Pandas系列4-数据矢量化

geekpy (/u/e5fa627c0613) ＋关注
2018.06.21 19:51 字数 745 阅读 490 评论 2 喜欢 1
(/u/e5fa627c0613)

问题

我们在处理数据问题时，经常会遇到的问题是要将原有数据进行转化，比如在原有数据的基础上+1操作，或者将原有数据的字符串全部转化为小写字符，更复杂的是要将原有数据的一部分提取出来使用。这些问题都是数据转化问题，即原有的数据不能直接使用，而要进一步转化后才能使用。

示例

这里举一个笔者在实际项目中遇到的例子来说明。

笔者项目中需要收集的app version信息，原始信息如下：

```
In [167]: df
Out[167]:
        app_version  uid
0  7.23.1-180522122    1
1  7.20.1-180502135    2
2  7.23.1-180522122    3
3  7.23.1-180522122    4
4  7.16.7-180411077    5
```

但是实际上，我们只需要"-"之前的版本号，而且后续比较的时候要用'-'之前的数字进行比较，因此这样就涉及到了将原版本数据进行转化，即只提取'-'之前的数字，而舍弃后边的数字。

迭代

一个显而易见的做法是通过遍历的方式来逐行修改，如下图所示：

```
In [178]: %%timeit
     ...: for index, row in df.iterrows():
     ...:     df.iloc[index, 0] = row['app_version'].split('-')[0]
     ...:
2.34 ms ± 47.4 µs per loop (mean ± std. dev. of 7 runs, 100 loops each)

In [179]: df
Out[179]:
  app_version  uid
0      7.23.1    1
1      7.20.1    2
2      7.23.1    3
3      7.23.1    4
4      7.16.7    5
```


再进一步，我们可以使用apply方法，如下：

```
In [181]: df
Out[181]:
        app_version  uid
0  7.23.1-180522122    1
1  7.20.1-180502135    2
2  7.23.1-180522122    3
3  7.23.1-180522122    4
4  7.16.7-180411077    5

In [182]: %%timeit
     ...: df['app_version'] = df['app_version'].apply(lambda x: x.split('-')[0])
     ...:
247 µs ± 11.4 µs per loop (mean ± std. dev. of 7 runs, 1000 loops each)

In [183]: df
Out[183]:
  app_version  uid
0      7.23.1    1
1      7.20.1    2
2      7.23.1    3
3      7.23.1    4
4      7.16.7    5
```

我们可以发现使用 `apply` 不仅使得代码更加简洁，而且速度也有了较明显的提升。但是以上方法本质上都是通过迭代的方式一条一条的修改，那么我们能否进一步提升性能呢？

矢量化

```
In [197]: df
Out[197]:
        app_version  uid
0  7.23.1-180522122    1
1  7.20.1-180502135    2
2  7.23.1-180522122    3
3  7.23.1-180522122    4
4  7.16.7-180411077    5

In [198]: %%timeit
     ...: df['app_version'] = df['app_version'].str.split('-').str.get(0)
     ...:
424 µs ± 11.3 µs per loop (mean ± std. dev. of 7 runs, 1000 loops each)
```

这里发现矢量化貌似不能提高性能啊，这是为什么？

> 这里我猜测是由于我们的矢量化代码是分为两步操作，且在数据量较小的情况下就会显得慢

为了验证这个假设，我做了如下实验：
先将原数据concat为2560条记录，然后再计算时间

```
2557  7.23.1-180522122    3
2558  7.23.1-180522122    4
2559  7.16.7-180411077    5

[2560 rows x 2 columns]

In [232]: df3 = df

In [233]: %%timeit
     ...: df['app_version'] = df['app_version'].apply(lambda x: x.split('-')[0])
     ...:
1.36 ms ± 35.4 µs per loop (mean ± std. dev. of 7 runs, 1000 loops each)

# 矢量化方式
In [250]: %%timeit
     ...: df['app_version'] = df['app_version'].str.split("-").str.get(0)
     ...:
2.61 ms ± 113 µs per loop (mean ± std. dev. of 7 runs, 100 loops each)
```

发现单纯的数据量增大并没有影响结果，那么用其它转化来测试下，这里获取字符串长度的转化进行实验

```
In [253]: %%timeit
     ...: df['length'] = df['app_version'].str.len()
     ...:
901 µs ± 17.5 µs per loop (mean ± std. dev. of 7 runs, 1000 loops each)

In [254]: %%timeit
     ...: df3['length'] = df3['app_version'].apply(lambda x: len(x))
     ...:
     ...:
1.16 ms ± 14.7 µs per loop (mean ± std. dev. of 7 runs, 1000 loops each)
```

我们看到在这里就体现出了矢量化的优势，因为这里大家都是一步。

结论：当矢量化步数只有一步时，其性能还是要比apply方式好的，但当需要多步的时候，不一定好于apply方式。

那么，我们能否将其转化为一步呢？后发现有extract这样的函数，使用如下：

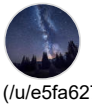
```
In [312]: df
Out[312]:
        app_version  uid
0  7.23.1-180522122    1
1  7.20.1-180502135    2
2  7.23.1-180522122    3
3  7.23.1-180522122    4
4  7.16.7-180411077    5

In [313]: %%timeit
     ...: df['app_version'] = df['app_version'].str.extract(r"([0-9\.]+)-[0-9]+", exp
     ...: se)
     ...:
247 µs ± 8.95 µs per loop (mean ± std. dev. of 7 runs, 1000 loops each)
```

通过 `extract` 终于实现了一步的矢量化。而且性能上也是最优的。

> 这里需要注意的是，如果使用timeit, 由于多次操作，会导致后续df中'app_version'的值变为NaN。当我们只操作一次的时候则不存在此问题。

References

- Working with text data (https://pandas.pydata.org/pandas-docs/stable/text.html)
- Working with strings (https://jakevdp.github.io/PythonDataScienceHandbook/03.10-working-with-strings.html)
- 优化Pandas代码执行速度入门指南 (https://python.freelycode.com/contribution/detail/1083)
- Pandas 中map, applymap and apply的区别 (https://blog.csdn.net/u010814042/article/details/76401133)



geekpy (/u/e5fa627c0613)

写了 95418 字，被 253 人关注，获得了 407 个喜欢

(/u/e5fa627c0613)

+关注

找工作中... 有职位信息，请联系life_0130@126.com