PCA实战

01 梯度上升法求解PCA

我们在用最大投影方差求解PCA的时候，我们得到了：

$$\mathrm{Var}(X_{\mathrm{project}}) = \frac{1}{m}\sum_{i=1}^{m}\left(\sum_{j=1}^{n}X_j^{(i)}w_j\right)^2$$

我们的目标是求w使得方差最大化，就转换成了求目标函数的最优化问题，可以用梯度上升法进行求解。

$$\nabla f = \begin{pmatrix}\frac{\partial f}{\partial w_1}\\ \frac{\partial f}{\partial w_2}\\ \cdots\\ \frac{\partial f}{\partial w_n}\end{pmatrix} = \frac{2}{m}\begin{pmatrix}\sum_{i=1}^{m}\left(X_1^{(i)}w_1 + X_2^{(i)}w_2 + \ldots + X_n^{(i)}w_n\right)X_1^{(i)}\\ \sum_{i=1}^{m}\left(X_1^{(i)}w_1 + X_2^{(i)}w_2 + \ldots + X_n^{(i)}w_n\right)X_2^{(i)}\\ \sum_{i=1}^{m}\left(X_1^{(i)}w_1 + X_2^{(i)}w_2 + \ldots + X_n^{(i)}w_n\right)X_n^{(i)}\end{pmatrix} = \frac{2}{m}\begin{pmatrix}\sum_{i=1}^{m}\left(X^{(i)}w\right)X_1^{(i)}\\ \sum_{i=1}^{m}\left(X^{(i)}w\right)X_2^{(i)}\\ \sum_{i=1}^{m}\left(X^{(i)}w\right)X_n^{(i)}\end{pmatrix}$$

化简后得到：

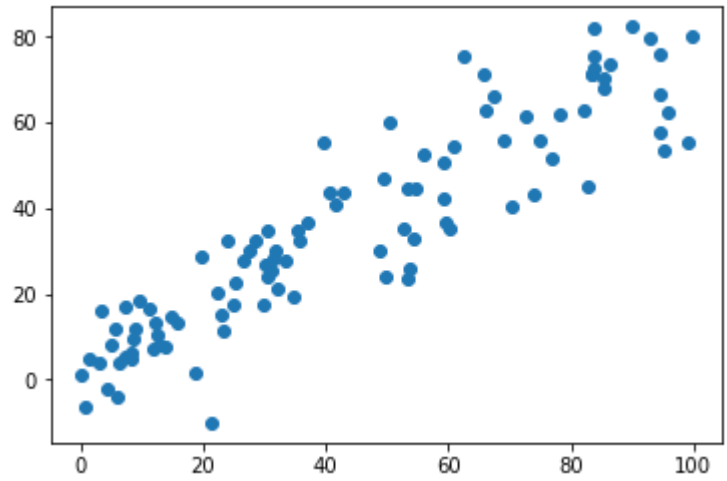
$$\nabla f = \frac{2}{m}\begin{pmatrix}\sum_{i=1}^{m}\left(X^{(i)}w\right)X_1^{(i)}\\ \sum_{i=1}^{m}\left(X^{(i)}w\right)X_2^{(i)}\\ \cdots\\ \sum_{i=1}^{m}\left(X^{(i)}w\right)X_n^{(i)}\end{pmatrix} = \frac{2}{m}\cdot X^T(Xw)$$

梯度上升法的思路和梯度下降法的思路完全一样，只是前者求极大值，后者求极小值。

接下来我们使用代码来实现：

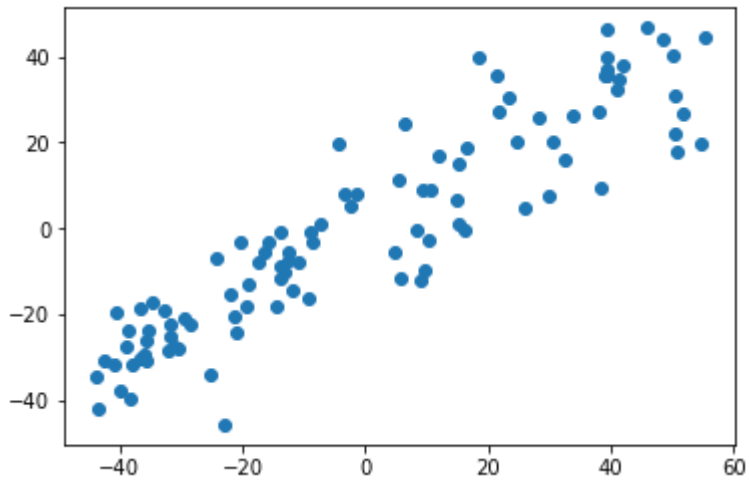
创建具有线性相关的数据：

```
X = np.empty((100, 2))
X[:,0] = np.random.uniform(0., 100., size=100)
X[:,1] = 0.75 * X[:,0] + 3. + np.random.normal(0, 10., size=100)
```



进行中心化：

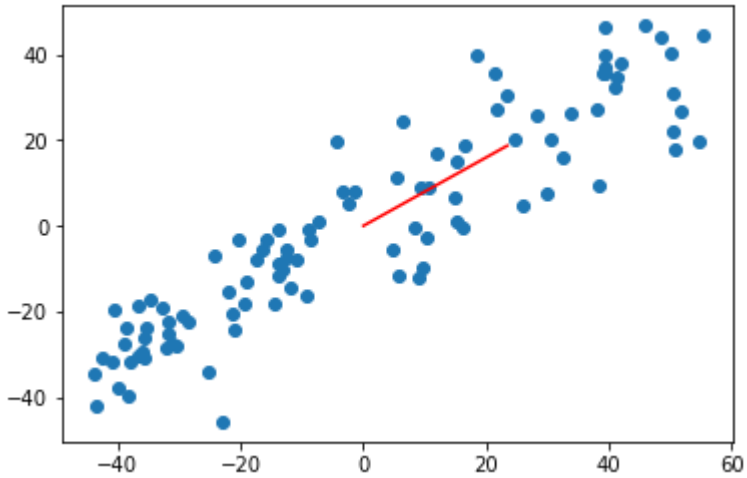
```
def demean(X):
    return X - np.mean(X, axis=0)
```



使用梯度上升法：

```
def gradient_ascent(df, X, initial_w, eta, n_iters = 1e4, epsilon=1e-8):

    w = direction(initial_w)
    cur_iter = 0

    while cur_iter < n_iters:
        gradient = df(w, X)
        last_w = w
        w = w + eta * gradient
        w = direction(w) # 注意1：每次求一个单位方向
        if(abs(f(w, X) - f(last_w, X)) < epsilon):
            break

        cur_iter += 1

    return w
```

画出对应的主轴为：

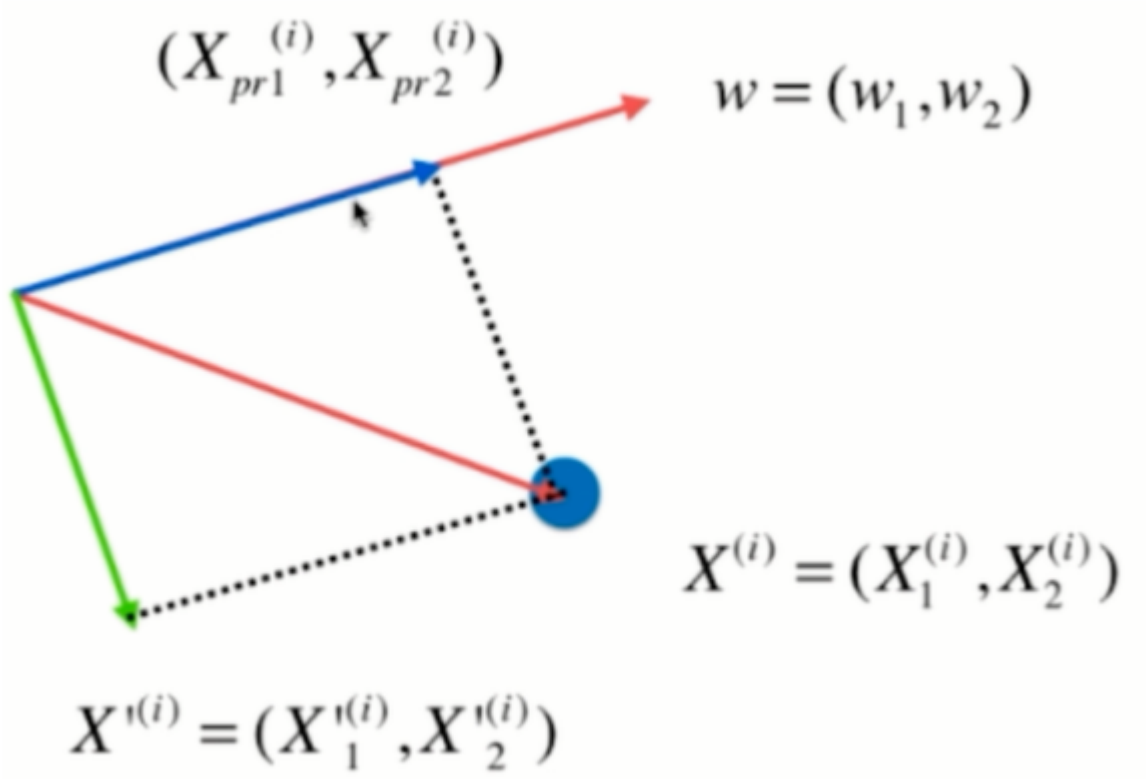


02 获得前n个主成分

我们先考虑如何计算第一个主成分：还是使用梯度上升法

```
def first_component(X, initial_w, eta, n_iters = 1e4, epsilon=1e-8):

    w = direction(initial_w)
    cur_iter = 0

    while cur_iter < n_iters:
        gradient = df(w, X)
        last_w = w
        w = w + eta * gradient
        w = direction(w)
        if(abs(f(w, X) - f(last_w, X)) < epsilon):
            break

        cur_iter += 1

    return w
```

在上面，对于只有2维特征的样本x，我们求解出了第一个主成分，即w=（w1,w2），为了求解第二个主成分，这里需要将样本x在第一个主成分上的分量去除掉，这里使用的方法即空间几何的向量减法，得到的结果即下图中的绿线部分。



即$X^{(i)}$减去$X^{(i)}$在w方向上的分量$\left(X_{pr1}^{(i)}, X_{pr2}^{(i)}\right)$，得到新的样本数据$X'^{(i)} = \left(X_1'^{(i)}, X_2'^{(i)}\right)$

对应的代码实战为：

```
def first_n_components(n, X, eta=0.01, n_iters = 1e4, epsilon=1e-8):
    X_pca = X.copy()
    X_pca = demean(X_pca)
    res = []
    for i in range(n):
        initial_w = np.random.random(X_pca.shape[1])
        w = first_component(X_pca, initial_w, eta)
        res.append(w)

        X_pca = X_pca - X_pca.dot(w).reshape(-1, 1) * w

    return res
```

03 从高维数据向低维数据的映射

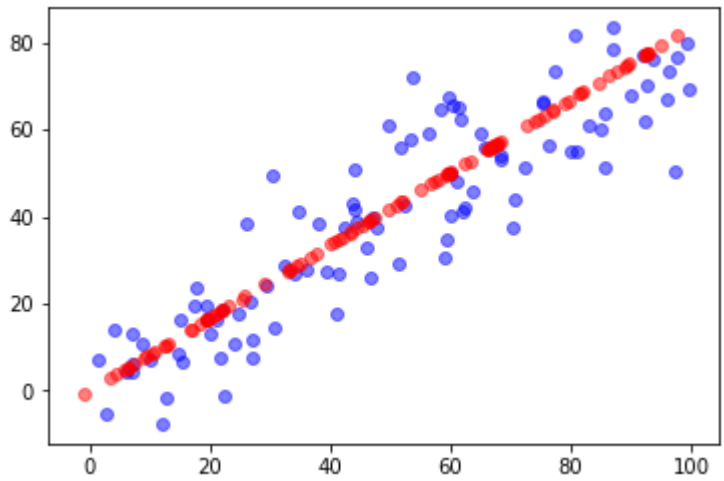
我们现在用封装好的PCA代码来跑一下：

```
from playML.PCA import PCA

pca = PCA(n_components=2)
pca.fit(X)
```

对应的主成分为：

```
array([[ 0.76676948,  0.64192256],
       [-0.64191827,  0.76677307]])
```

我们可以将样本数据和经过fit_transform降维后的数据展现在同一张图上：



scikit-learn中的PCA跟上面的使用方法一样。

04 scikit-learn中的PCA

那么我们来具体说一说scikit-learn中的PCA实战，看看经过PCA处理后的数据对knn算法有多少提升。

我们使用datasets中的手写识别数据集。首先我们使用knn算法跑结果看看：

```
from sklearn.neighbors import KNeighborsClassifier

knn_clf = KNeighborsClassifier()
knn_clf.fit(X_train, y_train)
knn_clf.score(X_test, y_test)
```

平均精度的结果为：

```
0.98666666666666669
```

然后我们使用PCA对数据进行降维：

```
from sklearn.decomposition import PCA

pca = PCA(n_components=2)
pca.fit(X_train)
X_train_reduction = pca.transform(X_train)
X_test_reduction = pca.transform(X_test)
```

最后再进行打分，得分为：

```
0.60666666666666669
```

可以发现，经过降维后，因为维度就剩下2个，得出的分值很低，为了提高分值，应该保留更多的特征值。但是具体保留多少特征合适？有什么衡量指标可以作为参考？

sklearn-PCA中有个变量为pca.explained_variance_ratio_，可以用来衡量映射后的数据保留了原始数据的多少信息（方差）。对于上面求得的结果，有如下的值：

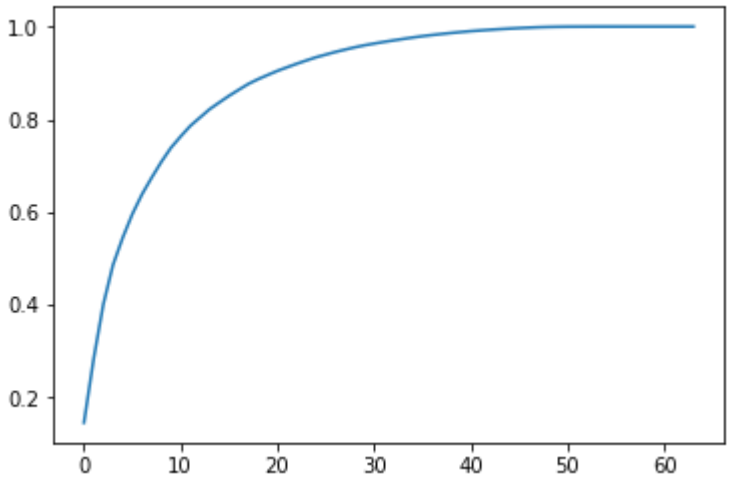
```
array([ 0.14566817,  0.13735469])
```

这表示：经过映射后的数据，只保留原始数据大约百分之（14+13=27）的方差，丢掉了原来约73的方差，这个损失是比较大的，所以得出来的分数很低。

现在的问题是，应该保留多少的维度使得信息丢失在允许法范围内。假设选取的主成分的个数和原始样本特征维度一样，由于explained_variance_ratio*是按照大小顺序排列，这时候观察explained_variance_ratio*发现，越到后面，值越小，甚至几乎可以忽略。

```
array([  1.45668166e-01,   1.37354688e-01,   1.17777287e-01,
         8.49968861e-02,   5.86018996e-02,   5.11542945e-02,
         4.26605279e-02,   3.60119663e-02,   3.41105814e-02,
         3.05407804e-02,   2.42337671e-02,   2.28700570e-02,
         1.80304649e-02,   1.79346003e-02,   1.45798298e-02,
         1.42044841e-02,   1.29961033e-02,   1.26617002e-02,
         1.01728635e-02,   9.09314698e-03,   8.85220461e-03,
         7.73828332e-03,   7.60516219e-03,   7.11864860e-03,
         6.85977267e-03,   5.76411920e-03,   5.71688020e-03,
         5.08255707e-03,   4.89020776e-03,   4.34888085e-03,
         3.72917505e-03,   3.57755036e-03,   3.26989470e-03,
         3.14917937e-03,   3.09269839e-03,   2.87619649e-03,
         2.50362666e-03,   2.25417403e-03,   2.20030857e-03,
         1.98028746e-03,   1.88195578e-03,   1.52769283e-03,
         1.42823692e-03,   1.38003340e-03,   1.17572392e-03,
         1.07377463e-03,   9.55152460e-04,   9.00017642e-04,
         5.79162563e-04,   3.82793717e-04,   2.38328586e-04,
         8.40132221e-05,   5.60545588e-05,   5.48538930e-05,
         1.08077650e-05,   4.01354717e-06,   1.23186515e-06,
         1.05783059e-06,   6.06659094e-07,   5.86686040e-07,
         7.44075955e-34,   7.44075955e-34,   7.44075955e-34,
         7.15189459e-34])
```

现在将这些值从前往后依次进行累加，并在坐标轴上展现。



观察该图可以发现，当横轴为0时，纵轴也为0（丢失了所有信息），当横轴为特征值大小时，纵轴为1（保留了全部信息）。因此，比如想要保留95%的信息，就可以通过该图知道应该保留多少个特征维度。

```
pca = PCA(0.95)
pca.fit(X_train)
```

结果为：0.97999999999999998。

值明显得到了提高。而且如果进行时间测试，会发现时间缩短了很多。这就是pca的作用。

之前已经说过，pca经过降维可以用于可视化。将高维度难以想象的数据，降为2维可以在平面上展示。手写识别数据降为2维的图像如下。

```
pca = PCA(n_components=2)
pca.fit(X)
X_reduction = pca.transform(X)
for i in range(10):
    plt.scatter(X_reduction[y==i,0], X_reduction[y==i,1], alpha=0.8)
plt.show()
```