SCIKIT-LEARN : DATA COMPRESSION VIA DIMENSIONALITY REDUCTION I - PRINCIPAL COMPONENT ANALYSIS (PCA)

SHARE
(http://www.addthis.com/bookmark.php?v=250&username=khhong7)

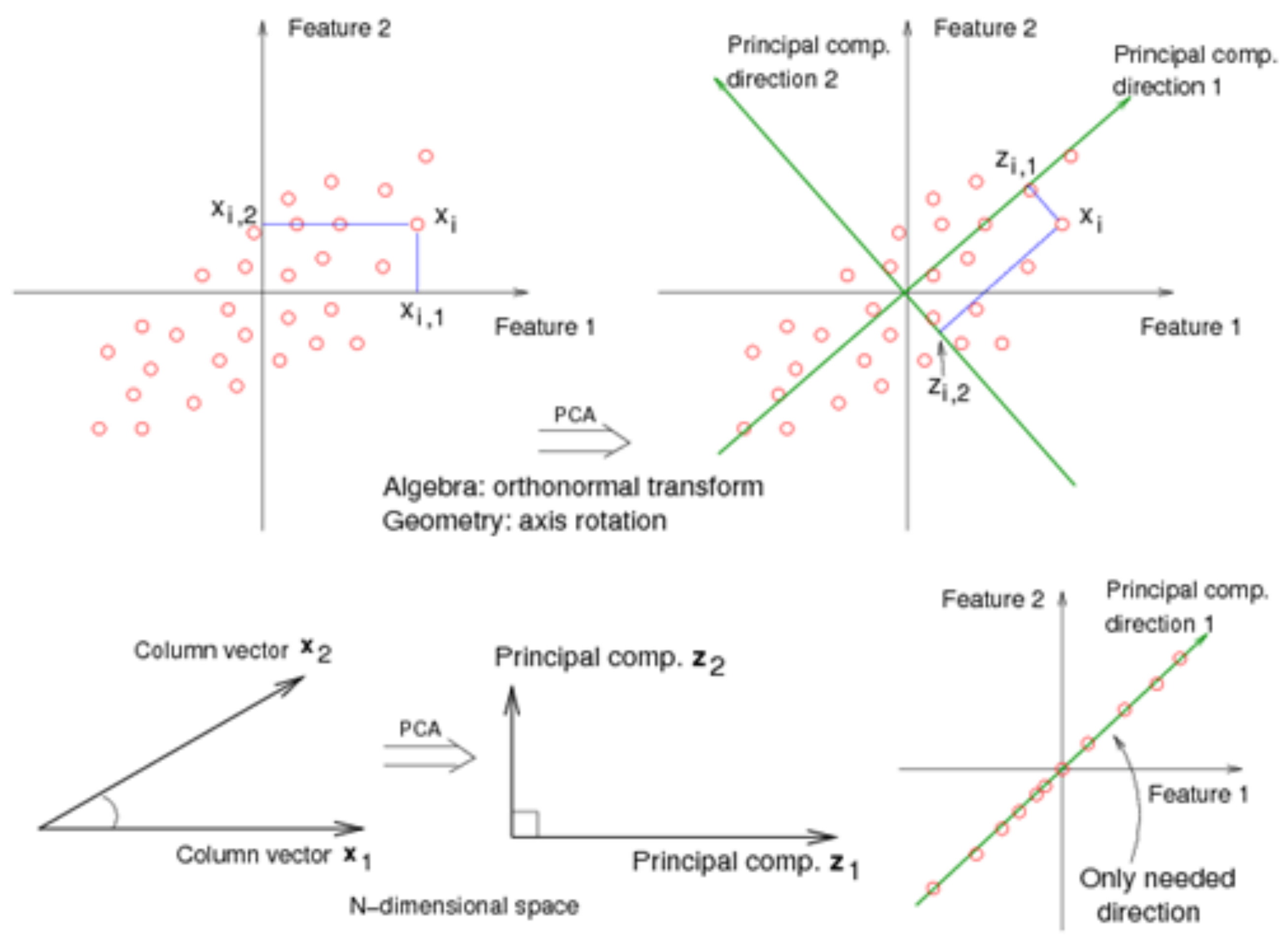
bogotobogo.com site search:

Custom Search Search

Principal component analysis (PCA)

> **Principal component analysis (PCA)** is a statistical procedure that uses an orthogonal transformation to convert a set of observations of possibly correlated variables into a set of values of linearly uncorrelated variables called principal components. - wiki (https://en.wikipedia.org/wiki/Principal_component_analysis)

PCA tries to find the directions of maximum variance (direction of orthogonal axes / principal components) in data and projects it onto a new subspace with lower dimension than the original one.



Principal Components Analysis (PCA) (https://onlinecourses.science.psu.edu/stat857/node/35)

Here, x_1 and x_2 are the original feature axes, and z_1 and z_2 are the principal components.

Dimensionality reduction via principal component analysis

In order to reduce dimensionality using PCA, we construct a transformation matrix W which has $d \times k$-dimension.

With the W matrix we can map a sample vector x onto a new k-dimensional feature subspace that has fewer dimensions than the original d-dimensional feature space:

$$\mathbf{x} = [x_1, x_2, \cdots, x_d], \qquad \mathbf{x} \in \mathbb{R}^d$$

$$\mathbf{x} = [x_1, x_2, \cdots, x_d], \qquad \mathbf{x} \in \mathrm{R}^d$$

$$\downarrow \mathbf{xW}, \qquad W \in \mathbb{R}^{d \times k}$$

$$\downarrow \mathbf{xW}, \qquad W \in \mathrm{R}^{d \times k}$$

$$\mathbf{z} = [z_1, z_2, \cdots, z_k], \qquad \mathbf{z} \in \mathbb{R}^k$$

$$\mathbf{z} = [z_1, z_2, \cdots, z_k], \qquad \mathbf{z} \in \mathrm{R}^k$$

After the transformation from the original d-dimensional data onto this new k-dimensional subspace ($k \le d$), the first principal component will have the largest possible variance, and all consequent principal components will have the largest possible variance given that they are uncorrelated (orthogonal) to the other principal components.

Machine Learning with scikit-learn

scikit-learn installation (/python/scikit-learn/scikit-learn_install.php)

scikit-learn : Features and feature extraction - iris dataset (/python/scikit-learn/scikit_machine_learning_fe

scikit-learn : Machine Learning Quick Preview (/python/scikit-learn/scikit_machine_learning_q

scikit-learn : Data Preprocessing I - Missing / Categorical data) (/python/scikit-learn/scikit_machine_learning_D Missing-Data-Categorical-Data.php)

scikit-learn : Data Preprocessing II - Partitioning a dataset / Feature scaling / Feature Selection / Regularization (/python/scikit-learn/scikit_machine_learning_D II-Datasets-Partitioning-Feature-scaling-Feature-Selection-Regularization.php)

scikit-learn : Data Preprocessing III - Dimensionality reduction vis Sequential feature selection / Assessing feature importance via random forests (/python/scikit-learn/scikit_machine_learning_D III-Dimensionality-reduction-via-Sequential-feature-selection-Assessing-feature-importance-via-random-forests.php)

Note that the PCA directions are highly sensitive to data scaling, and most likely we need to standardize the features prior to PCA if the features were measured on different scales and we want to assign equal importance to all features.

Here are the steps of PCA algorithm for dimensionality reduction:

1. Standardize the d-dimensional dataset.
2. Construct the covariance matrix.
3. Decompose the covariance matrix into its eigenvectors and eigenvalues.
4. Select k eigenvectors that correspond to the k largest eigenvalues, where k is the dimensionality of the new feature subspace ($k \le d$).
5. Construct a projection matrix W from the "top" k eigenvectors.
6. Transform the d-dimensional input dataset $\mathbf{x}$ using the projection matrix W to obtain the new k-dimensional feature subspace.

Eigenvalues & eigenvectors

Continued from the previous section for principal component analysis, in this section we'll standardize the data, construct the covariance matrix, obtain the eigenvalues and eigenvectors of the covariance matrix, and sort the eigenvalues by decreasing order to rank the eigenvectors.

Let's start by loading the Wine dataset from "https://archive.ics.uci.edu/ml/machine-learning-databases/wine/wine.data":

```
%pylab inline
```

```
Populating the interactive namespace from numpy and matplotlib
```

```
import pandas as pd

df_wine = pd.read_csv(
    'https://archive.ics.uci.edu/ml/machine-learning-databases/wine/wi
    header=None)
```

```
df_wine.head()
```

	0	1	2	3	4	5	6	7	8	9	10	11	12	13
0	1	14.23	1.71	2.43	15.6	127	2.80	3.06	0.28	2.29	5.64	1.04	3.92	1065
1	1	13.20	1.78	2.14	11.2	100	2.65	2.76	0.26	1.28	4.38	1.05	3.40	1050
2	1	13.16	2.36	2.67	18.6	101	2.80	3.24	0.30	2.81	5.68	1.03	3.17	1185
3	1	14.37	1.95	2.50	16.8	113	3.85	3.49	0.24	2.18	7.80	0.86	3.45	1480
4	1	13.24	2.59	2.87	21.0	118	2.80	2.69	0.39	1.82	4.32	1.04	2.93	735

Note that PCA is an **unsupervised** method, which means that information about the class labels is ignored. It shows clear contrast compared with a random forest which uses the class membership information to compute the node impurities, variance measures the spread of values along a feature axis. Recall we did the following for random forest:

```
df_wine.columns = ['Class label', 'Alcohol', 'Malic acid', 'Ash',
        'Alcalinity of ash', 'Magnesium', 'Total phenols', 'Flavanoids',
        'Nonflavanoid phenols', 'Proanthocyanins', 'Color intensity', 'Hue',
        'OD280/OD315 of diluted wines','Proline']
```

After the read-in, we process the Wine data into separate training (70%) and test (30%) sets and then standardize it to unit variance:

scikit-learn : Data Compression via Dimensionality Reduction I - Principal component analysis (PCA) (/python/scikit-learn/scikit_machine_learning_D_PCA.php)

scikit-learn : Data Compression via Dimensionality Reduction II - Linear Discriminant Analysis (LDA) (/python/scikit-learn/scikit_machine_learning_D

scikit-learn : Data Compression via Dimensionality Reduction III - Nonlinear mappings via kernel principal component (KPCA) analysis (/python/scikit-learn/scikit_machine_learning_D nonlinear-mappings-via-kernel-principal-component-analysis.php)

scikit-learn : Logistic Regression, Overfitting & regularization (/python/scikit-learn/scikit-learn_logistic_regression.php)

scikit-learn : Supervised Learning & Unsupervised Learning - e.g. Unsupervised PCA dimensionality reduction with iris dataset (/python/scikit-learn/scikit_machine_learning_S

scikit-learn : Unsupervised_Learning - KMeans clustering with iris dataset (/python/scikit-learn/scikit_machine_learning_U

scikit-learn : Linearly Separable Data - Linear Model & (Gaussian) radial basis function kernel (RBF kernel) (/python/scikit-learn/scikit_machine_learning_Li

scikit-learn : Decision Tree Learning I - Entropy, Gini, and Information Gain (/python/scikit-learn/scikt_machine_learning_De

scikit-learn : Decision Tree

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
X, y = df_wine.iloc[:, 1:].values, df_wine.iloc[:, 0].values
X_train, X_test, y_train, y_test = \
      train_test_split(X, y, test_size=0.3, random_state=0)
sc = StandardScaler()
X_train_std = sc.fit_transform(X_train)
X_test_std = sc.fit_transform(X_test)
```

Now we want to construct the covariance matrix which is symmetric with $d \times d$ $d \times d$-dimension, where d d is the dataset dimension. The covariance matrix stores the pairwise covariances between the different features.

The covariance between two features x_j x_j and x_k x_k on the population level can be calculated via the equation below:

$$\sigma_{jk} = \frac{1}{N} \sum_{i=1}^{N} (x_j^i - \mu_j)(x_k^i - \mu_k)$$

$$\sigma_{jk} = \frac{1}{N} \sum_{i=1}^{N} (x_j^i - \mu_j)(x_k^i - \mu_k)$$

where μ_j μ_j and μ_k μ_k are the sample means of feature j j and k k, respectively.

Note that the sample means are zero if we standardize the dataset.

A positive covariance between two features indicates that the features increase or decrease together, while a negative covariance means that the features vary in opposite directions.

A covariance matrix of three features can then be written as A A:

$$A = \begin{bmatrix} \sigma_{11}^2 & \sigma_{12} & \sigma_{13} \\ \sigma_{21} & \sigma_{22}^2 & \sigma_{23} \\ \sigma_{32} & \sigma_{32} & \sigma_{33}^2 \end{bmatrix}$$

$$A = \begin{bmatrix} \sigma_{11}^2 & \sigma_{12} & \sigma_{13} \\ \sigma_{21} & \sigma_{22}^2 & \sigma_{23} \\ \sigma_{32} & \sigma_{32} & \sigma_{33}^2 \end{bmatrix}$$

The eigenvectors of the covariance matrix represent the principal components, while the corresponding eigenvalues will define their magnitude.

In the case of the Wine dataset, we can obtain 13 eigenvectors and eigenvalues from the 13×13 13×13 covariance matrix.

An eigenvector ν ν satisfies the following condition where \lambda is the eigenvalue:

$$\mathbf{A}\nu = \lambda\nu$$

$$\mathbf{A}\nu = \lambda\nu$$

Learning II - Constructing the Decision Tree (/python/scikit-learn/scikit_machine_learning_C

scikit-learn : Random Decision Forests Classification (/python/scikit-learn/scikit_machine_learning_R

scikit-learn : k-Nearest Neighbors (k-NN) Algorithm (/python/scikit-learn/scikit_machine_learning_k-NN_k-nearest-neighbors-algorithm.php)

scikit-learn : Support Vector Machines (SVM) (/python/scikit-learn/scikit_machine_learning_S

scikit-learn : Support Vector Machines (SVM) II (/python/scikit-learn/scikit_machine_learning_S

Flask with Embedded Machine Learning I : Serializing with pickle and DB setup (/python/Flask/Python_Flask_Em

Flask with Embedded Machine Learning II : Basic Flask App (/python/Flask/Python_Flask_Em

Flask with Embedded Machine Learning III : Embedding Classifier (/python/Flask/Python_Flask_Em

Flask with Embedded Machine Learning IV : Deploy (/python/Flask/Python_Flask_Em

Flask with Embedded Machine Learning V : Updating the classifier (/python/Flask/Python_Flask_Em

scikit-learn : Sample of a spam comment filter using SVM - classifying a good one or a bad one (/python/scikit-learn/scikit_learn_Support_Vecto

We're going to use the **linalg.eig** function from NumPy to obtain the eigenpairs of the Wine covariance matrix:

```
covariant_matrix = np.cov(X_train_std.T)
```

```
covariant_matrix[0::5]
```

```
array([[ 1.00813008,  0.08797701,  0.23066952, -0.32868099,  0.2141631 ,
         0.35576761,  0.2991246 , -0.16913744,  0.09649074,  0.56962271,
        -0.04781543,  0.07403492,  0.63277882],
       [ 0.35576761, -0.30124242,  0.12235533, -0.37018442,  0.16513295,
         1.00813008,  0.88119961, -0.45396901,  0.6196806 , -0.06935051,
         0.45718802,  0.72214462,  0.56326772],
       [-0.04781543, -0.54992807, -0.10928021, -0.25313262,  0.05792599,
         0.45718802,  0.58331869, -0.3178224 ,  0.32282167, -0.52395358,
         1.00813008,  0.60022569,  0.2452794 ]])
```

```
eigen_values, eigen_vectors = np.linalg.eig(covariant_matrix)
```

```
eigen_values, eigen_vectors[::5]
```

```
(array([ 4.8923083 ,  2.46635032,  1.42809973,  1.01233462,  0.84906459,
         0.60181514,  0.52251546,  0.08414846,  0.33051429,  0.29595018,
         0.16831254,  0.21432212,  0.2399553 ]),
 array([[  1.46698114e-01,   5.04170789e-01,  -1.17235150e-01,
          2.06254611e-01,  -1.87815947e-01,  -1.48851318e-01,
         -1.79263662e-01,  -5.54687162e-02,  -4.03054922e-01,
         -4.17197583e-01,   2.75660860e-01,   4.03567189e-01,
          4.13320786e-04],
        [  3.89344551e-01,   9.36399132e-02,   1.80804417e-01,
          1.93179478e-01,   1.40645426e-01,   1.22248798e-02,
          5.31455344e-02,  -4.21265116e-01,   1.35111456e-01,
         -2.80985650e-01,   2.83897644e-01,  -6.18600153e-01,
          9.45645138e-02],
        [  3.00325353e-01,  -2.79243218e-01,   9.32387182e-02,
          2.41740256e-02,  -3.72610811e-01,   2.16515349e-01,
         -3.84654748e-01,  -1.05383688e-01,  -5.17259438e-01,
          1.97814118e-01,  -1.98844532e-01,  -2.00456386e-01,
         -3.02254353e-01]]))
```

We computed the covariance matrix of the standardized training dataset using the **numpy.cov()** function.

Using the linalg.eig function, we performed the eigendecomposition that yielded 13 **eigenvalues** and the corresponding **eigenvectors** stored as columns in a 13×13 matrix.

Since we want to reduce the dimensionality of our dataset by compressing it onto a new feature subspace, we only select the subset of the eigenvectors (principal components) that contains most of the information (variance).

Since the eigenvalues define the magnitude of the eigenvectors, we have to sort the eigenvalues by decreasing magnitude, and we are interested in the top k eigenvectors based on the values of their corresponding eigenvalues.

But before we collect those k most informative eigenvectors, let's plot the variance explained ratios of the eigenvalues.

The variance explained ratio of an eigenvalue λ_j is simply the fraction of an eigenvalue λ_j and the total sum of the eigenvalues:

$$\frac{\lambda_j}{\sum_{j=1}^{d} \lambda_j}$$

$$\frac{\lambda_j}{\sum_{j=1}^{d} \lambda_j}$$

Using the NumPy **cumsum()** function, we can then calculate the **cumulative sum** of explained variances, which we will plot via matplotlib's **step()** function:

MACHINE LEARNING ALGORITHMS

Batch gradient descent algorithm (/python/python_numpy_batch_

Single Layer Neural Network - Perceptron model on the Iris dataset using Heaviside step activation function (/python/scikit-learn/Perceptron_Model_with_Ir

Batch gradient descent versus stochastic gradient descent (SGD) (/python/scikit-learn/scikit-learn_batch-gradient-descent-versus-stochastic-gradient-descent.php)

Single Layer Neural Network - Adaptive Linear Neuron using linear (identity) activation function with batch gradient descent method (/python/scikit-learn/Single-Layer-Neural-Network-Adaptive-Linear-Neuron.php)

Single Layer Neural Network : Adaptive Linear Neuron using linear (identity) activation function with stochastic gradient descent (SGD) (/python/scikit-learn/Single-Layer-Neural-Network-Adaptive-Linear-Neuron-with-Stochastic-Gradient-Descent.php)

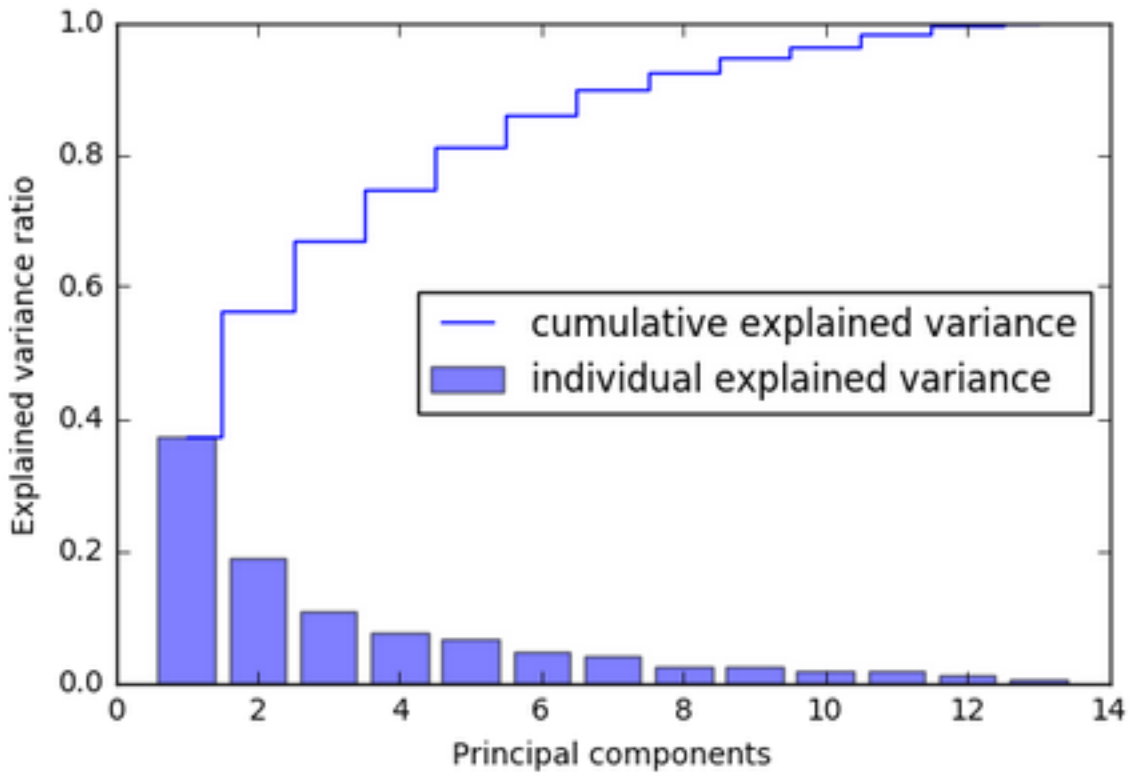
VC (Vapnik-Chervonenkis) Dimension and Shatter (/python/scikit-learn/scikit_machine_learning_V

Bias-variance tradeoff (/python/scikit-learn/scikit_machine_learning_B variance-Tradeoff.php)

Logistic Regression (/python/scikit-learn/logistic_regression.php)

Maximum Likelihood Estimation (MLE)

```
tot = sum(eigen_values)
var_exp = [(i / tot) for i in sorted(eigen_values, reverse=True)]
cum_var_exp = np.cumsum(var_exp)

bar(range(1,14), var_exp, alpha=0.5, align='center',
                 label='individual explained variance')
step(range(1,14), cum_var_exp, where='mid',
                 label='cumulative explained variance')
ylabel('Explained variance ratio')
xlabel('Principal components')
legend(loc='best')
show()
```



The plot shows that the first principal component alone accounts for 40 percent of the variance. Also, we can see that the first two principal components combined explain almost 60 percent of the variance in the data.

Transform dataset onto a new principal axes

Now that we have decomposed the covariance matrix into eigen-pairs, we want to transform the Wine dataset onto the new principal component axes.

We're going to sort the eigen-pairs by descending order of the eigenvalues, and construct a projection matrix from the selected eigenvectors. Then, using the projection matrix we will transform the data onto the lower-dimensional subspace.

Let's start by sorting the eigen-pairs by decreasing order of the eigenvalues:

```
eigen_pairs = \
[(np.abs(eigen_values[i]),eigen_vectors[:,i]) for i in range(len(eigen_values))]
eigen_pairs.sort(reverse=True)
```

```
eigen_pairs[:5]
```

```
[(4.8923083032737509,
  array([ 0.14669811, -0.24224554, -0.02993442, -0.25519002,  0.12079772,
          0.38934455,  0.42326486, -0.30634956,  0.30572219, -0.09869191,
          0.30032535,  0.36821154,  0.29259713])),
 (2.4663503157592306,
  array([ 0.50417079,  0.24216889,  0.28698484, -0.06468718,  0.22995385,
          0.09363991,  0.01088622,  0.01870216,  0.03040352,  0.54527081,
         -0.27924322, -0.174365  ,  0.36315461])),
 (1.4280997275048455,
  array([-0.11723515,  0.14994658,  0.65639439,  0.58428234,  0.08226275,
          0.18080442,  0.14295933,  0.17223475,  0.1583621 , -0.14242171,
          0.09323872,  0.19607741, -0.09731711])),
 (1.0123346209044966,
  array([ 0.20625461,  0.1304893 ,  0.01515363, -0.09042209, -0.83912835,
          0.19317948,  0.14045955,  0.33733262, -0.1147529 ,  0.07878571,
          0.02417403,  0.18402864,  0.05676778])),
 (0.8490645933450266,
  array([-0.18781595,  0.56863978, -0.29920943, -0.04124995, -0.02719713,
          0.14064543,  0.09268665, -0.08584168,  0.56510524,  0.01323461,
         -0.37261081,  0.08937967, -0.21752948]))]
```

Next, we collect the two eigenvectors that correspond to the two largest values to capture about 60 percent of the variance in this dataset.

(/python/scikit-learn/Maximum-Likelyhood-Estimation-MLE.php)

Neural Networks with backpropagation for XOR using one hidden layer (/python/python_Neural_Networ

minHash (/Algorithms/minHash_Jaccard_S

tf-idf weight (/Algorithms/tf_idf_term_frequen

Natural Language Processing (NLP): Sentiment Analysis I (IMDb & bag-of-words) (/Algorithms/Machine_Learning_

Natural Language Processing (NLP): Sentiment Analysis II (tokenization, stemming, and stop words) (/Algorithms/Machine_Learning_

Natural Language Processing (NLP): Sentiment Analysis III (training & cross validation) (/Algorithms/Machine_Learning_

Natural Language Processing (NLP): Sentiment Analysis IV (out-of-core) (/Algorithms/Machine_Learning_

Locality-Sensitive Hashing (LSH) using Cosine Distance (Cosine Similarity) (/Algorithms/Locality_Sensitive_H

ARTIFICIAL NEURAL NETWORKS (ANN)

1. Introduction (/python/scikit-learn/Artificial-Neural-Network-ANN-1-Introduction.php)

2. Forward Propagation (/python/scikit-learn/Artificial-Neural-Network-ANN-2-Forward-Propagation.php)

Note that choosing the number of principal components has to be determined from a trade-off between computational efficiency and the performance of the classifier, however, we only chose two eigenvectors for the demonstration purpose.

```
w= np.hstack((eigen_pairs[0][1][:, np.newaxis], eigen_pairs[1][1][:, np.newaxis]))
w.shape
```

```
(13, 2)
```

```
w
```

```
array([[ 0.14669811,  0.50417079],
       [-0.24224554,  0.24216889],
       [-0.02993442,  0.28698484],
       [-0.25519002, -0.06468718],
       [ 0.12079772,  0.22995385],
       [ 0.38934455,  0.09363991],
       [ 0.42326486,  0.01088622],
       [-0.30634956,  0.01870216],
       [ 0.30572219,  0.03040352],
       [-0.09869191,  0.54527081],
       [ 0.30032535, -0.27924322],
       [ 0.36821154, -0.174365  ],
       [ 0.29259713,  0.36315461]])
```

Now we've created a 13×2 **projection matrix $\mathbf{W}$** from the top two eigenvectors.

Using the projection matrix, we can now transform a sample $\mathbf{x}$ (represented as 1×13 row vector) onto the PCA subspace obtaining $\mathbf{x}'$ which is a 2-D sample vector consisting of two new features:

$$\mathbf{x}' = \mathbf{x}\mathbf{W}$$

$$\mathbf{x}' = \mathbf{x}\mathbf{W}$$


```
X_train_std[0]
```

```
array([ 0.91083058, -0.46259897, -0.01142613, -0.82067872,  0.06241693,
        0.58820446,  0.93565436, -0.7619138 ,  0.13007174, -0.51238741,
        0.65706596,  1.94354495,  0.93700997])
```

```
X_train_std[0].dot(w)
```

```
array([ 2.59891628,  0.00484089])
```

In the same way, we can transform the entire 124×13 training dataset onto the two principal components by calculating the matrix dot product:

$$\mathbf{X}' = \mathbf{X}\mathbf{W}$$

$$\mathbf{X}' = \mathbf{X}\mathbf{W}$$


```
X_train_pca = X_train_std.dot(w)
X_train_std.shape, w.shape, X_train_pca.shape
```

```
((124, 13), (13, 2), (124, 2))
```

Finally, it's time to visualize the transformed Wine training set, now stored as an 124×2 matrix, in a two-dimensional scatterplot:

3. Gradient Descent (/python/scikit-learn/Artificial-Neural-Network-ANN-3-Gradient-Descent.php)

4. Backpropagation of Errors (/python/scikit-learn/Artificial-Neural-Network-ANN-4-Backpropagation.php)

5. Checking gradient (/python/scikit-learn/Artificial-Neural-Network-ANN-5-Checking-Gradient.php)

6. Training via BFGS (/python/scikit-learn/Artificial-Neural-Network-ANN-6-Training-via-BFGS-Broyden-Fletcher-Goldfarb-Shanno-algorithm-a-variant-of-gradient-descent.php)

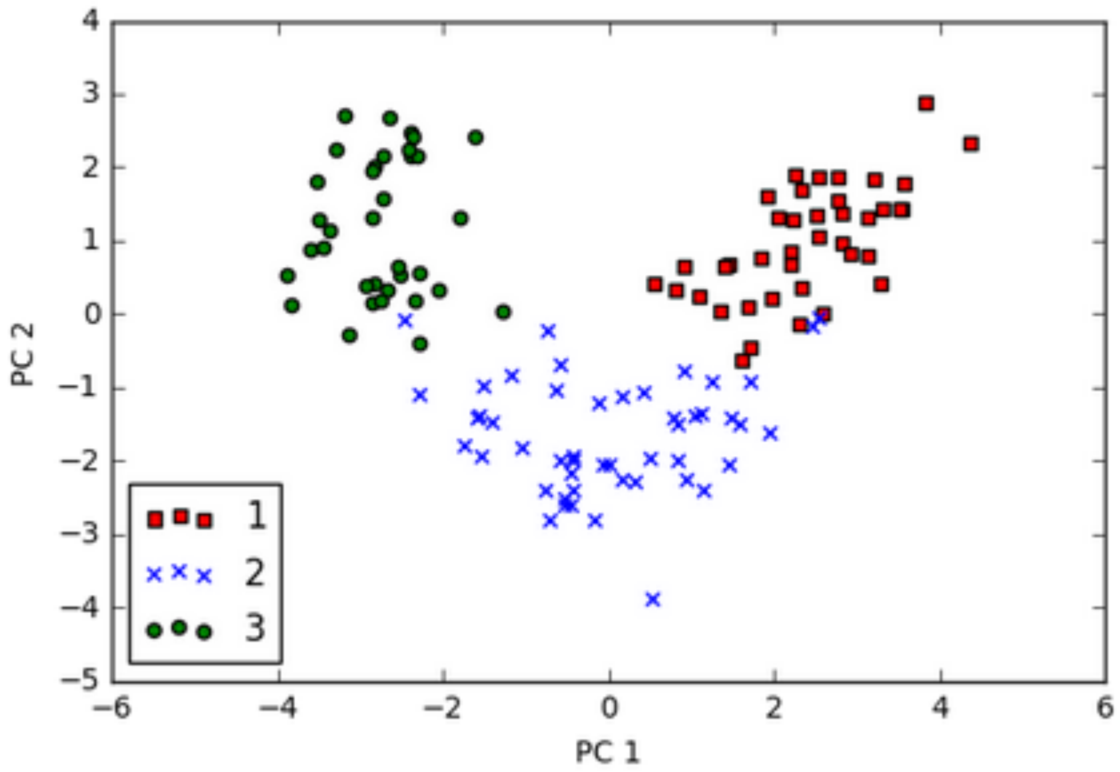
7. Overfitting & Regularization (/python/scikit-learn/Artificial-Neural-Network-ANN-7-Overfitting-Regularization.php)

8 - Deep Learning I : Image Recognition (Image uploading) (/python/scikit-learn/Artificial-Neural-Network-ANN-8-Deep-Learning-1-Image-Recognition-Image-Uploading.php)

9 - Deep Learning II : Image Recognition (Image classification) (/python/scikit-learn/Artificial-Neural-Network-ANN-9-Deep-Learning-2-Image-Recognition-Image-Classification.php)

10 - Deep Learning III : Deep Learning III : Theano, TensorFlow, and Keras (/python/scikit-learn/Artificial-Neural-Network-ANN-10-Deep-Learning-3-Theano-TensorFlow-Keras.php)

```
colors = ['r', 'b', 'g']
markers = ['s', 'x', 'o']
for l, c, m in zip(np.unique(y_train), colors, markers):
    scatter(X_train_pca[y_train==l, 0], X_train_pca[y_train==l, 1],
            c=c, label=l, marker=m)
xlabel('PC 1')
ylabel('PC 2')
legend(loc='lower left')
show()
```



We we can see from the plot, the data is more spread along the x-axis which is the first principal component than the y-axis which is the second principal component.

Though this is consistent with the explained variance ratio plot that we created in the previous subsection, we can intuitively see that a linear classifier will likely be able to separate the classes well.

One more thing to remind once again:
Although we encoded the class labels information for the purpose of illustration in the preceding scatter plot, we have to keep in mind that **PCA is an unsupervised** technique that doesn't use class label information.

PCA in scikit-learn

In this section we want to learn how to use the PCA class implemented in scikit-learn.

PCA is another one ofa scikit-learn's transformer classes, where we first fit the model using the training data before we transform both the training data and the test data using the same model parameters.

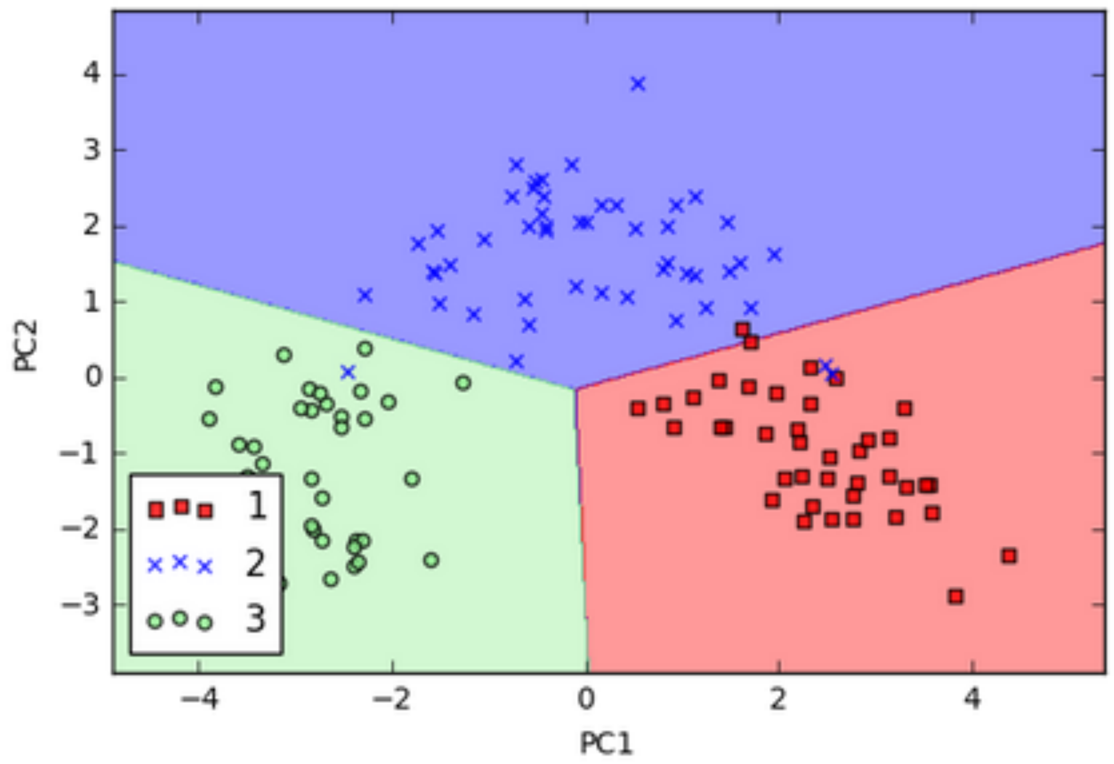
Let's use the PCA from scikit-learn on the Wine training dataset, and classify the transformed samples via logistic regression.

To visualize the decision regions, we'll use the following code:

Python tutorial

```
from matplotlib.colors import ListedColormap

def plot_decision_regions(X, y, classifier, resolution=0.02):

    # setup marker generator and color map
    markers = ('s', 'x', 'o', '^', 'v')
    colors = ('red', 'blue', 'lightgreen', 'gray', 'cyan')
    cmap = ListedColormap(colors[:len(np.unique(y))])

    # plot the decision surface
    x1_min, x1_max = X[:, 0].min() - 1, X[:, 0].max() + 1
    x2_min, x2_max = X[:, 1].min() - 1, X[:, 1].max() + 1
    xx1, xx2 = np.meshgrid(np.arange(x1_min, x1_max, resolution),
    np.arange(x2_min, x2_max, resolution))
    Z = classifier.predict(np.array([xx1.ravel(), xx2.ravel()]).T)
    Z = Z.reshape(xx1.shape)
    contourf(xx1, xx2, Z, alpha=0.4, cmap=cmap)
    xlim(xx1.min(), xx1.max())
    ylim(xx2.min(), xx2.max())

    # plot class samples
    for idx, cl in enumerate(np.unique(y)):
        scatter(x=X[y == cl, 0], y=X[y == cl, 1], alpha=0.8, c=cmap(idx),
                marker=markers[idx], label=cl)
```

```
from sklearn.linear_model import LogisticRegression
from sklearn.decomposition import PCA
pca = PCA(n_components=2)
lr = LogisticRegression()
X_train_pca = pca.fit_transform(X_train_std)
X_test_pca = pca.transform(X_test_std)
lr.fit(X_train_pca, y_train)
plot_decision_regions(X_train_pca, y_train, classifier=lr)
xlabel('PC1')
ylabel('PC2')
legend(loc='lower left')
show()
```



From the picture above, we can see the decision regions for the training model reduced to the two principal component axes.

Note: Covariance-matrix

This section explains the core concept of covariance matrix (ref Statistics 101: The Covariance Matrix (https://www.youtube.com/watch?v=locZabK4Als)).

Here is the data we're going to use:

String, and Slicing
(/python/python_strings.php)

Strings - Methods
(/python/python_strings_metho

Formatting Strings -
expressions and method calls
(/python/python_string_formatti

Files and os.path
(/python/python_files.php)

Traversing directories
recursively
(/python/python_traversing_dire

Subprocess Module
(/python/python_subprocess_mo

Regular Expressions with
Python
(/python/python_regularExpress

Object Types - Lists
(/python/python_lists.php)

Object Types - Dictionaries and
Tuples
(/python/python_dictionaries_tu

Functions def, *args, **kargs
(/python/python_functions_def.p

Functions lambda
(/python/python_functions_lamb

Built-in Functions
(/python/python_functions_built

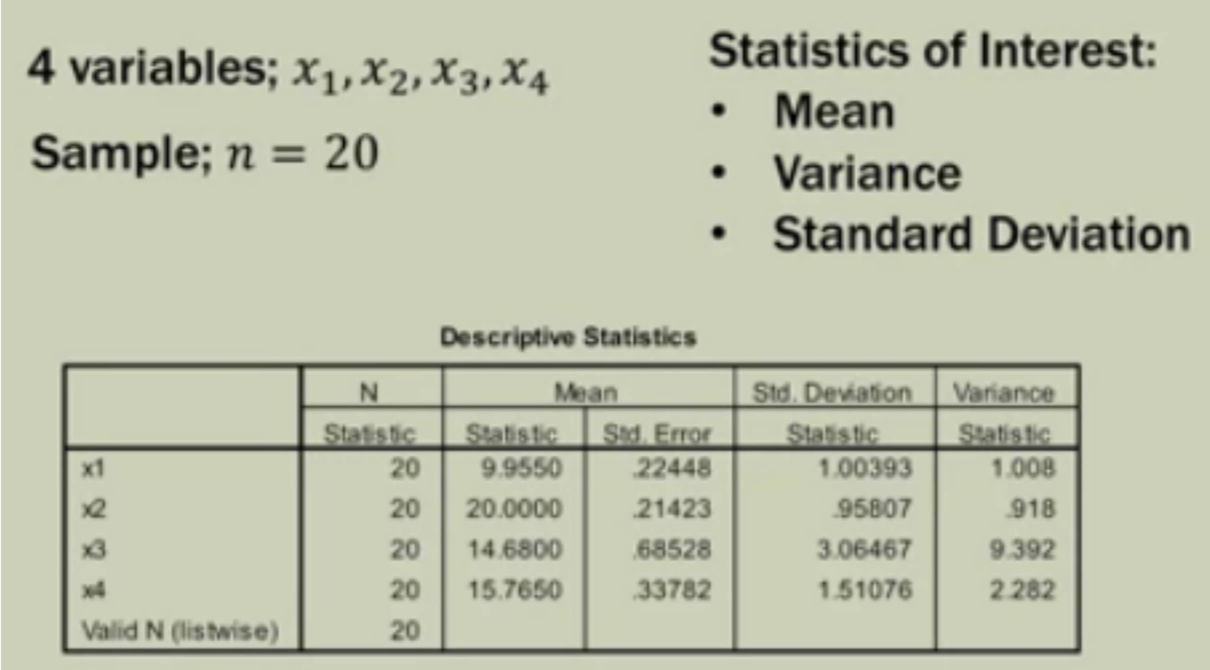
map, filter, and reduce
(/python/python_fncs_map_filter

Decorators
(/python/python_decorators.php

List Comprehension
(/python/python_list_comprehen

Sets (union/intersection) and
itertools - Jaccard coefficient
and shingling to check
plagiarism
(/python/python_sets_union_inte

Hashing (Hash tables and
hashlib)
(/python/python_hash_tables_ha



4 variables; x_1, x_2, x_3, x_4

Sample; $n = 20$

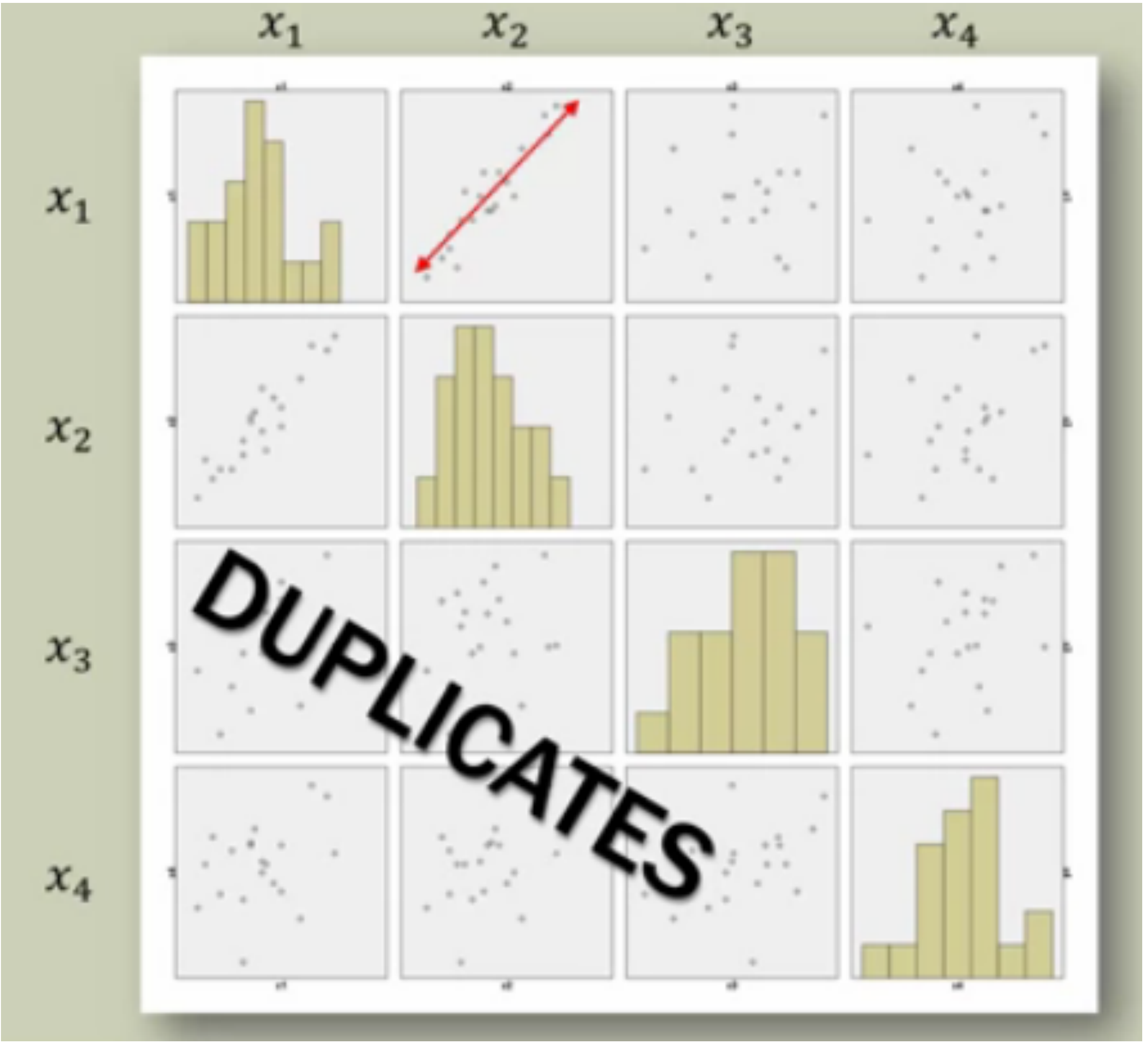
Statistics of Interest:

- Mean
- Variance
- Standard Deviation

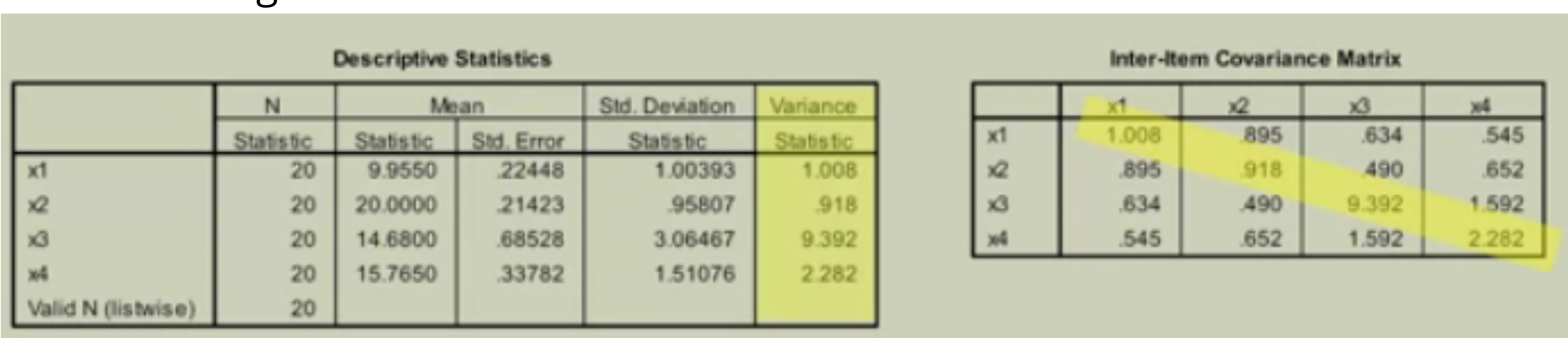
Descriptive Statistics

	N	Mean		Std. Deviation	Variance
	Statistic	Statistic	Std. Error	Statistic	Statistic
x1	20	9.9550	.22448	1.00393	1.008
x2	20	20.0000	.21423	.95807	.918
x3	20	14.6800	.68528	3.06467	9.392
x4	20	15.7650	.33782	1.51076	2.282
Valid N (listwise)	20				

Here is the scatter plots of each variable against the others:



So, we get the following covariance matrix



Descriptive Statistics

	N	Mean		Std. Deviation	Variance
	Statistic	Statistic	Std. Error	Statistic	Statistic
x1	20	9.9550	.22448	1.00393	1.008
x2	20	20.0000	.21423	.95807	.918
x3	20	14.6800	.68528	3.06467	9.392
x4	20	15.7650	.33782	1.51076	2.282
Valid N (listwise)	20				

Inter-Item Covariance Matrix

	x1	x2	x3	x4
x1	1.008	.895	.634	.545
x2	.895	.918	.490	.652
x3	.634	.490	9.392	1.592
x4	.545	.652	1.592	2.282

As we can see from the picture above, the diagonal of a covariance matrix gives us the two kinds of variances: the variance of each variables against others and the covariance with itself. So, the off-diagonal entries are the covariance between pair of variables.

	x_1	x_2	x_3	x_4
x_1	$Var(x_1)$	$Cov(x_1,x_2)$	$Cov(x_1,x_3)$	$Cov(x_1,x_4)$
x_2		$Var(x_2)$	$Cov(x_2,x_3)$	$Cov(x_2,x_4)$
x_3			$Var(x_3)$	$Cov(x_3,x_4)$
x_4				$Var(x_4)$

Github repository

Dictionary Comprehension with zip (/python/python_dictionary_com

The yield keyword (/python/python_function_with_y

Generator Functions and Expressions (/python/python_generators.php

generator.send() method (/python/python_function_with_g

Iterators (/python/python_iterators.php)

Classes and Instances (__init__, __call__, etc.) (/python/python_classes_instanc

if__name__ == '__main__' (/python/python_if__name__equ

argparse (/python/python_argparse.php)

Exceptions (/python/python_try_except_fina

@static method vs class method (/python/python_differences_be

Private attributes and private methods (/python/python_private_attribu

bits, bytes, bitstring, and constBitStream (/python/python_bits_bytes_bits

json.dump(s) and json.load(s) (/python/python-json-dumps-loads-file-read-write.php)

Python Object Serialization - pickle and json (/python/python_serialization_pi

Python Object Serialization - yaml and json (/python/python_yaml_json_conv

Priority queue and heap queue data structure (/python/python_PriorityQueue_

Source is available from bogotobogo-Machine-Learning (https://github.com/Einsteinish/bogotobogo-Machine-Learning.git).

Next:

Data Compression via Dimensionality Reduction II - Linear Discriminant Analysis (LDA) (/python/scikit-learn/scikit_machine_learning_Data_Compresssion_via_Dimensionality_Reduction_2_Linear_Discriminant_Analysis.php)

Machine Learning with scikit-learn

principal component (KPCA) analysis (/python/scikit-learn/scikit_machine_learning_Data_Compresssion_via_Dimensionality_Reduction_3-nonlinear-mappings-via-kernel-principal-component-analysis.php)

scikit-learn : Logistic Regression, Overfitting & regularization (/python/scikit-learn/scikit-learn_logistic_regression.php)

scikit-learn : Supervised Learning & Unsupervised Learning - e.g. Unsupervised PCA dimensionality reduction with iris dataset (/python/scikit-learn/scikit_machine_learning_Supervised_Learning_Unsupervised_Learning.php)

scikit-learn : Unsupervised_Learning - KMeans clustering with iris dataset (/python/scikit-learn/scikit_machine_learning_Unsupervised_Learning_Clustering.php)

scikit-learn : Linearly Separable Data - Linear Model & (Gaussian) radial basis function kernel (RBF kernel) (/python/scikit-learn/scikit_machine_learning_Linearly_Separable_NonLinearly_RBF_Separable_Data_SVM_GUI.php)

scikit-learn : Decision Tree Learning I - Entropy, Gini, and Information Gain (/python/scikit-learn/scikt_machine_learning_Decision_Tree_Learning_Informatioin_Gain_IG_Impurity_Entropy_Gini_Classification_Entropy.php)

scikit-learn : Decision Tree Learning II - Constructing the Decision Tree (/python/scikit-learn/scikit_machine_learning_Constructing_Decision_Tree_Learning_Information_Gain_IG_Impurity_Entropy_Gini_Classification.php)

scikit-learn : Random Decision Forests Classification (/python/scikit-learn/scikit_machine_learning_Random_Decision_Forests_Ensemble_Learning_Classification.php)

scikit-learn : Support Vector Machines (SVM) (/python/scikit-learn/scikit_machine_learning_Support_Vector_Machines_SVM.php)

scikit-learn : Support Vector Machines (SVM) II (/python/scikit-learn/scikit_machine_learning_Support_Vector_Machines_SVM_2.php)

Flask with Embedded Machine Learning I : Serializing with pickle and DB setup (/python/Flask/Python_Flask_Embedding_Machine_Learning_1.php)

Flask with Embedded Machine Learning II : Basic Flask App (/python/Flask/Python_Flask_Embedding_Machine_Learning_2.php)

Flask with Embedded Machine Learning III : Embedding Classifier (/python/Flask/Python_Flask_Embedding_Machine_Learning_3.php)

Flask with Embedded Machine Learning IV : Deploy (/python/Flask/Python_Flask_Embedding_Machine_Learning_4.php)

Flask with Embedded Machine Learning V : Updating the classifier (/python/Flask/Python_Flask_Embedding_Machine_Learning_5.php)

scikit-learn : Sample of a spam comment filter using SVM - classifying a good one or a bad one (/python/scikit-learn/scikit_learn_Support_Vector_Machines_SVM_spam_filtermachine_learning_.php)

MACHINE LEARNING ALGORITHMS AND CONCEPTS

Batch gradient descent algorithm (/python/python_numpy_batch_gradient_descent_algorithm.php)

Basics (/python/python_network_progr

Python Network Programming I - Basic Server / Client : B File Transfer (/python/python_network_progr

Python Network Programming II - Chat Server / Client (/python/python_network_progr

Python Network Programming III - Echo Server using socketserver network framework (/python/python_network_progr

Python Network Programming IV - Asynchronous Request Handling : ThreadingMixIn and ForkingMixIn (/python/python_network_progr

Python Interview Questions I (/python/python_interview_ques

Python Interview Questions II (/python/python_interview_ques

Python Interview Questions III (/python/python_interview_ques

Python Interview Questions IV (/python/python_interview_ques

Python Interview Questions V (/python/python_interview_ques

Image processing with Python image library Pillow (/python/python_image_process

Python and C++ with SIP (/python/python_cpp_sip.php)

PyDev with Eclipse (/python/pydev_eclipse_plugin_i

Matplotlib (/python/python_matplotlib.php

Redis with Python (/python/python_redis_with_pyth

NumPy array basics A (/python/python_numpy_array_t

Single Layer Neural Network - Perceptron model on the Iris dataset using Heaviside step activation function (/python/scikit-learn/Perceptron_Model_with_Iris_DataSet.php)

Batch gradient descent versus stochastic gradient descent (/python/scikit-learn/scikit-learn_batch-gradient-descent-versus-stochastic-gradient-descent.php)

Single Layer Neural Network - Adaptive Linear Neuron using linear (identity) activation function with batch gradient descent method (/python/scikit-learn/Single-Layer-Neural-Network-Adaptive-Linear-Neuron.php)

Single Layer Neural Network : Adaptive Linear Neuron using linear (identity) activation function with stochastic gradient descent (SGD) (/python/scikit-learn/Single-Layer-Neural-Network-Adaptive-Linear-Neuron-with-Stochastic-Gradient-Descent.php)

Logistic Regression (/python/scikit-learn/logistic_regression.php)

VC (Vapnik-Chervonenkis) Dimension and Shatter (/python/scikit-learn/scikit_machine_learning_VC_Dimension_Shatter.php)

Bias-variance tradeoff (/python/scikit-learn/scikit_machine_learning_Bias-variance-Tradeoff.php)

Maximum Likelihood Estimation (MLE) (/python/scikit-learn/Maximum-Likelyhood-Estimation-MLE.php)

Neural Networks with backpropagation for XOR using one hidden layer (/python/python_Neural_Networks_Backpropagation_for_XOR_using_one_hidden_layer.php)

minHash (/Algorithms/minHash_Jaccard_Similarity_Locality_sensitive_hashing_LSH.php)

tf-idf weight (/Algorithms/tf_idf_term_frequency_inverse_document_frequency_NLP_Natural_Language_Processing.php)

Natural Language Processing (NLP): Sentiment Analysis I (IMDb & bag-of-words) (/Algorithms/Machine_Learning_NLP_Sentiment_Analysis_1.php)

Natural Language Processing (NLP): Sentiment Analysis II (tokenization, stemming, and stop words) (/Algorithms/Machine_Learning_NLP_Sentiment_Analysis_2.php)

Natural Language Processing (NLP): Sentiment Analysis III (training & cross validation) (/Algorithms/Machine_Learning_NLP_Sentiment_Analysis_3.php)

Natural Language Processing (NLP): Sentiment Analysis IV (out-of-core) (/Algorithms/Machine_Learning_NLP_Sentiment_Analysis_4.php)

Locality-Sensitive Hashing (LSH) using Cosine Distance (Cosine Similarity) (/Algorithms/Locality_Sensitive_Hashing_LSH_using_Cosine_Distance_Similarity.php)

ARTIFICIAL NEURAL NETWORKS (ANN)

[Note] Sources are available at Github - Jupyter notebook files (https://github.com/Einsteinish/Artificial-Neural-Networks-with-Jupyter.git)

1. Introduction (/python/scikit-learn/Artificial-Neural-Network-ANN-1-Introduction.php)

NumPy Matrix and Linear Algebra (/python/python_numpy_matrix_

Pandas with NumPy and Matplotlib (/python/python_Pandas_NumPy

Celluar Automata (/python/python_cellular_autom

Batch gradient descent algorithm (/python/python_numpy_batch_

Longest Common Substring Algorithm (/python/python_longest_comm

Python Unit Test - TDD using unittest.TestCase class (/python/python_unit_testing.ph

Simple tool - Google page ranking by keywords (/python/python_site_page_rank

Google App Hello World (/python/GoogleApp/python_Go

Google App webapp2 and WSGI (/python/GoogleApp/python_Go

Uploading Google App Hello World (/python/GoogleApp/python_Go

Python 2 vs Python 3 (/python/python_differences_Py

virtualenv and virtualenvwrapper (/python/python_virtualenv_virtu

Uploading a big file to AWS S3 using boto module (/DevOps/AWS/aws_S3_uploadin

Scheduled stopping and starting an AWS instance (/DevOps/AWS/aws_stopping_st

Cloudera CDH5 - Scheduled stopping and starting services (/Hadoop/BigData_hadoop_CDH

Removing Cloud Files -

2. Forward Propagation (/python/scikit-learn/Artificial-Neural-Network-ANN-2-Forward-Propagation.php)

3. Gradient Descent (/python/scikit-learn/Artificial-Neural-Network-ANN-3-Gradient-Descent.php)

4. Backpropagation of Errors (/python/scikit-learn/Artificial-Neural-Network-ANN-4-Backpropagation.php)

5. Checking gradient (/python/scikit-learn/Artificial-Neural-Network-ANN-5-Checking-Gradient.php)

6. Training via BFGS (/python/scikit-learn/Artificial-Neural-Network-ANN-6-Training-via-BFGS-Broyden-Fletcher-Goldfarb-Shanno-algorithm-a-variant-of-gradient-descent.php)

7. Overfitting & Regularization (/python/scikit-learn/Artificial-Neural-Network-ANN-7-Overfitting-Regularization.php)

8. Deep Learning I : Image Recognition (Image uploading) (/python/scikit-learn/Artificial-Neural-Network-ANN-8-Deep-Learning-1-Image-Recognition-Image-Uploading.php)

9. Deep Learning II : Image Recognition (Image classification) (/python/scikit-learn/Artificial-Neural-Network-ANN-9-Deep-Learning-2-Image-Recognition-Image-Classification.php)

10 - Deep Learning III : Deep Learning III : Theano, TensorFlow, and Keras (/python/scikit-learn/Artificial-Neural-Network-ANN-10-Deep-Learning-3-Theano-TensorFlow-Keras.php)

Rackspace API with curl and subprocess (/python/python_Rackspace_API

Checking if a process is running/hanging and stop/run a scheduled task on Windows (/python/python-Windows-Check-if-a-Process-is-Running-Hanging-Schtasks-Run-Stop.php)

Apache Spark 1.3 with PySpark (Spark Python API) Shell (/Hadoop/BigData_hadoop_Apac

Apache Spark 1.2 Streaming (/Hadoop/BigData_hadoop_Apac

bottle 0.12.7 - Fast and simple WSGI-micro framework for small web-applications ... (/python/Bottle/Python_Bottle_F

Flask app with Apache WSGI on Ubuntu14/CentOS7 ... (/python/Flask/Python_Flask_Blo

Selenium WebDriver (/python/python_Selenium_Web

Fabric - streamlining the use of SSH for application deployment (/python/Fabric/python_Fabric.p

Ansible Quick Preview - Setting up web servers with Nginx, configure enviroments, and deploy an App (/DevOps/Ansible/Ansible_Settin

Neural Networks with backpropagation for XOR using one hidden layer (/python/python_Neural_Networ

NLP - NLTK (Natural Language Toolkit) ... (/python/NLTK/NLTK_install.php

RabbitMQ(Message broker server) and Celery(Task queue) ... (/python/RabbitMQ_Celery/pytho

OpenCV3 and Matplotlib ... (/python/OpenCV_Python/pytho

Simple tool - Concatenating slides using FFmpeg ... (/FFMpeg/ffmpeg_fade_in_fade_o

iPython - Signal Processing with NumPy (/python/OpenCV_Python/pytho

iPython and Jupyter - Install Jupyter, iPython Notebook, drawing with Matplotlib, and publishing it to Github (/python/IPython/IPython_Jupyte

iPython and Jupyter Notebook with Embedded D3.js (/python/IPython/iPython_Jupyte

Downloading YouTube videos using youtube-dl embedded with Python (/VideoStreaming/YouTube/yout dl-embedding.php)

Machine Learning : scikit-learn ... (/python/scikit-learn/scikit_machine_learning_S

Django 1.6/1.8 Web Framework ... (/python/Django/Python_Django

CONTACT

BogoToBogo
contactus@bogotobogo.com (mailto:#)

FOLLOW BOGOTOBOGO

(https://www.facebook.com/KHongSanFrancisco) (https://twitter.com/KHongTwit) (https://plus.google.com/u/0/+KHongSanFrancisco/posts)

ABOUT US (/ABOUT_US.PHP)

contactus@bogotobogo.com (mailto:#)

Golden Gate Ave, San Francisco, CA 94115

Golden Gate Ave, San Francisco, CA 94115