数量很少）；
- 一般数值很大的时候不用这个，计算量太大。但是单个样本又不能太少，否则容易发生误分。
- 最大的缺点是无法给出数据的内在含义。

2、其他

- 关于algorithm参数kd_tree的原理，可以查看《统计学方法 李航》书中的讲解；
- 关于距离度量的方法还有切比雪夫距离、马氏距离、巴氏距离等；
- 下篇文章将讲解决策树，欢迎各位的捧场！
- 如有问题，请留言。如有错误，还望指正，谢谢！

PS: 如果觉得本篇本章对您有所帮助，欢迎关注、评论、赞！

参考资料:

1. 本文中提到的电影类别分类、约会网站配对效果判定、手写数字识别实例和数据集，均来自于《机器学习实战》的第二章k-近邻算法。
2. 本文的理论部分，参考自《统计学习方法 李航》的第三章k近邻法以及《机器学习实战》的第二章k-近邻算法。



JackCui

关注人工智能及互联网的个人博客

[查看熊掌号](#)