دانشگاه تهران

دانشکده علوم و فنون نوین

KNN

نام و نام خانوادگی	رامتین کبیری

فهرست گزارش سوال‌ها

تعاریف و مقدمات..... 1
بررسی کد..... 1
بررسی نتایج..... 4

تعاریف و مقدمات

دسته بندی با الگوریتم K-نزدیکترین همسایگی بر اساس فاصله داده مورد بررسی با داده های برچسب دار موجود در دیتاست آموزش انجام می پذیرد.

در این مسئله از فاصله اقلیدسی به عنوان معیار اندازه گیری فاصله استفاده شده است. فرمول فاصله اقلیدسی برای داده a و b با تعداد I مشخصه به شکل زیر است:

$$D_{(a,b)} = sqrt(sum_{k=(0-I)} (a_k - b_k)^2))$$

در این الگوریتم k تا از نزدیکترین همسایه های داده مورد بررسی را پیدا می کنیم سپس برچسب های آن ها را می شماریم و هر برچسبی که تعدادش بیشتر بود را به داده مورد بررسی می دهیم.

بررسی کد

بخش های کد به شرح زیر می باشند:

1. تابع محاسبه فاصله اقلیدسی: این تابع فاصله دو داده a و b را محاسبه می کند.

```
### distance calculator: {
def distance(a, b):
    l = len(a)
    sum_out = 0
    for c in range(0, l):
        sum_out += math.pow((a[c] - b[c]), 2)

    dist = math.sqrt(sum_out)
    return dist

# #check : {
# a_test = np.array([1, 3, 6, 2])
# b_test = np.array([5, 2, 0, 1])
# print(distance(a_test, b_test))
# }
### }
```

2. الگوریتم k-نزدیکترین همسایگی: این تابع براساس یک دیتاست آموزش و تعداد همسایگی های مورد نظر به

داده های دیتاست آزمایش برچسب می زند و مقدار خطای برچسب زنی خود را محاسبه می کند.

```
### k-nearest neighbor : {
def knn(train, test, k):
    q_test = np.shape(test)
    i_test = q_test[0]
    j = q_test[1]
    q_train = np.shape(train)
    i_train = q_train[0]
    t = 0
    f = 0

    for row in range(0, i_test):
        a = test[row, 0:(j-1)]
        result = test[row, -1]
        dist_list = []
        for index in range(0, i_train):
            b = train[index, 0:(j-1)]
            dist = distance(a, b)
            dist_list.append(((float(dist)), index))

        knn_list = []
        for k_neighbor in range(0, k):
            knn_list.append(min(dist_list))
            dist_list.remove(min(dist_list))

        neighbors = []
        for column in range(0, k):
            w = knn_list.pop(0)
            neighbors.append(float(w[1]))


        neighbor_result = []
        for column in range(0, k):
            e = neighbors.pop(0)
            nbresult = train[int(e), -1]
            neighbor_result.append(float(nbresult))

        r0 = neighbor_result.count(float(0))
        r1 = neighbor_result.count(float(1))
        r2 = neighbor_result.count(float(2))
        r3 = neighbor_result.count(float(3))
        vote = []
        vote.append((r0,0))
        vote.append((r1,1))
        vote.append((r2,2))
        vote.append((r3,3))
```

```
        maximum = max(vote)
        knn_result = maximum[1]
        if float(knn_result) == float(result):
            t += 1
        elif float(knn_result) != float(result):
            f += 1

    error = ((f) / (f + t))
    # print("true: ", t)
    # print("false: ", f)
    # print("error: ",error)
    return error

### }
```

3. خواندن داده های مسئله و تبدیل آن ها به فرم های قابل استفاده : در این بخش داده ها از روی فایل Thyroid.txt خوانده و سپس به یک ماتریس تبدیل می شود و در ادامه برای انجام cross validation به 5 قسمت مساوی تقسیم می شود و در 5 بار انجام cross validation هربار یک قسمت به عنوان آزمایش و باقی چهار قسمت برای آموزش درنظر گرفته شده اند.
در ادامه یکبار هم داده ها به فرم نرمالیزه درآمده و مجدد تبدیلات لازم برای ساخت داده های مورد نیاز cross validation این بار روی داده های نرمالیزه انجام گرفته است. برای نرمالیزه کردن داده ها (بجز باینری و مقدار برچسب) از فرمول زیر استفاده شده است:

$$x\_new = |(x\_old - avg(x))| / variance(x)$$

4. دریافت خروجی : داده هایی که برای cross validation ساختیم را یک بار در فرم ساده و بار دیگر در فرم نرمالیزه به تابع k-نزدیکترین همسایگی می دهیم و این کار را برای k های 1 تا 10 انجام میدهیم و برای هر k مقدار میانگین را حساب می کنیم .
5. رسم نمودار : روی یک نمودار، میانگین خطای الگوریتم k-نزدیکترین همسایگی را برای هر دو حالت اصلی و نرمالیزه رسم نمودیم.

```
### error avrage diagram : {
fig, ax = plt.subplots()
k_list= [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

```
ax.plot(k_list, avrage, 'b', linewidth=2.0)
ax.plot(k_list, navrage, 'g', linewidth=2.0)
plt.show()

### }
```

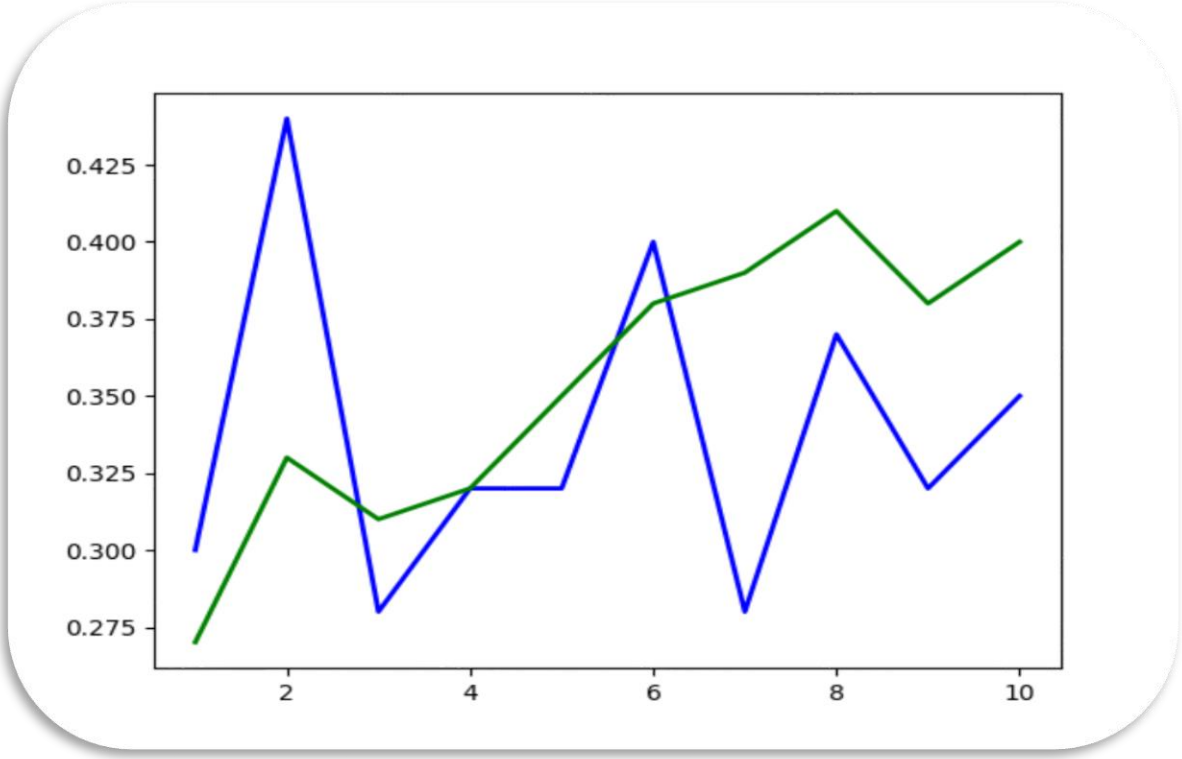
بررسی نتایج

مقادیری خروجی برنامه به شرح جدول زیر است:

k	Original data	Normalized data
1	0.3	0.27
2	0.44000000000000006	0.32999999999999996
3	0.28	0.31
4	0.32	0.31999999999999995
5	0.32	0.35
6	0.4	0.38
7	0.27999999999999997	0.39
8	0.37	0.41
9	0.31999999999999995	0.38
10	0.35	0.4

کمترین میزان خطا برای حالت اصلی با $k = 7$ اتفاق افتاده است و در حالت نرمالیزه با مقدار $k = 1$.

درنمودار رسم شده برای این جدوا نیز نحوه تغییرات خطا با افزایش k را می توانید مشاهده کنید.



سبز : نرمالیزه - آبی : اصلی