1　主成分分析とは

・次元圧縮の手法である

用途１：100 次元のデータを 3 次元に圧縮して可視データとして分析する

用途２：多次元の説明変数を圧縮してモデル計算量を減らす

次元削除するので当然情報量は落ちるがそれで充分な場合に有用

$\rightarrow$ 主成分分析では次元毎の寄与率 (データ全体に対する圧縮後データの情報量) がわかる

$\rightarrow$ 圧縮度合いが適切か否か客観的な判断が可能

・分析可能な主成分の数は説明変数の次元数と一致 (二次ならば２つの主成分を持つ)

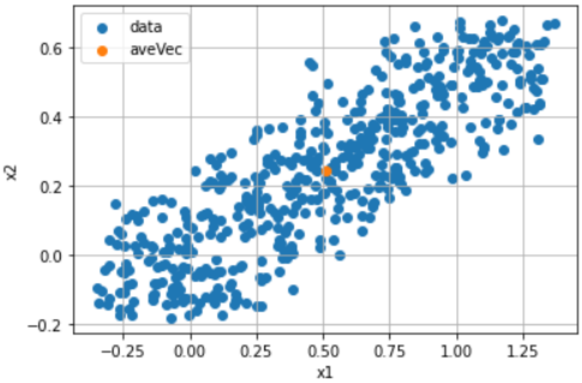
・元のデータに対する主成分のベクトルは元データの分散共分散行列より導出される固有ベクトルと一致

・主成分の情報量割合 (寄与率) は元データの分散共分散行列より導出される固有値の割合と一致

2 主成分分析の処理 (理解の為、一部 python コード作成)

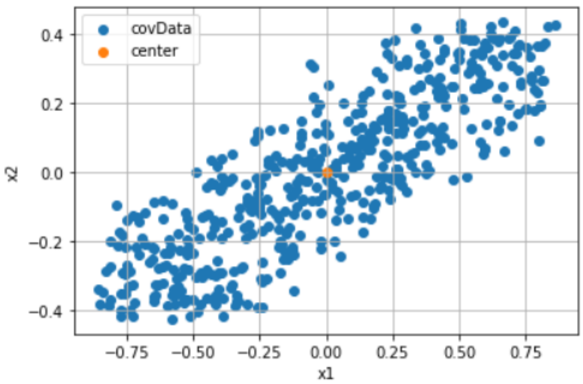
・data 生成

```
import numpy as np
import matplotlib.pyplot as plt
import random
%matplotlib inline

#主成分分析のコードを作ってみる
#第一主成分として見えそうなデータがx2 = 0.5 * x1になるように加工
dsize = 500
dt = np.array([i for i in range(dsize)])

#変数は適当にノイズを印可
data = np.zeros(dsize*2).reshape(2,500)
for item in dt:
    coef = (dsize - abs((item + 1) - (dsize/2)))/1.3
    if item%2 == 0:coef *= -1
    data[0][item]=(item + (random.random()*coef))/dsize
    data[1][item]=(item + (random.random()*coef))/(dsize * 2)

#平均ベクトルを算出(データの中心)
aveVec=np.array([np.average(data[0]), np.average(data[1])])
print(aveVec)

plt.grid()
plt.xlabel("x1")
plt.ylabel("x2")
plt.scatter(data[0],data[1],label='data')
plt.scatter(aveVec[0],aveVec[1],label='aveVec')
plt.legend()

✓ 0.1s
[0.50768241 0.24323452]
<matplotlib.legend.Legend at 0x1d469e368e0>
```



・データ行列化 (中心を 0 座標へ)

```
#平均ベクトルが(0,0)に配置される様にoffsetさせる(データ行列)
data2 = np.zeros(dsize*2).reshape(2,500)
for i in range(dsize):
    data2[0][i] = data[0][i] - aveVec[0]
    data2[1][i] = data[1][i] - aveVec[1]

aveVec2=np.array([np.average(data2[0]), np.average(data2[1])])
print(aveVec2)

plt.grid()
plt.xlabel("x1")
plt.ylabel("x2")
plt.scatter(data2[0],data2[1],label='covData')
plt.scatter(aveVec2[0],aveVec2[1],label='center')
plt.legend()

✓ 0.1s

[0.00000000e+00 1.42108547e-17]

<matplotlib.legend.Legend at 0x1d469eb99a0>
```

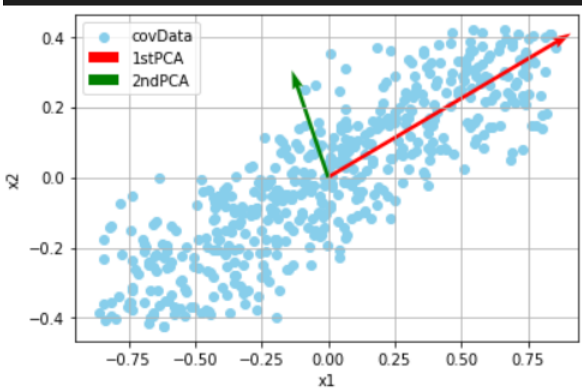


・分散共分散行列算出 ($Var(\bar{X})$)、固有ベクトル/固有値の算出、主成分観察
(実際の主成分はノルム 1 になるように制約をつける (後述))

```
#分散共分散行列を計算してみる
var = np.zeros(4).reshape(2,2)
for i in range(dsize):
    var[0][0] += data2[0][i]*data2[0][i]
    var[0][1] += data2[0][i]*data2[1][i]
    var[1][0] += data2[1][i]*data2[0][i]
    var[1][1] += data2[1][i]*data2[1][i]
var /= dsize

#分散共分散行列より固有値と固有ベクトルを取得する
#固有値と固有ベクトルについてはstage1で学習済なので説明省略
l, v = np.linalg.eig(var)
print("固有値{}".format(l))
print("固有ベクトル{}".format(v))

#とりあえず主成分のベクトル(=固有ベクトルと同じ向きのはず)を見てみる
#それらしく見えているかな
plt.grid()
plt.xlabel("x1")
plt.ylabel("x2")
plt.scatter(data2[0],data2[1],label='covData',color='skyblue')
plt.quiver(aveVec2[0],aveVec2[1],v[0][0],v[1][0],
        angles='xy',scale_units='xy',scale=1,label='1stPCA',color='red')
plt.quiver(aveVec2[0],aveVec2[1],v[0][1],v[1][1],
        angles='xy',scale_units='xy',scale=3,label='2ndPCA',color='green')
plt.legend()

#寄与率の計算
print("寄与率(第一主成分):{}".format(l[0] /(l[0]+l[1])))
print("寄与率(第二主成分):{}".format(l[1] /(l[0]+l[1])))
```

✓ 0.2s

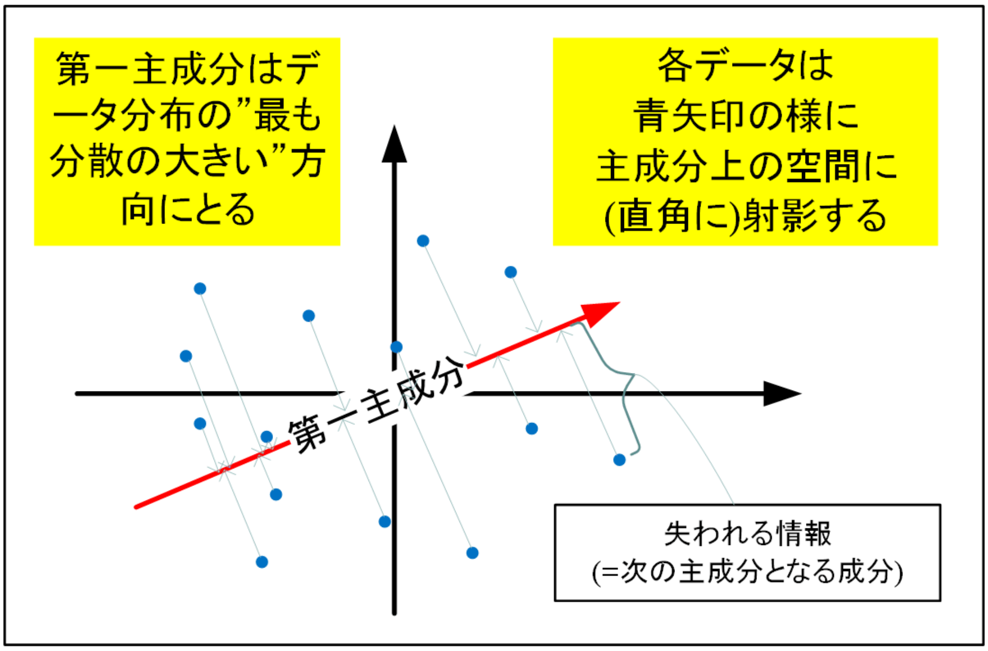
```
固有値[0.22776919 0.01069099]
固有ベクトル[[ 0.91248733 -0.40910496]
 [ 0.40910496  0.91248733]]
寄与率(第一主成分):0.955166555575661
寄与率(第二主成分):0.04483344442433899
```

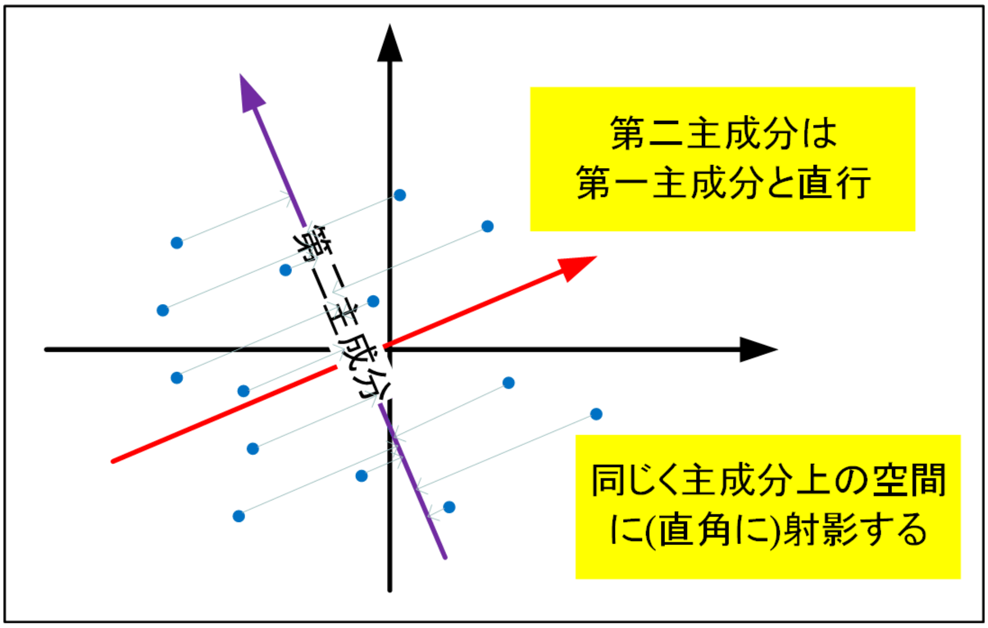


・主成分に対してデータが写像される

(講義資料における射影ベクトル $s_j = (s_{1j}, \cdots, s_{nj})^T = \bar{\mathbf{X}}\mathbf{a}_j$ の分散共分散行列をとり、

分散最大となる様な $\mathbf{a}_j$(ただしノルム 1 となる様に制約をつける) を求める ($\bar{\mathbf{X}}$ はデータ行列))





・s_j の分散共分散行列は以下

$$Var(\bar{X}) = \frac{1}{n}s_j^T s_j = \frac{1}{n}(\bar{\mathbf{X}}\mathbf{a}_j)^T(\bar{\mathbf{X}}\mathbf{a}_j) = \frac{1}{n}\mathbf{a}_j^T\bar{\mathbf{X}}^T\bar{\mathbf{X}}\mathbf{a}_j = \mathbf{a}_j^T Var(\bar{\mathbf{X}})\mathbf{a}_j$$

・上記に対してノルムが 1 となる制約を入れる (ラグランジュ関数による制約つき最適化問題)

$$E(\mathbf{a}_j) = \mathbf{a}_j^T Var(\bar{\mathbf{X}})\mathbf{a}_j - \lambda(\mathbf{a}_j^T\mathbf{a}_j - 1)$$

・ラグランジュ関数を最大とする (=微分して 0 になる) 様な $\mathbf{a}_j$ を探索

$\frac{\partial E(\mathbf{a}_j)}{\partial \mathbf{a}_j} = 2Var(\bar{\mathbf{X}})\mathbf{a}_j - 2\lambda\mathbf{a}_j = 0$ より

$Var(\bar{\mathbf{X}})\mathbf{a}_j = \lambda\mathbf{a}_j$ となり、

$\mathbf{a}_j$ は元の分散共分散行列の固有ベクトル一致し、λ は固有値と一致する事になる

・また以下より射影先の分散は固有値と一致する事になる

$$Var(\mathbf{s}_1) = \mathbf{a}_1^T Var(\bar{\mathbf{X}})\mathbf{a}_1 = \lambda_1\mathbf{a}_1^T\mathbf{a}_1 = \lambda_1$$

3 寄与率について

・主成分の情報量の割合を示す

・第一主成分は元のデータの分散と一致する

・第二主成分以降は上の次元における固有値と一致する (上の項で証明した通り)

・寄与率は分散共分散行列の固有値より求まる

m 次元データにおける第 k 主成分の寄与率は $C_k = \frac{\lambda_k}{\sum_{i=1}^m \lambda_i}$

m 次元データにおける第 k 主成分までの累積寄与率は $r_k = \frac{\sum_{i=1}^k \lambda_i}{\sum_{i=1}^m \lambda_i}$

4 実習

ハンズオンコードに対して以下 Cell を追加 (次元圧縮による精度変化を体感)

とりあえず自分なりに実習をする(PCAで圧縮したものがモデルの精度にどう影響するか体感)

```
#N次元に圧縮したうえでロジスティック回帰して精度を見てみる
#説明変数全体をStandardScaleしてから主成分分析でN次元に圧縮して
#そこから訓練データとテストデータに分けて検証すれば良いかな?

# 標準化
scaler_pcaN = StandardScaler()
X_scaled = scaler_pcaN.fit_transform(X)

#主成分分析(N次元)=>ロジスティック回帰
#Nは"TOP: TOTAL NO. of ITERATIONS REACHED LIMIT."しない程度に設定
#精度を追求しないならば「5次元程度でも十分戦えそう」
#精度を追求したとしても「13次元で十分かな?」
#という「感想」に至った
for n in range(1,14,1):
  #PCA(N次元)
  print('-----PCA{}-----'.format(n))
  pcaN = PCA(n_components=n)
  X_pcaN = pcaN.fit_transform(X_scaled)
  print(X_pcaN.shape)

  # 学習用とテスト用でデータを分離
  X_train_pcaN, X_test_pcaN, y_train_pcaN, y_test_pcaN = train_test_split(X_pcaN, y, random_state=0)

  # ロジスティック回帰で学習
  logistic_pcaN = LogisticRegressionCV(cv=10, random_state=0)
  logistic_pcaN.fit(X_train_pcaN, y_train_pcaN)

  # 検証
  print('Train score: {:.3f}'.format(logistic_pcaN.score(X_train_pcaN, y_train_pcaN)))
  print('Test score: {:.3f}'.format(logistic_pcaN.score(X_test_pcaN, y_test_pcaN)))
  print('Confustion matrix:¥n{}'.format(confusion_matrix(y_true=y_test_pcaN, y_pred=logistic_pcaN.predict(X_test_pcaN))))
```

```
-----PCA1-----
(569, 1)
Train score: 0.923
Test score: 0.902
Confustion matrix:
[[82  8]
 [ 6 47]]
-----PCA2-----
(569, 2)
```

```
-----PCA12-----
(569, 12)
Train score: 0.986
Test score: 0.965
Confustion matrix:
[[88  2]
 [ 3 50]]
-----PCA13-----
(569, 13)
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_logistic.py:940: ConvergenceWarning: lbfgs failed to converge (st
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
Train score: 0.986
Test score: 0.972
Confustion matrix:
[[88  2]
 [ 2 51]]
```

コードは本ドキュメントと同じレポジトリに提出する

https://github.com/toruuno/report_ml/blob/master/skl_pca.ipynb