Unsupervised Learning

with Scikit Learn

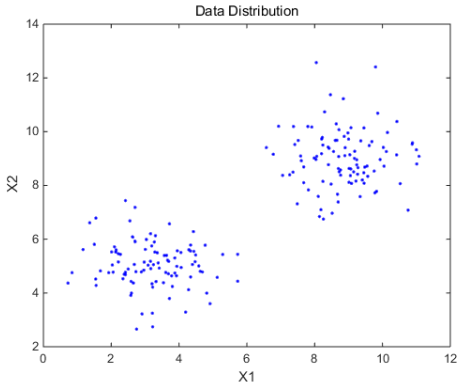
by Prof. Seungchul Lee
iSystems Design Lab
http://isystems.unist.ac.kr/
UNIST

Table of Contents

1. K-means Clustering

Unsupervised Learning

- Data clustering is an unsupervised learning problem
- Given:
  - m unlabeled examples $\{x^{(1)}, x^{(2)} \cdots, x^{(m)}\}$
  - the number of partitions k
- Goal: group the examples into k partitions



$$\{x^{(1)}, x^{(2)}, \cdots, x^{(m)}\} \quad \Rightarrow \quad \text{Clustering}$$

- the only information clustering uses is the similarity between examples
- clustering groups examples based of their mutual similarities
- A good clustering is one that achieves:
  - high within-cluster similarity
  - low inter-cluster similarity

1.1. (Iterative) Algorithm

Randomly initialize k cluster centroids $\mu_1, \mu_2, \cdots, \mu_k \in \mathbb{R}^n$

Repeat{

for $i = 1$ to m

$c_i :=$ index (from 1 to k) of cluster centroid closest to $x^{(i)}$

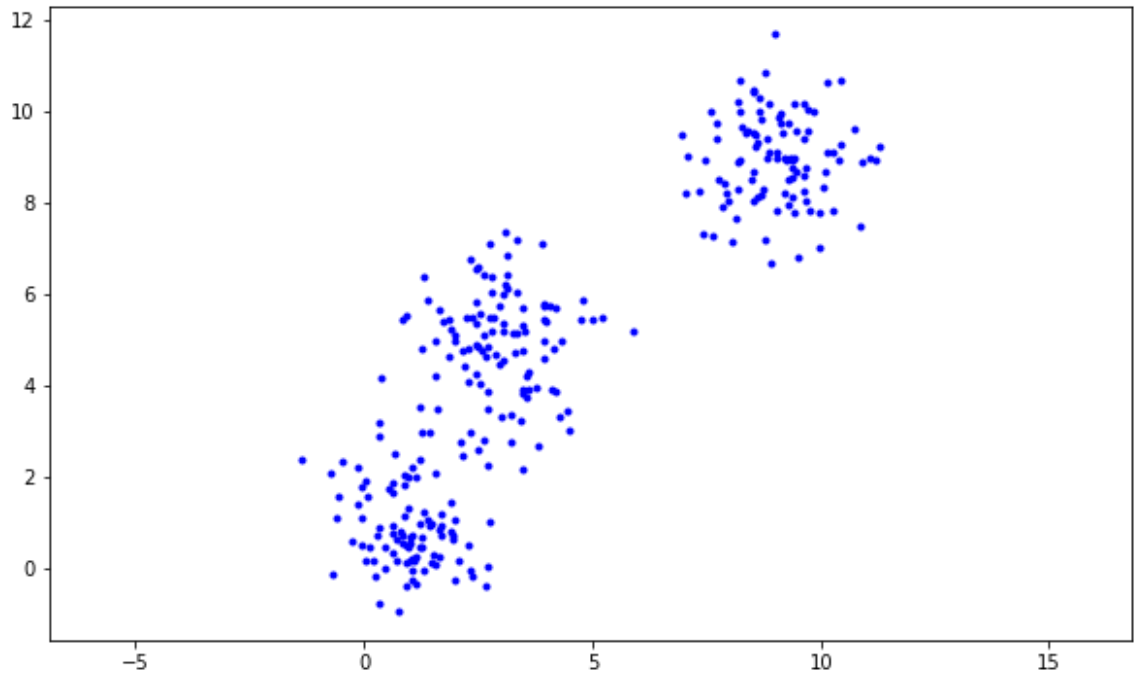
for $k = 1$ to k

$\mu_k :=$ average (mean) of points assigned to cluster k

}

In [1]:

```
import numpy as np
import matplotlib.pyplot as plt
from six.moves import cPickle
%matplotlib inline

X = cPickle.load(open('./data_files/kmeans_example.pkl','rb'))

plt.figure(figsize=(10,6))
plt.plot(X[:,0],X[:,1],'b.')
plt.axis('equal')
plt.show()
```



In [2]:

```
from sklearn.cluster import KMeans

kmeans = KMeans(n_clusters = 3, random_state = 0)
kmeans.fit(X)
```

Out[2]:

```
KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=300,
    n_clusters=3, n_init=10, n_jobs=1, precompute_distances='auto',
    random_state=0, tol=0.0001, verbose=0)
```

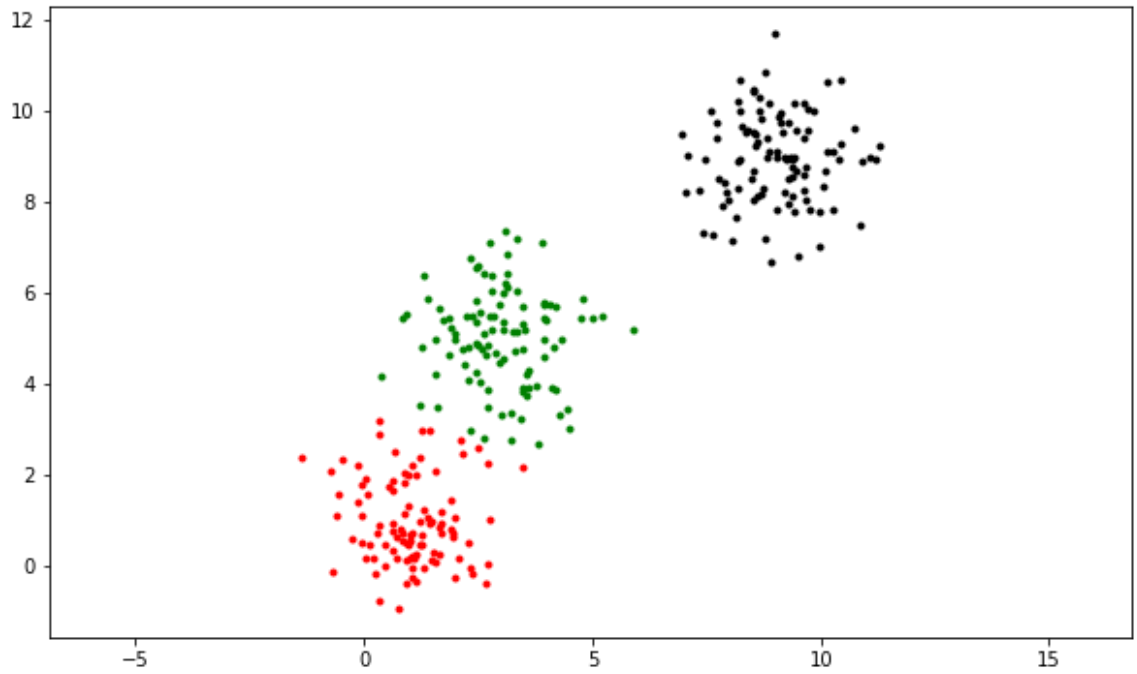
In [3]:

```
print(kmeans.labels_)
```

```
[2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 0 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 0 2 2 2 2 2 2 2 2 2 2 0 2 2 2 2 2 2 2 0 0 0 0 0 0 0 2 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1]
```

In [4]:

```
plt.figure(figsize=(10,6))

plt.plot(X[kmeans.labels_ == 0,0],X[kmeans.labels_ == 0,1],'g.')
plt.plot(X[kmeans.labels_ == 1,0],X[kmeans.labels_ == 1,1],'k.')
plt.plot(X[kmeans.labels_ == 2,0],X[kmeans.labels_ == 2,1],'r.')

plt.axis('equal')
plt.show()
```

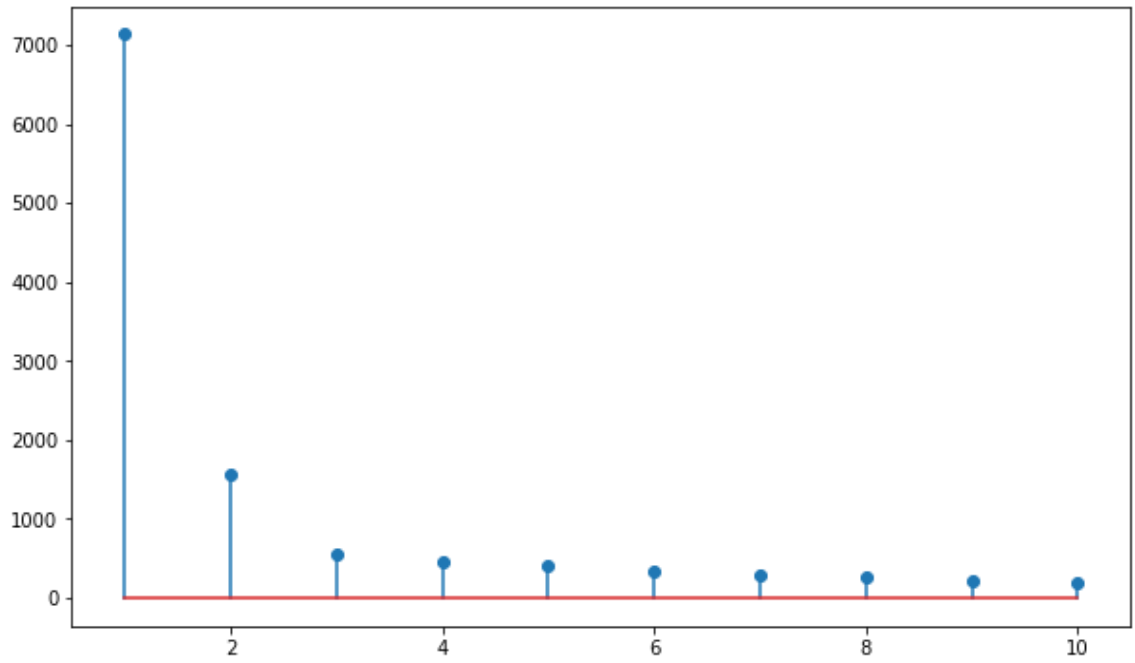


1.2. Choosing the Number of Clusters

- Idea: when adding another cluster does not give much better modeling of the data
- One way to select k for the K-means algorithm is to try different values of k, plot the K-means objective versus k, and look at the 'elbow-point' in the plot

In [5]:

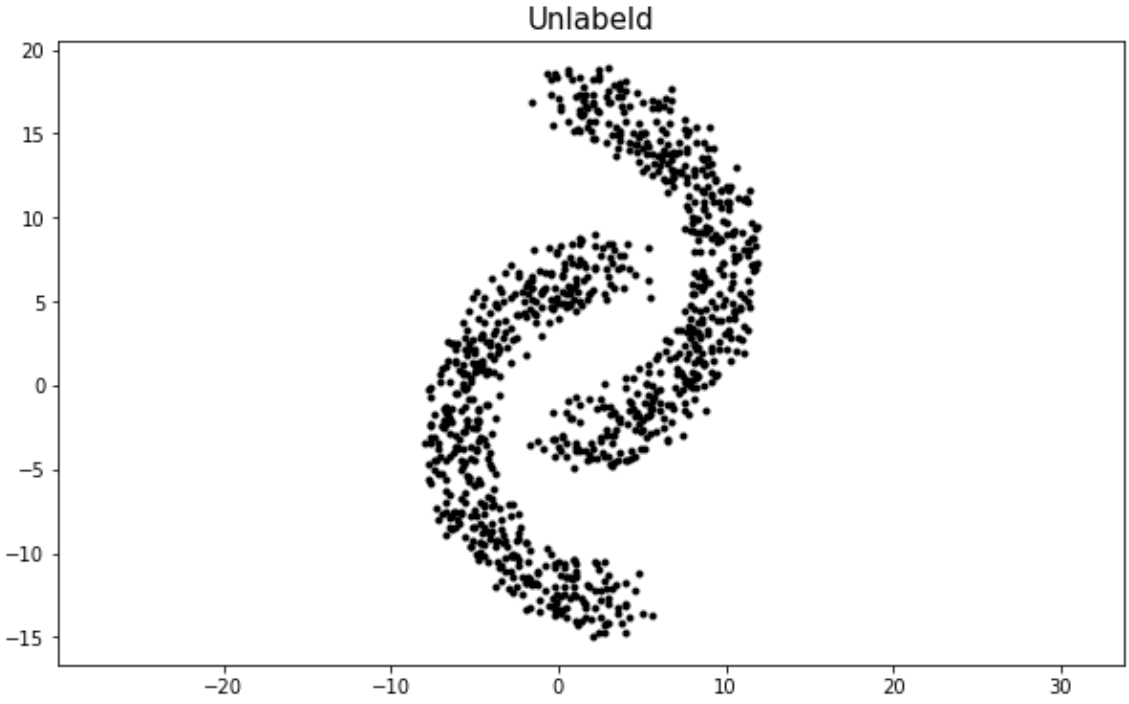
```
cost = []
for i in range(1,11):
    kmeans = KMeans(n_clusters=i, random_state=0).fit(X)
    cost.append(abs(kmeans.score(X)))

plt.figure(figsize=(10,6))
plt.stem(range(1,11),cost)
plt.xlim([0.5, 10.5])
plt.show()
```



1.3. K-means: Limitations

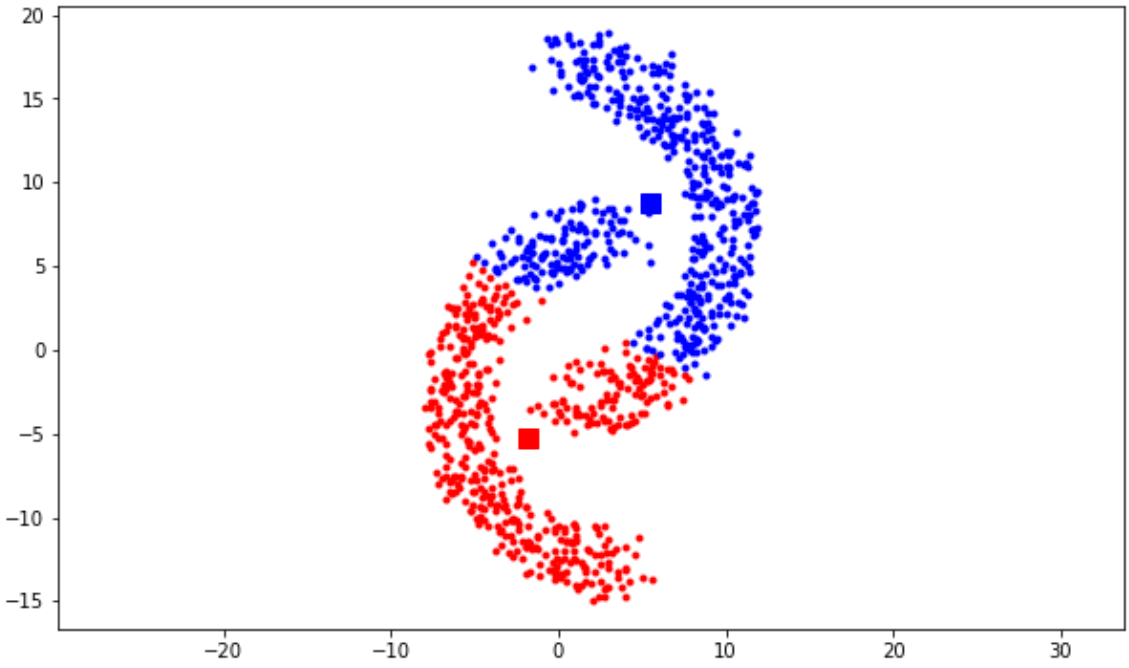
In [6]:

```
X = cPickle.load(open('./data_files/kmeans_lim.pkl','rb'))

plt.figure(figsize=(10,6))
plt.plot(X[:,0], X[:,1],'k.')

plt.title('Unlabeld', fontsize='15')
plt.axis('equal')
plt.show()
```



In [7]:

```
kmeans = KMeans(n_clusters=2, random_state=0).fit(X)

plt.figure(figsize=(10,6))
plt.plot(X[kmeans.labels_ == 0,0],X[kmeans.labels_ == 0,1],'r.')
plt.plot(X[kmeans.labels_ == 1,0],X[kmeans.labels_ == 1,1],'b.')
plt.plot(kmeans.cluster_centers_[0][0], kmeans.cluster_centers_[0][1],'rs',markersize=1
0)
plt.plot(kmeans.cluster_centers_[1][0], kmeans.cluster_centers_[1][1],'bs',markersize=1
0)

plt.axis('equal')
plt.show()
```



2. Gaussian Mixture Model

Let's consider a generative model for the data

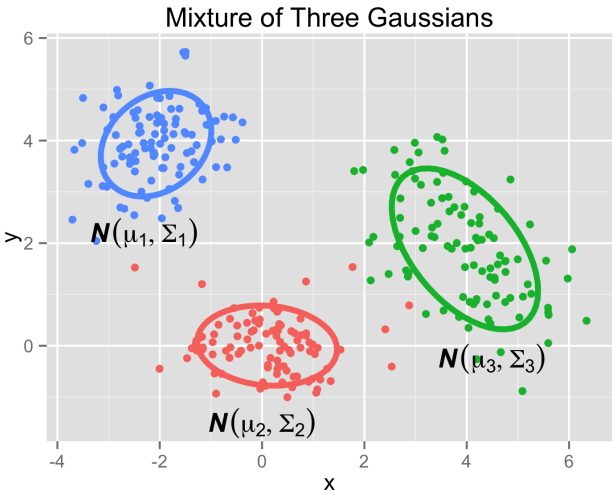
Suppose

- there are k clusters
- we have a (Gaussian) probability density for each cluster

Generate a point as follows:

1) Choose a random cluster $z \in \{1, 2, \cdots, k\}$

2) Choose a point from the (Gaussian) distribution for cluster z_i

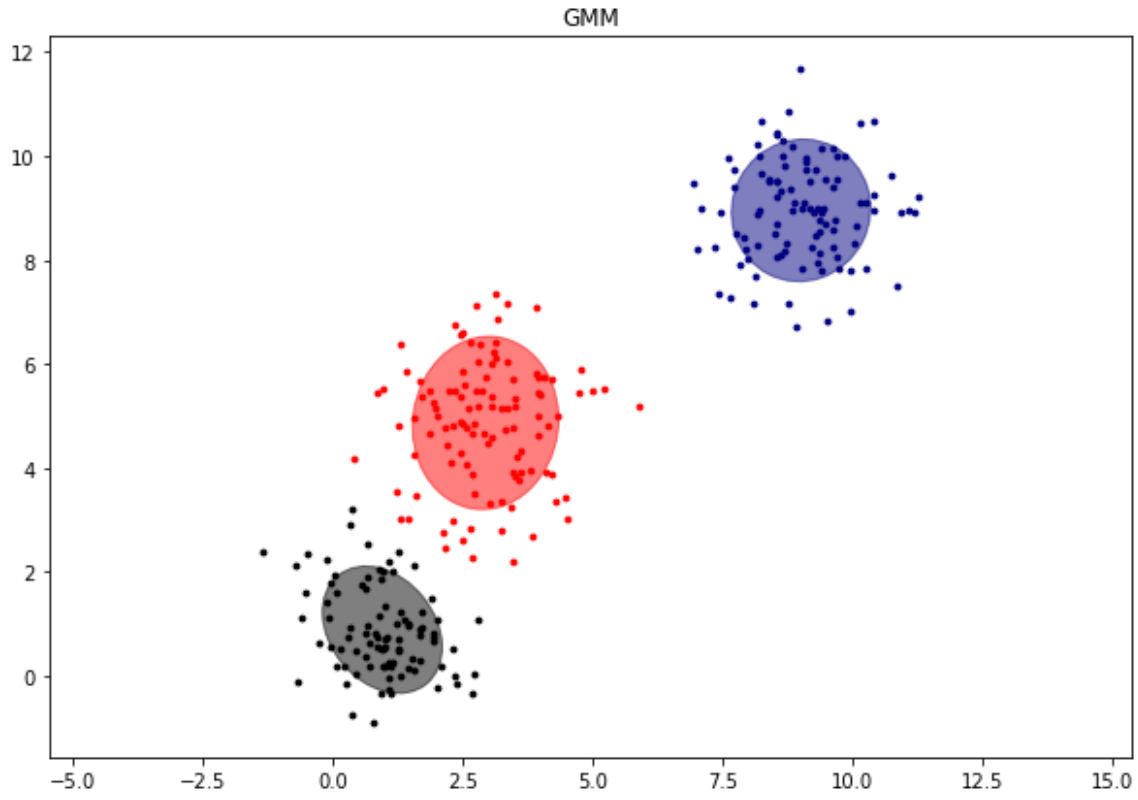


In [8]:

```
from sklearn import mixture
import itertools
from scipy import linalg
import matplotlib as mpl

X = cPickle.load(open('./data_files/kmeans_example.pkl','rb'))

gmm = mixture.GaussianMixture(n_components=3, covariance_type='full')
gmm.fit(X)
```

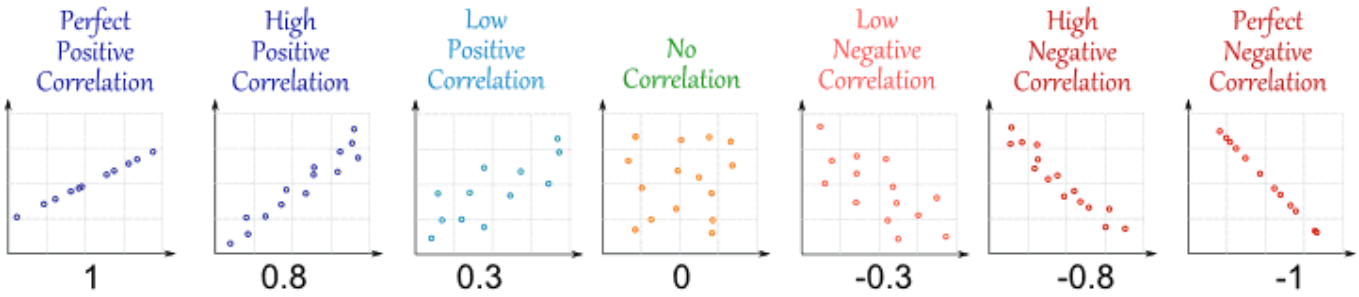
In [9]:

```
# Plot (do not have to understand the code)

plt.figure(figsize=(10,6))
splot = plt.subplot(1, 1, 1)
Y_ = gmm.predict(X)

color_iter = itertools.cycle(['navy', 'black', 'red','darkorange'])

for i, (mean, cov, color) in enumerate(zip(gmm.means_, gmm.covariances_,
                                           color_iter)):
    v, w = linalg.eigh(cov)
    if not np.any(Y_ == i):
        continue
    plt.scatter(X[Y_ == i, 0], X[Y_ == i, 1], 8.0, color=color)

    # Plot an ellipse to show the Gaussian component
    angle = np.arctan2(w[0][1], w[0][0])
    angle = 180. * angle / np.pi  # convert to degrees
    v = 2. * np.sqrt(2.) * np.sqrt(v)
    ell = mpl.patches.Ellipse(mean, v[0], v[1], 180. + angle, color=color)
    ell.set_clip_box(splot.bbox)
    ell.set_alpha(.5)
    splot.add_artist(ell)

plt.title('GMM')
plt.subplots_adjust(hspace=.35, bottom=.02)
plt.axis('equal')
plt.show()
```



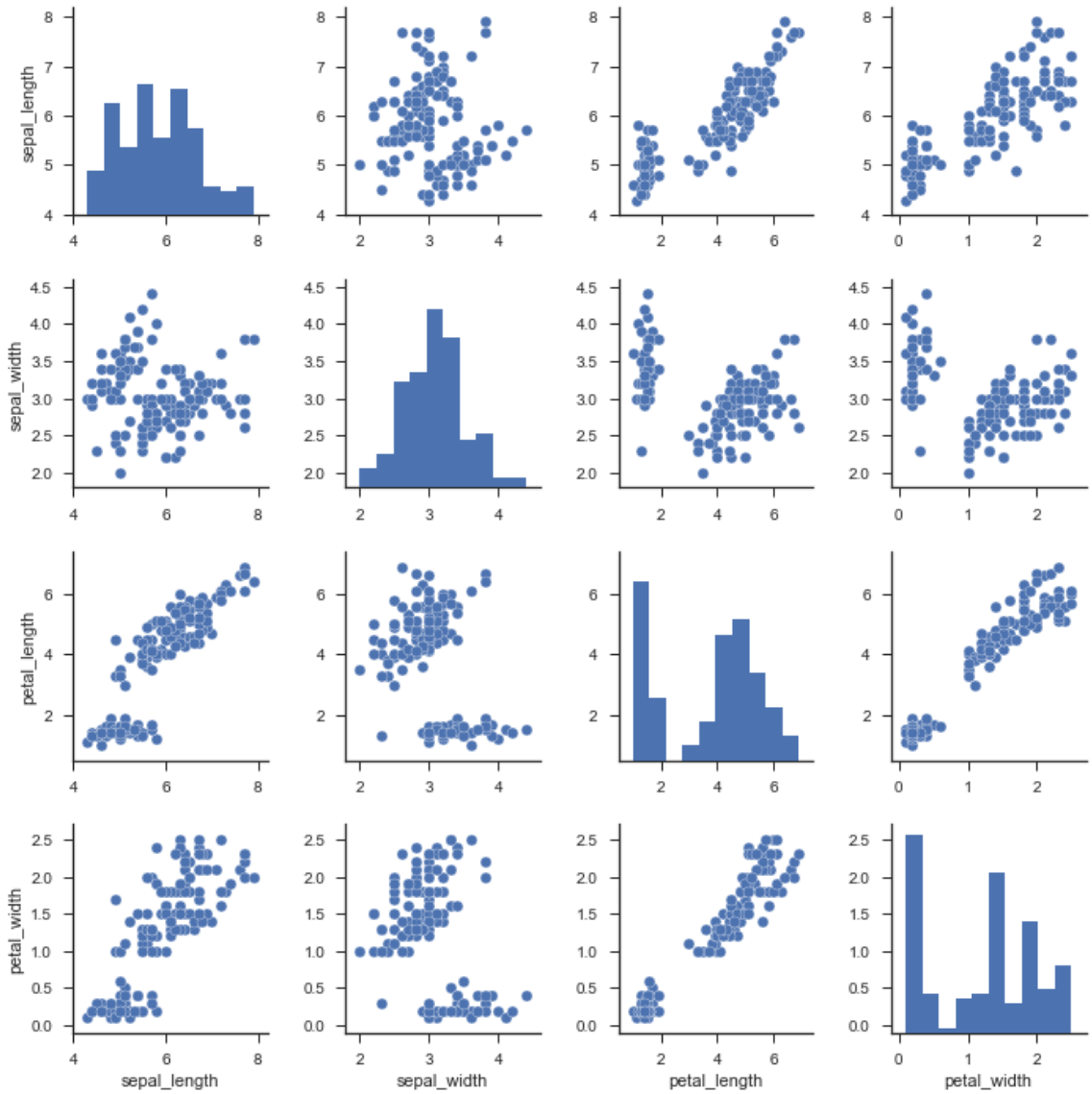
3. Correlation Analysis

- Statistical relationship between two sets of data
- http://rpsychologist.com/d3/correlation/ (http://rpsychologist.com/d3/correlation/)



In [10]:

```
import seaborn as sns
import pandas

sns.set(style="ticks", color_codes=True)
iris = cPickle.load(open('./data_files/iris.pkl', 'rb'))

g = sns.pairplot(iris)
# sns.plt.show()
```



4. Principal Component Analysis (PCA)

Motivation: Can we describe high-dimensional data in a "simpler" way?

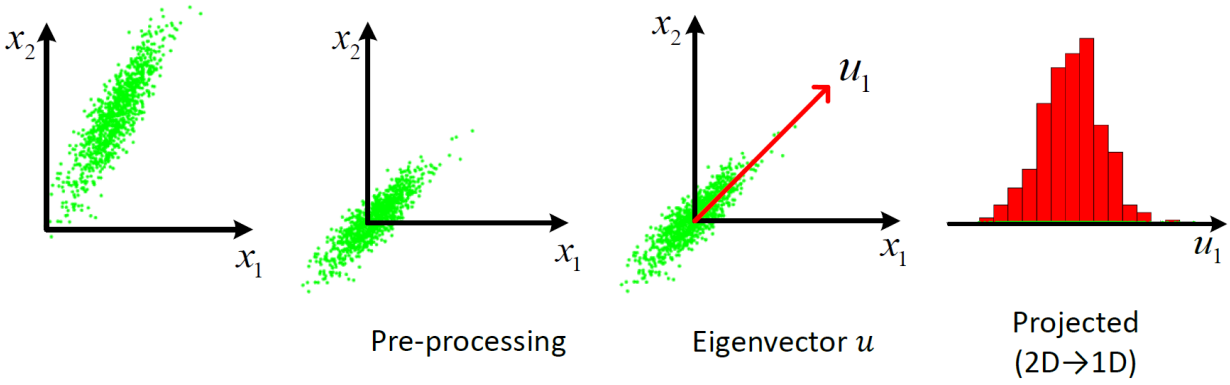
→ Dimension reduction without losing too much information
→ Find a low-dimensional, yet useful representation of the data

Dimension Reduction method

1. Choose top k (orthonormal) eigenvectors, $U = [u_1, u_2, \cdots, u_k]$
2. Project x_i onto span $\{u_1, u_2, \cdots, u_k\}$

$$z^{(i)} = \begin{bmatrix} u_1^T x^{(i)} \\ u_2^T x^{(i)} \\ \vdots \\ u_k^T x^{(i)} \end{bmatrix} \quad \text{or} \quad z = U^T x$$

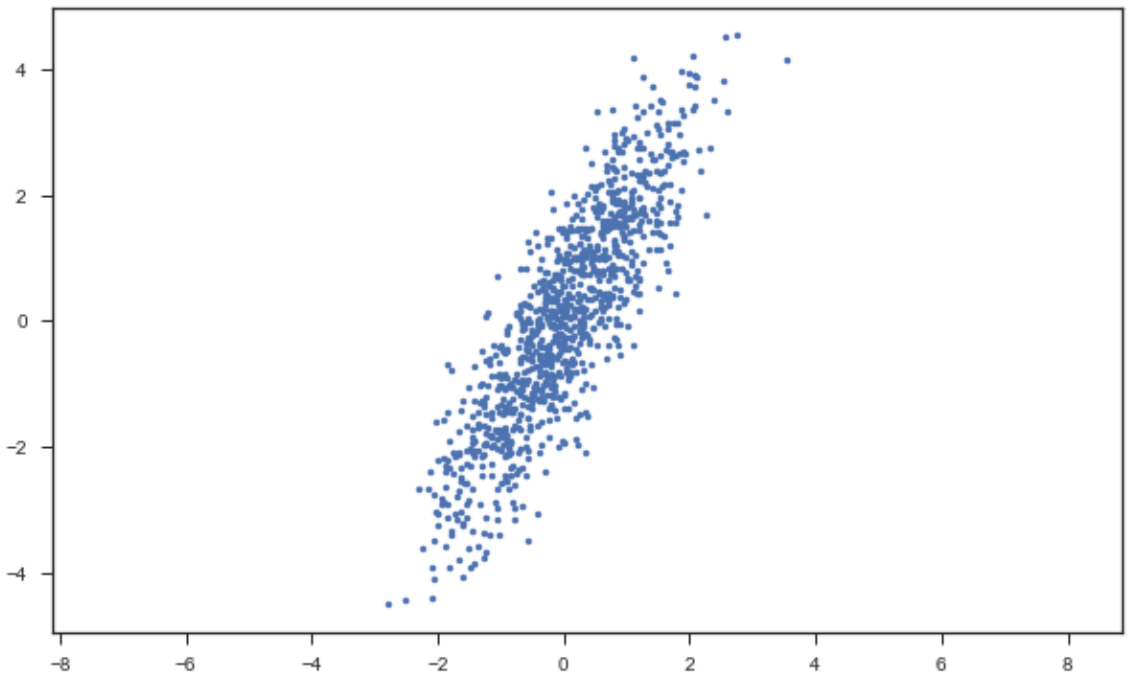
- Pictorial summary of PCA



$x^{(i)} \rightarrow$ projection onto unit vector $u \implies u^T x^{(i)} =$ distance from the origin along u

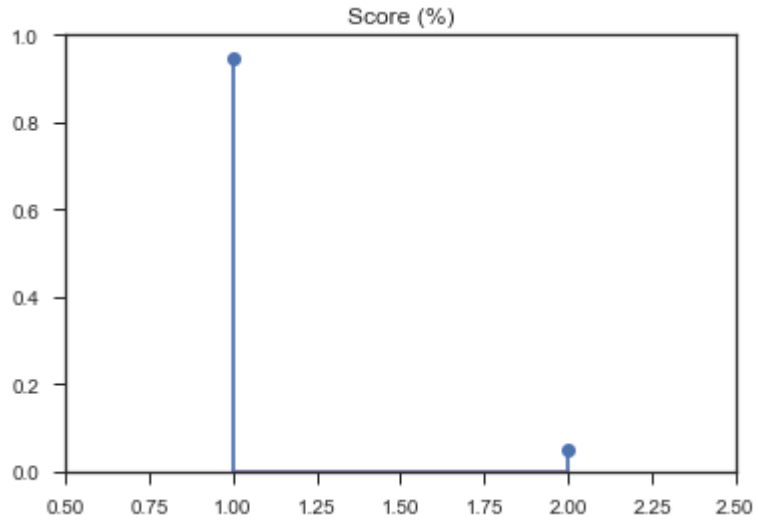
In [11]:

```
X = cPickle.load(open('./data_files/pca_example.pkl','rb'))

plt.figure(figsize=(10,6))
plt.plot(X[:, 0], X[:, 1],'b.')
plt.axis('equal')
plt.show()
```



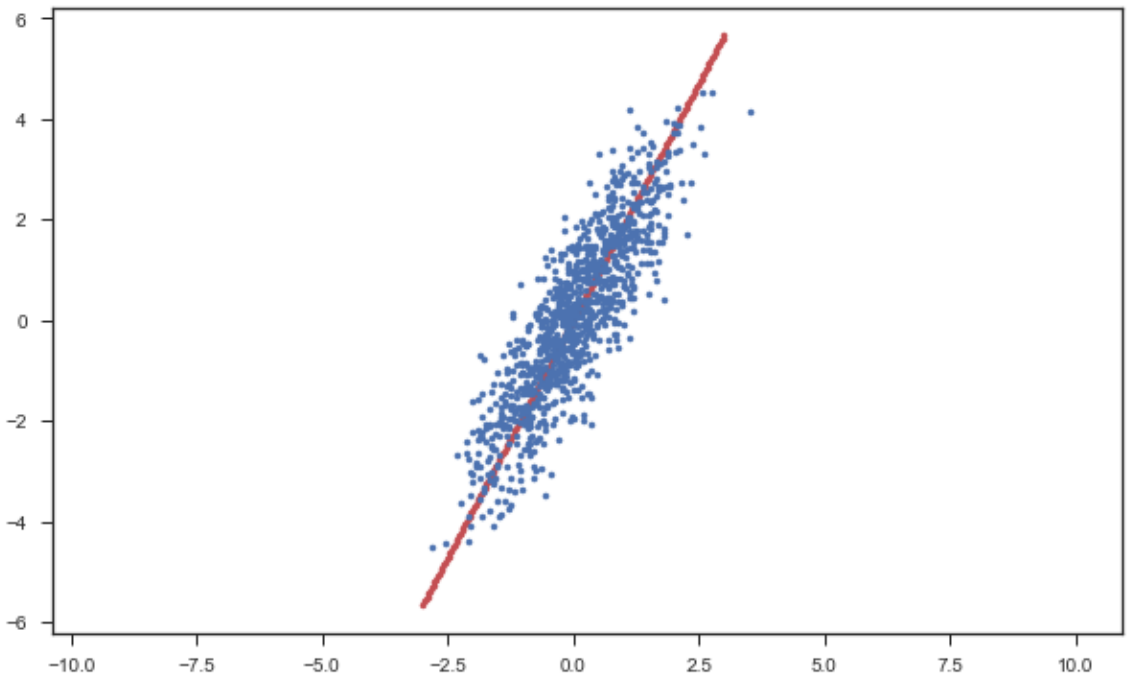
In [12]:

```
from sklearn.decomposition import PCA

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

plt.figure()
plt.stem(range(1,3),pca.explained_variance_ratio_)

plt.xlim([0.5, 2.5])
plt.ylim([0, 1])
plt.title('Score (%)')
plt.show()
```



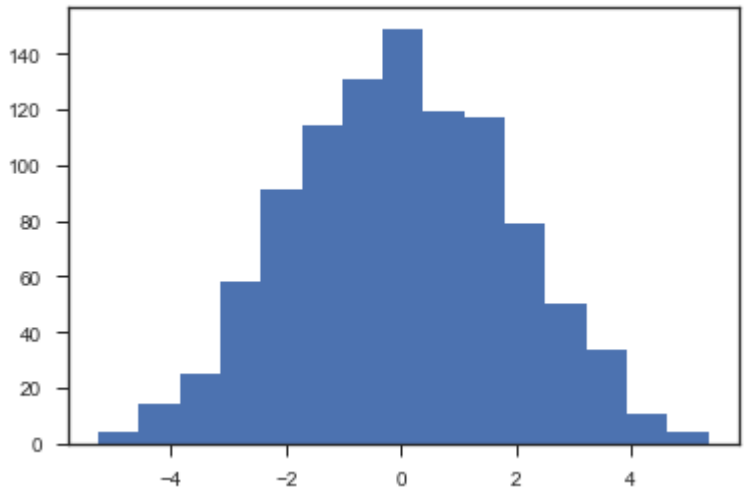
In [13]:

```
# Nomalization and calculate gradient
principal_axis = pca.components_[0, :]
u1 = principal_axis/(np.linalg.norm(principal_axis))
h = u1[1]/u1[0]

xp = np.linspace(-3,3,200)
yp = xp.dot(h)

plt.figure(figsize=(10,6))
plt.plot(xp, yp, 'r.')
plt.plot(X[:, 0], X[:, 1],'b.')
plt.axis('equal')
plt.show()
```



In [14]:

```
pca = PCA(n_components=1)
pca.fit(X)
reduced_coordinate = pca.fit_transform(X)
plt.hist(reduced_coordinate, 15)
plt.show()
```



In [15]:

```
%%javascript
$.getScript('https://kmahelona.github.io/ipython_notebook_goodies/ipython_notebook_toc.
js')
```