机器学习笔记

cuihu

2020 年 9 月 10 日

摘要

菜菜 sklearn 笔记

1 PCA 与 SVD 降维算法

我们希望能够找出一种办法来帮助我们衡量特征上所带的信息量，让我们在降维的过程中，能够即减少特征的数量，又保留大部分有效信息——将那些带有重复信息的特征合并，并删除那些带无效信息的特征等等——逐渐创造出能够代表原特征矩阵大部分信息的，特征更少的，新特征矩阵。
从方差的这种应用就可以推断出，如果一个特征的方差很大，则说明这个特征上带有大量的信息。因此，在降维中，**使用的信息量衡量指标，就是样本方差，又称可解释性方差，方差越大，特征所带的信息量越多。**

$$Var = \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \hat{x})^2$$

Var 代表一个特征的方差，n 代表样本量，xi 代表一个特征中的每个样本取值，xhat 代表这一列样本的均值。

1.1 实现

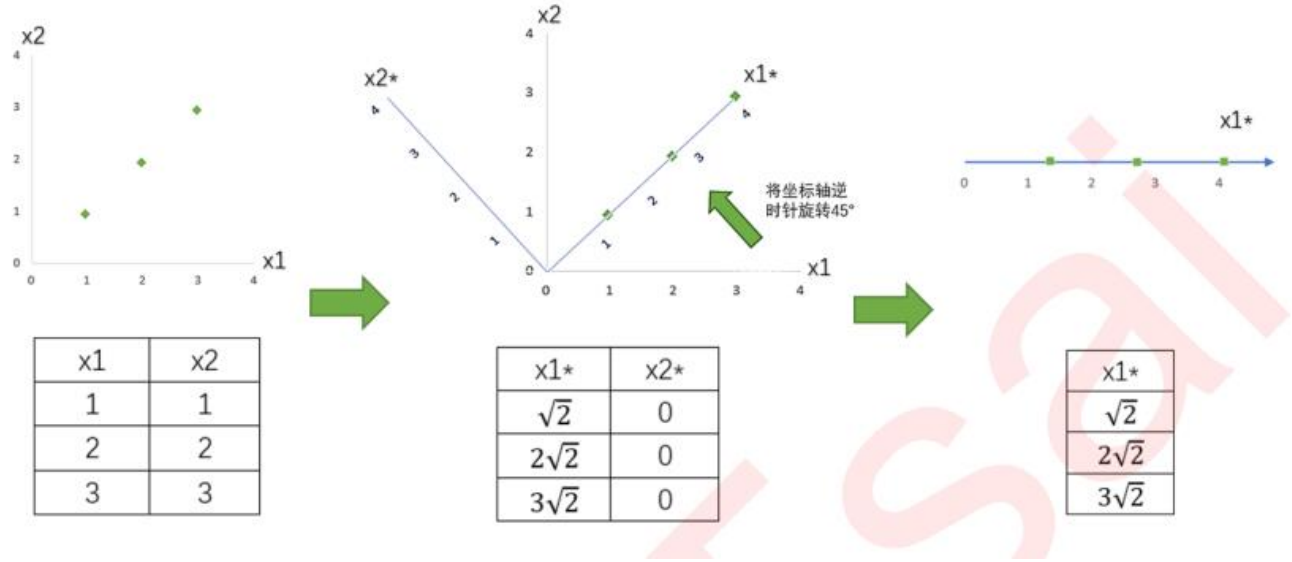
```
class sklearn.decomposition.PCA (
n_components=None,
copy=True,
whiten=False,
svd_solver='auto',
tol=0.0,
iterated_power='auto',
random_state=None)
```

找出 n 个新特征向量，让数据能够被压缩到少数特征上并且总信息量不损失太多的技术就是矩阵分解。
PCA 使用方差作为信息量的衡量指标，并且特征值分解来找出空间 V。降维时，它会通过一系列数学的神秘操作（比如说，产生协方差矩阵）将特征矩阵 X 分解为以下三个矩阵，其中和是辅助的矩阵，Σ 是一个对角矩阵（即除了对角线上有值，其他位置都是 0 的矩阵），其对角线上的元素就是方差。降维完成之后，PCA 找到的每个新特征向量就叫做“主成分”，而被丢弃的特征向量被认为信息量很少，这些信息很可能就是噪音。

$$X \to Q\Sigma Q^{-1} \tag{1}$$

而 SVD 使用奇异值分解来找出空间 V，其中 Σ 也是一个对角矩阵，不过它对角线上的元素是奇异值，这也是 SVD 中用来衡量特征上的信息量的指标。U和V^{T}分别是左奇异矩阵和右奇异矩阵，也都是辅助矩阵。

$$X \to U\Sigma V^T$$



x1	x2
1	1
2	2
3	3

x1*	x2*
$\sqrt{2}$	0
$2\sqrt{2}$	0
$3\sqrt{2}$	0

x1*
$\sqrt{2}$
$2\sqrt{2}$
$3\sqrt{2}$

图 1: 二维降一维

1.2 思考：PCA 和特征选择技术都是特征工程的一部分，它们有什么不同？

特征工程中有三种方式：特征提取，特征创造和特征选择。仔细观察上面的降维例子和上周我们讲解过的特征选择，你发现有什么不同了吗?

特征选择是从已存在的特征中选取携带信息最多的，选完之后的特征依然具有可解释性，我们依然知道这个特征在原数据的哪个位置，代表着原数据上的什么含义。

而 PCA，是将已存在的特征进行压缩，降维完毕后的特征不是原本的特征矩阵中的任何一个特征，而是通过某些方式组合起来的新特征。通常来说，在新的特征矩阵生成之前，我们无法知晓 PCA 都建立了怎样的新特征向量，新特征矩阵生成之后也不具有可读性，我们无法判断新特征矩阵的特征是从原数据中的什么特征组合而来，新特征虽然带有原始数据的信息，却已经不是原数据上代表着的含义了。**以 PCA 为代表的降维算法因此是特征创造**（feature creation，**或** feature construction）**的一种。**

可以想见，PCA 一般不适用于探索特征和标签之间的关系的模型（如线性回归），因为无法解释的新特征和标签之间的关系不具有意义。在线性回归模型中，我们使用特征选择。

1.3 参数

```
class sklearn.decomposition.PCA (
n_components=None,
copy=True,
whiten=False,
svd_solver='auto',
tol=0.0,
iterated_power='auto',
random_state=None)
```

1.3.1 重要参数 n_components

该参数就是我们需要降维的目标个数，如果我们需要可视化一组特征，那么降低到二维，以下就是例子。：

```
# 调用模块
import matplotlib.pyplot as plt
```

```
from sklearn.datasets import load_iris # 数据集。
from sklearn.decomposition import PCA # PCA

# 提取数据集
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from sklearn.decomposition import PCA

# 提取数据集。
iris = load_iris()
y = iris.target
x = iris.data



# 作为数据表或特征矩阵，x 是几维度。

print(x.shape)

import pandas as pd
df = pd.DataFrame(x)    #（150，4）
# print(df)# 可以看出一共有四个特征，150行。

# 建模
pca = PCA(n_components=2)
# 实例化。
pca = pca.fit(X=x)
x_dr = pca.transform(x)
print(x_dr.shape) # 从思维转换到了二维。  （150，2）

# 可视化。
x_dr[y == 0, 0]
# 第一种话花的索引。
# 标签的为 0 1 2
# plt.figure()
# plt.scatter(x_dr[y==0, 0], x_dr[y==0, 1], c="red", label=iris.target_names[0])
# plt.scatter(x_dr[y==1, 0], x_dr[y==1, 1], c="black", label=iris.target_names[1])
# plt.scatter(x_dr[y==2, 0], x_dr[y==2, 1], c="orange", label=iris.target_names[2]
# plt.legend()
# plt.title('PCA of IRIS dataset')
# plt.show()

colors = ['red', 'black', 'orange']
iris.target_names
plt.figure()
for i in [0, 1, 2]:
```

```
plt.scatter(x_dr[y == i, 0],
x_dr[y == i, 1],
alpha=.7,
c=colors[i],
label=iris.target_names[i])
plt.legend()
plt.title('PCA of IRIS dataset')
plt.show()


#探索降维后的数据
# 属性explained_variance_，查看降维后每个特征向量上所带的信息量大小。（可解释方差大
print(pca.explained_variance_)  # 可解释方差大小。

# 可解释方差贡献率。也就是新的特征向量对原始特征的信息量的贡献率。 [4.22824171  0.2
print(pca.explained_variance_ratio_)    # [0.92461872  0.05306648]

# 大部分信息被集中在了第一维。

print(pca.explained_variance_ratio_.sum())  #0.9776852063187951

# 选择最好的n_components：累积可解释方差贡献率曲线
```

1.3.2 选择最好的 n_components: 累计可解释方差贡献率曲线

当参数 n_components 中不填写任何值，则默认返回 min(X.shape) 个特征，一般来说，样本量都会大于特征数目，所以什么都不填就相当于转换了新特征空间，但没有减少特征的个数。一般来说，不会使用这种输入方式。但我们却可以使用这种输入方式来画出累计可解释方差贡献率曲线，以此选择最好的 n_components 的整数取值。
累积可解释方差贡献率曲线是一条以降维后保留的特征个数为横坐标，降维后新特征矩阵捕捉到的可解释方差贡献率为纵坐标的曲线，能够帮助我们决定 n_components 最好的取值。

```
import numpy as np
pca_line = PCA().fit(x)  #没有参数。
plt.plot([1,2,3,4], np.cumsum(pca_line.explained_variance_ratio_))
plt.xticks([1,2,3,4])
plt.grid()
plt.xlabel('number of components after dimension reduction')
plt.ylabel('cumulative explained variance ratio')
plt.show()
```

1.3.3 最大似然估计选择超参数

除了输入整数，n_components 还有哪些选择呢？之前我们提到过，矩阵分解的理论发展在业界独树一帜，勤奋智慧的数学大神 Minka, T.P. 在麻省理工学院媒体实验室做研究时找出了让 PCA 用最大似然估计 (maximum likelihood estimation) 自选超参数的方法，输入“mle”作为 n_components 的参数输入，就可以调用这种方法。

```
pca_mle = PCA(n_components='mle')
pca_mle = pca_mle.fit(x)
```

```
x_mle = pca_mle.transform(x)

print(x_mle.shape)  # (150, 3)

print(pca_mle.explained_variance_ratio_.sum())  # 0.9947878161267248
```

1.3.4 按信息量占比选超参数

输入 [0,1] 之间的浮点数，并且让参数 svd_solver =='full'，表示希望降维后的总解释性方差占比大于 n_components 指定的百分比，即是说，希望保留百分之多少的信息量。比如说，如果我们希望保留 97% 的信息量，就可以输入 n_components = 0.97，PCA 会自动选出能够让保留的信息量超过 97% 的特征数量。

```
pca_f = PCA(n_components=0.97, svd_solver='full').fit(x)  # 大于百分之97即
x_f = pca_f.transform(x)
print(pca_f.explained_variance_ratio_)
print(x_f.shape)
```

1.4 PCA 中的 SVD

1.4.1 PCA 中的 SVD 哪里来？