☂Author:chenshuo

作业内容

1. 读取 tweet 数据（AAL.xlsx → Stream → Tweet content）；
2. 计算每条 tweet 的四种情感得分；
3. 将得分作为新的数据列与 Stream 数据合并；
4. 选取任意连续 60 天的所有数据；
5. 将 Date 列转换为日期格式；
6. 将新的日期设置为 index；
7. 挑选出有价值的属性['Hour', 'Tweet content', 'Favs', 'RTs', 'Followers', 'Following', 'Is a RT', 'Hashtags', 'Symbols', 'compound', 'neg', 'neu', 'pos']；
8. 去掉 compound 的得分为 0（即中立）的数据；
9. 去掉 Followers 为空的数据；
10. 增加新的一列 Compound_multiplied，由 compound 和 Followers 相乘而来；
11. 将Compound_multiplied标准化，作为新的一列Compound_multiplied_scaled；
12. 计算每日所有数据的均值；
13. 将均值结果保存为 excel 表

抽取连续60天数据

1. 设置index，loc

可能的问题：抽取数据后df为空

In [6]:
```
# df_concat.sort_index(inplace = True)          #按时间升序

s_date = datetime.datetime(2016,4,1)          #s_date = '20160401' ; s_date = '2016-04-01'亦可
e_date = datetime.datetime(2016,5,30)

df_concat.loc[s_date:e_date,:]
```

Out[6]:

	Tweet Id	Hour	User Name	Nickname	Bio	Tweet content	Favs	RTs	Latitude	Longitude	...	Original Tweet User Name	User Mentions	Hashtags	Symbols	Media	URLs	compound	neg
Date																			

0 rows × 29 columns

问题原因：源文件时间降序

解决方法

- 改为升序

```
df_concat.sort_index(inplace = True)
```

- start_date与end_date对调

```
df_concat.loc[e_date:s_date,:]
```

2. Date column 索引

```
df_concat[(df_concat['Date'] >= '2016-04-01') & (df_concat['Date'] <= '2016-05-30')]
```

数据标准化

调用standardscaler(score)函数报错

问题原因

```
score = df_new['Compound_multiplied'].values
```

```
ValueError: Expected 2D array, got 1D array instead:
array=[  105.7965 -5202.496     814.0095 ...   190.5182    122.7961    439.4518].
Reshape your data either using array.reshape(-1, 1) if your data has a single feature or array.reshape(1, -1) if it contains a single sample.
```

```
In  [94]: df_new['Compound_multiplied'].values
Out[94]: array([  105.7965, -5202.496 ,   814.0095, ...,    190.5182,    122.7961,
                  439.4518])
```

```
In  [95]: df_new['Compound_multiplied'].values.shape
Out[95]: (1933,)
```

解决方法

- reshape(-1,1)

```
score = df_new['Compound_multiplied'].values.reshape(-1,1)
```

```
In [96]: df_new['Compound_multiplied'].values.reshape(-1,1)
Out[96]: array([[  105.7965],
                [-5202.496 ],
                [  814.0095],
                ...,
                [  190.5182],
                [  122.7961],
                [  439.4518]])
```

```
In [97]: df_new['Compound_multiplied'].values.reshape(-1,1).shape
Out[97]: (1933, 1)
```

- df[['column']]

```
score = df_new[['Compound_multiplied']].values
```

```
In [84]: df_new[['Compound_multiplied']].values
Out[84]: array([[  105.7965],
                [-5202.496 ],
                [  814.0095],
                ...,
                [  190.5182],
                [  122.7961],
                [  439.4518]])
```

```
In [98]: df_new[['Compound_multiplied']].values.shape
Out[98]: (1933, 1)
```

几种不同的标准化方法

StandardScale

- 原理

  $z=\frac{x-mean(x)}{std(x)}$

- 调用

```
def standard_scaler(score):
    scaler = StandardScaler().fit(score)   #先fit再transform，保存标准化转换器
    scaled_data = scaler.transform(score)
    #scaled_data = StandardScaler().fit_transform(x)   #更高效
    return scaled_data
```

- 实现

np.mean(x)

```
def standard_scaler_1(x):
    x = (x - np.mean(x)) / np.std(x)
    return x
```

实验

```
x = np.array([[-1,1],
             [-2,2],
             [-3,4]])
```

```
MaxAbsScaler().fit_transform(x)
array([[-0.33333333,  0.25      ],
       [-0.66666667,  0.5       ],
       [-1.        ,  1.        ]])
```

```
x / np.max(np.abs(x))
array([[-0.25,  0.25],
       [-0.5 ,  0.5 ],
       [-0.75,  1.  ]])
```

```
x / np.max(np.abs(x), axis = 0)
array([[-0.33333333,  0.25      ],
       [-0.66666667,  0.5       ],
       [-1.        ,  1.        ]])
```

np.mean(x,axis = 0)

```
def standard_scaler_1(x):          #实现StandardScaler
    x = (x - np.mean(x,axis = 0)) / np.std(x, axis = 0)
    return x
```

MinMaxScaler

- 原理

$z=\frac{x-min(x)}{max(x)-min(x)}$

- 调用

```
def min_max_scaler(x):               #MinMaxScaler
#     scaler = MinMaxScaler.fit(x)
#     scaled_data = scaler.transform(x)
    scaled_data = MinMaxScaler().fit_transform(x)
    return scaled_data             #实现MinMaxScaler
```

- 实现

```
def min_max_scaler_1(x):
    x = (x - np.min(x, axis = 0)) /(np.max(x, axis = 0) - np.min(x, axis = 0))
    return x
```

MaxAbsScaler

- 原理

$z=\frac{x}{max(|x|)}$

- 调用

```
def max_abs_scaler(x):          #MaxAbsScaler

    scaled_data = MaxAbsScaler().fit_transform(x)

#     scaler = preprocessing.MaxAbsScaler()
#     scaled_data = scaler.fit_transform(x)

    return scaled_data
```

- 实现

```
def max_abs_scaler_1(x):         #实现MaxAbsScaler
    x = x / np.max(x,axis = 0)
    return x
```

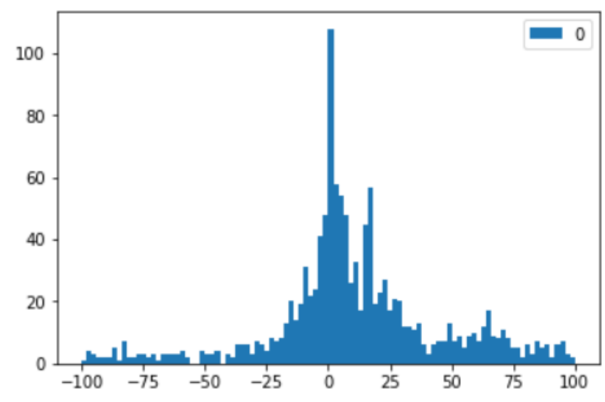
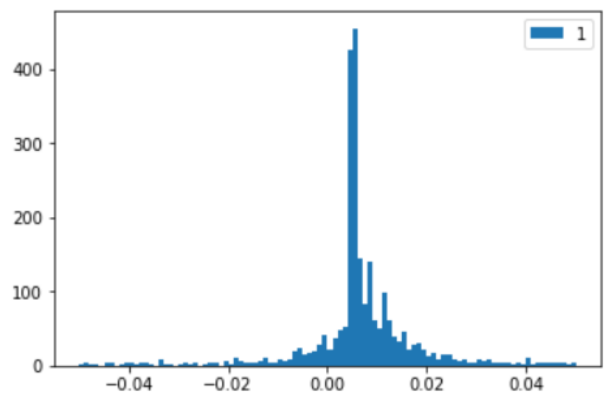
更多方法

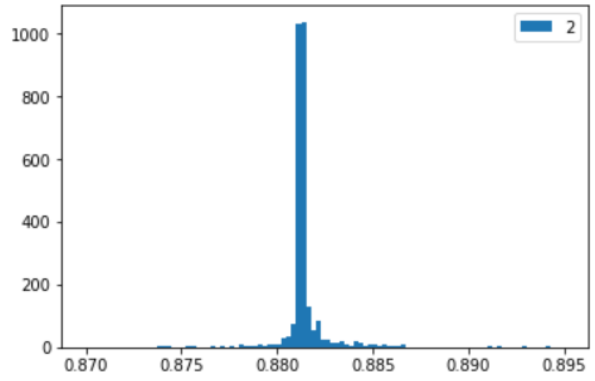
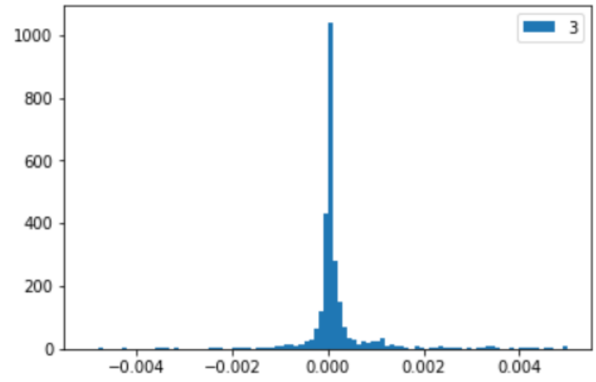
User guide: See the Preprocessing data section for further details.

`preprocessing.Binarizer` ([threshold, copy])	Binarize data (set feature values to 0 or 1) according to a threshold
`preprocessing.FunctionTransformer` ([func, …])	Constructs a transformer from an arbitrary callable.
`preprocessing.KBinsDiscretizer` ([n_bins, …])	Bin continuous data into intervals.
`preprocessing.KernelCenterer` ()	Center a kernel matrix
`preprocessing.LabelBinarizer` ([neg_label, …])	Binarize labels in a one-vs-all fashion
`preprocessing.LabelEncoder`	Encode labels with value between 0 and n_classes-1.
`preprocessing.MultiLabelBinarizer` ([classes, …])	Transform between iterable of iterables and a multilabel format
`preprocessing.MaxAbsScaler` ([copy])	Scale each feature by its maximum absolute value.
`preprocessing.MinMaxScaler` ([feature_range, copy])	Transforms features by scaling each feature to a given range.
`preprocessing.Normalizer` ([norm, copy])	Normalize samples individually to unit norm.
`preprocessing.OneHotEncoder` ([n_values, …])	Encode categorical integer features as a one-hot numeric array.
`preprocessing.OrdinalEncoder` ([categories, dtype])	Encode categorical features as an integer array.
`preprocessing.PolynomialFeatures` ([degree, …])	Generate polynomial and interaction features.
`preprocessing.PowerTransformer` ([method, …])	Apply a power transform featurewise to make data more Gaussian-like.
`preprocessing.QuantileTransformer` ([…])	Transform features using quantiles information.
`preprocessing.RobustScaler` ([with_centering, …])	Scale features using statistics that are robust to outliers.
`preprocessing.StandardScaler` ([copy, …])	Standardize features by removing the mean and scaling to unit variance
`preprocessing.add_dummy_feature` (X[, value])	Augment dataset with an additional dummy feature.
`preprocessing.binarize` (X[, threshold, copy])	Boolean thresholding of array-like or scipy.sparse matrix
`preprocessing.label_binarize` (y, classes[, …])	Binarize labels in a one-vs-all fashion
`preprocessing.maxabs_scale` (X[, axis, copy])	Scale each feature to the [-1, 1] range without breaking the sparsity.
`preprocessing.minmax_scale` (X[, …])	Transforms features by scaling each feature to a given range.
`preprocessing.normalize` (X[, norm, axis, …])	Scale input vectors individually to unit norm (vector length).
`preprocessing.quantile_transform` (X[, axis, …])	Transform features using quantiles information.
`preprocessing.robust_scale` (X[, axis, …])	Standardize a dataset along any axis
`preprocessing.scale` (X[, axis, with_mean, …])	Standardize a dataset along any axis
`preprocessing.power_transform` (X[, method, …])	Power transforms are a family of parametric, monotonic transformations that are applied to make data more Gaussian-like.

不同标准化方法处理数据后的结果

一维

Compound_multiplied	StandardScale
	
MinMaxScaler	**MaxAbsScaler**

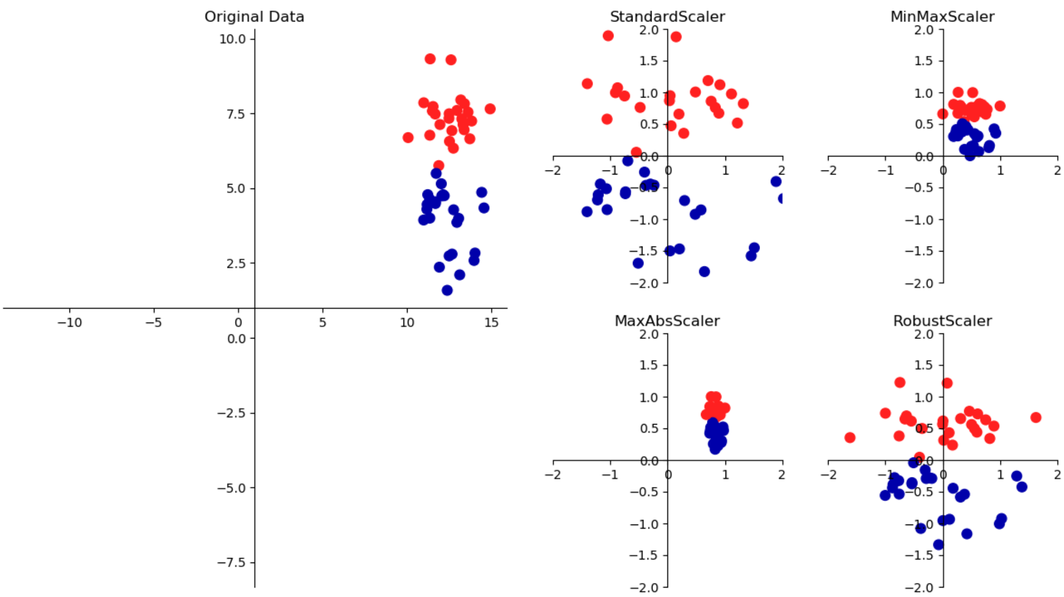
Compound_multiplied	StandardScale
	

二维

- 生成数据

```
X, y = make_blobs(n_samples=50, centers=2, random_state=4, cluster_std=1)
```

- 结果分析

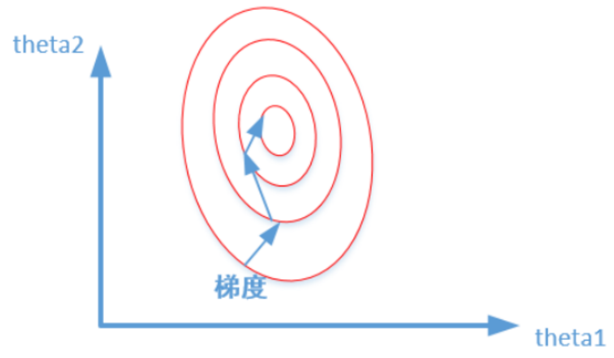


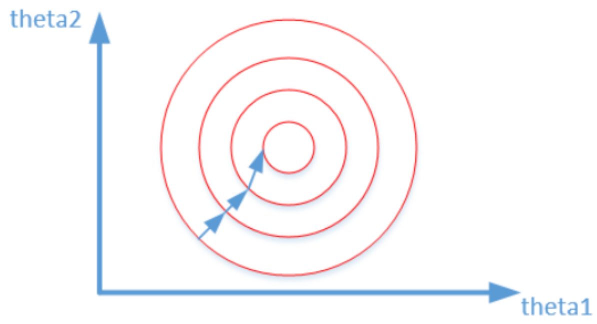
数据标准化的作用

- 无量纲化，便于不同评价指标的比较
- 弱化奇异值对模型的影响
- 加快梯度下降求最优解的速度

未归一化	归一化

未归一化	归一化
$J = (3\theta_{1} + 600\theta_{2} - y_{correct})^{2}$	$(0.5\theta_{1}+0.55\theta_{2}-y_{correct})^{2}$





参考资料

> 《python机器学习基础教程》——人民邮电出版社
> 知乎：https://zhuanlan.zhihu.com/p/27627299