Lab 5. Principal Components Analysis (PCA) and Support Vector Machines (SVM)

Intro to Machine Learning
Fall 2018, Innopolis University

Lecture recap

- What is dimensionality reduction?
- How many dimensionality reduction methods do you know?
- What is transformation?
- What is PCA?
- How to select number of PCs?
- What is eigenvalue and eigenvector?
- What the difference between SVD and SVM?

Questions about the lecture

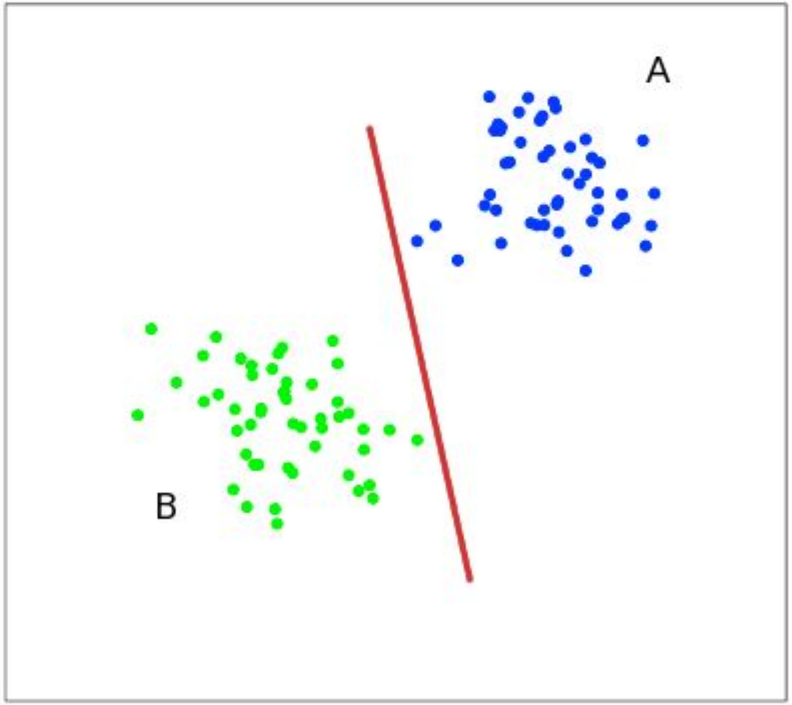
Was the material already familiar to you?

What new things have you learned?

What was hard to understand?

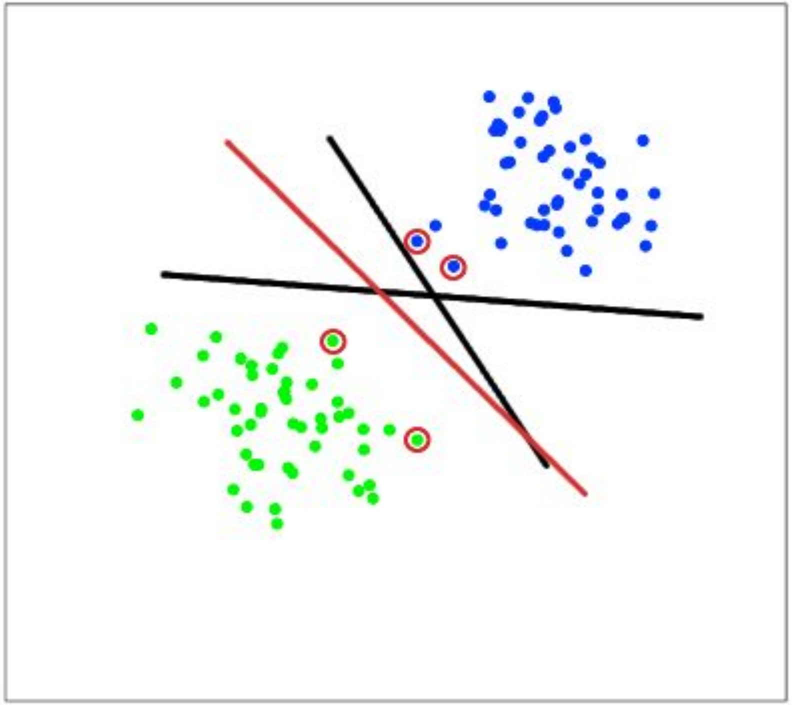
Support Vector Machine (SVM)

- We have N classes of data (N=2)
- We need to separate classes
  It means to find a classifier function
- We need to find an “optimal” classifier



Support Vector Machine (SVM)

Which classifier is the best?
How to find a best classifier?
What does make one classifier better than others?
How to find a condition of optimality?



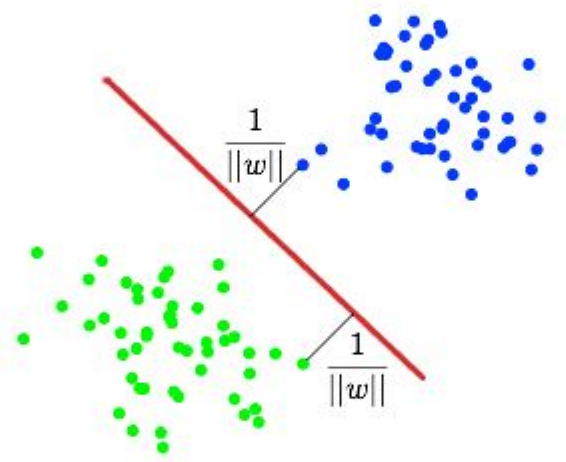
Support Vector Machine (SVM)

The classifier should be equidistant from each class.
We need to calculate the distance and maximize it.

Our classifier: $$F(\mathbf{x}) = \operatorname{sign}(\langle \mathbf{w}, \mathbf{x} \rangle + b)$$

F(x) = 1 are blue data points
F(x) = -1 are green data points

The distance is equal to $\frac{1}{||\mathbf{w}||}$ (*analytic geometry exercise)



Support Vector Machine (SVM)

We need to maximize $\frac{1}{||\mathbf{w}||}$ or minimize ||w||

$$\begin{cases} \arg\min\limits_{\mathbf{w},b} ||w||^2, \\ y_i(\langle \mathbf{w}, \mathbf{x}_i \rangle + b) \geqslant 1, \quad i = 1, \ldots, m. \end{cases}$$

Method of Lagrange multipliers can solve this problem.
We can find a minimum of ||w||
with respect to condition:

$$y_i(\langle \mathbf{w}, \mathbf{x}_i \rangle + b) \geqslant 1, \quad i = 1, \ldots, m.$$

Method of Lagrange Multipliers

$$L(x,\ y,\ \lambda) = f(x,\ y) - \lambda \cdot \psi(x,\ y).$$

$$\begin{cases} \left.\dfrac{\partial f}{\partial x}\right|_{(x_0,\ y_0)} & -\lambda_0 \cdot \left.\dfrac{\partial \psi}{\partial x}\right|_{(x_0,\ y_0)} & = & 0, \\ \left.\dfrac{\partial f}{\partial y}\right|_{(x_0,\ y_0)} & -\lambda_0 \cdot \left.\dfrac{\partial \psi}{\partial y}\right|_{(x_0,\ y_0)} & = & 0, \\ & -\psi(x_0,\ y_0) & = & 0. \end{cases}$$

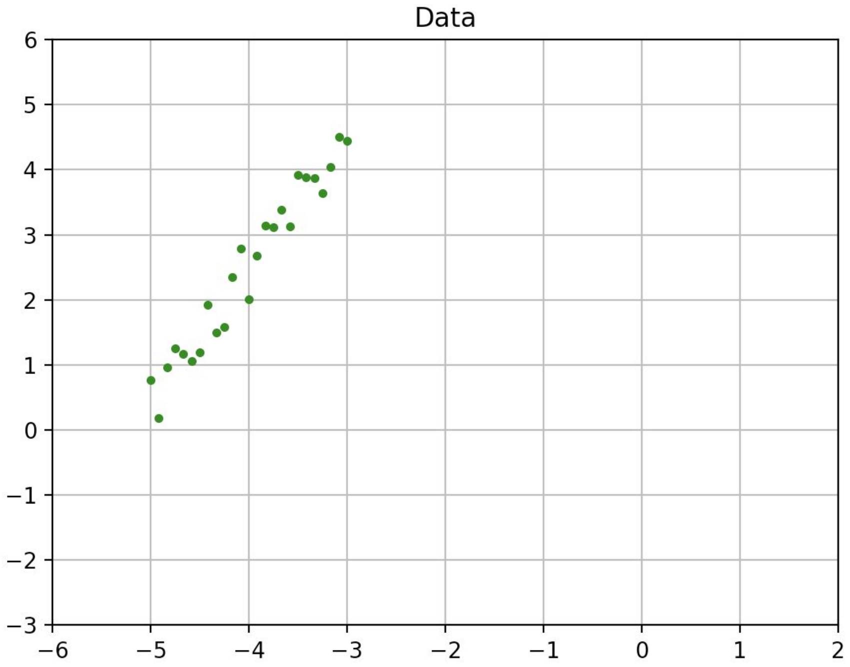
Principal Components Analysis (PCA)

There are 4 basic steps for PCA:

1. Generate data
2. Center data
3. Project data
4. Restore data

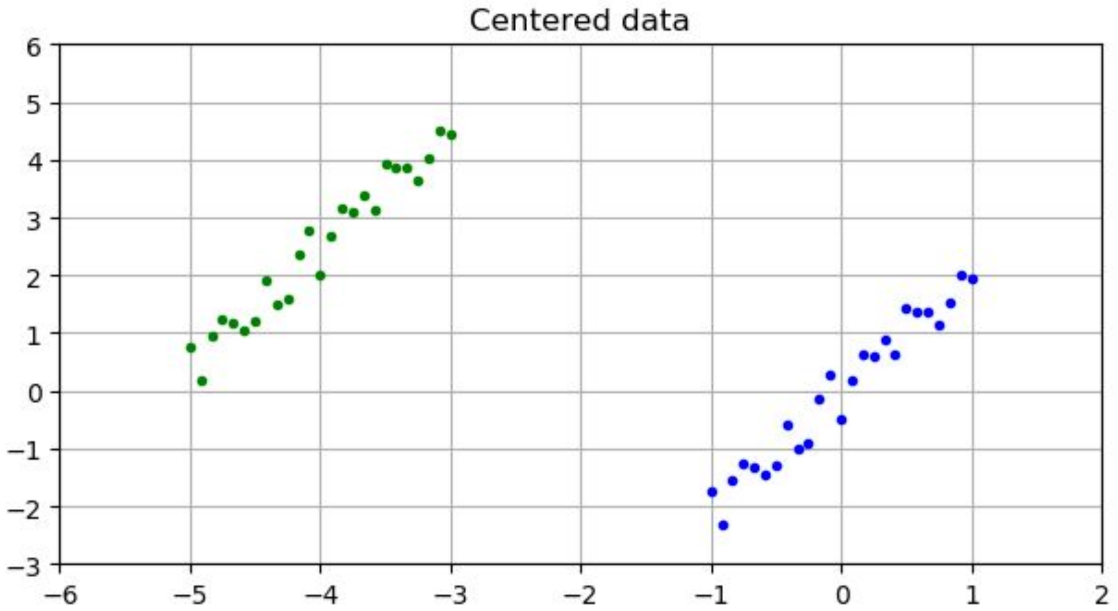
Generate data

```
N = 25
np.random.seed(10)
x = np.linspace(-5, -3, N)
y = 10 + 2*x + np.random.random(size=(N,))

plt.title("Data")
plt.plot(x, y, '.', color="green")
plt.axis([-6, 2, -3, 6])
plt.grid('True')
```



We first center it

```
x_centered = x - x.mean()
y_centered = y - y.mean()
```



Project data. Covariance matrix

$$Cov(X_i, X_j) = E\Big[\big(X_i - E(X_i)\big) \cdot \big(X_j - E(X_j)\big)\Big] = E(X_i X_j) - E(X_i) \cdot E(X_j)$$

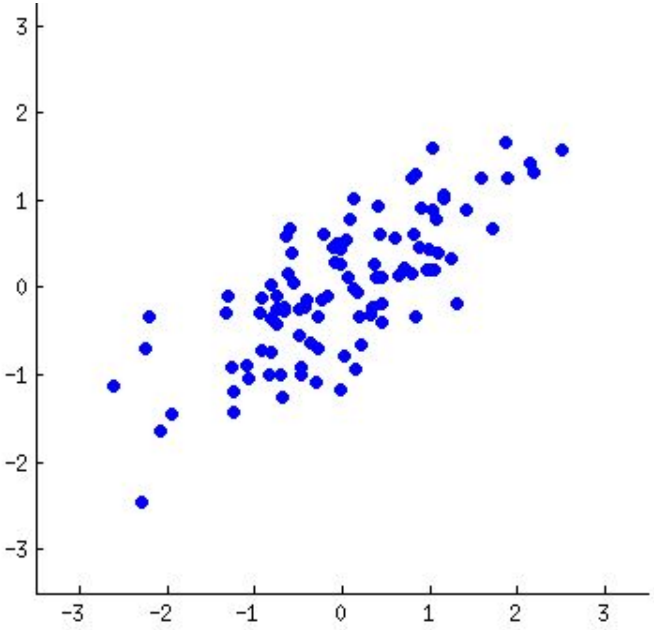
$$Cov(X_i, X_i) = Var(X_i)$$

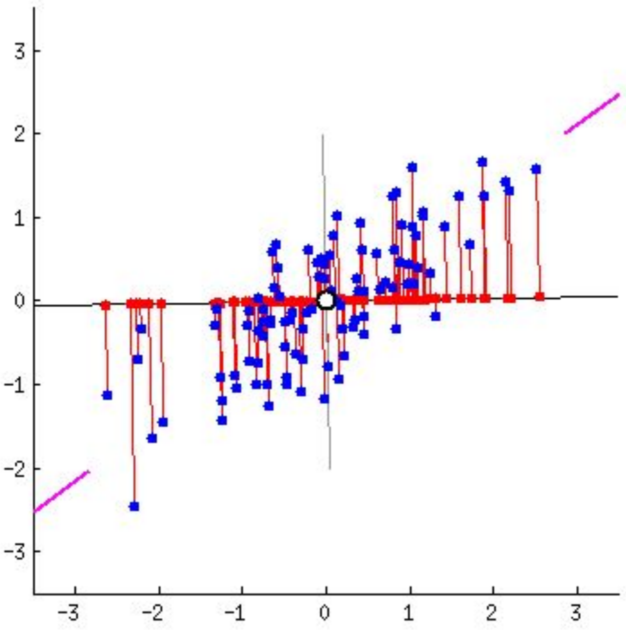
$$Cov(X_i, X_j) = E(X_i X_j)$$

How to calculate Biased and Unbiased Variance

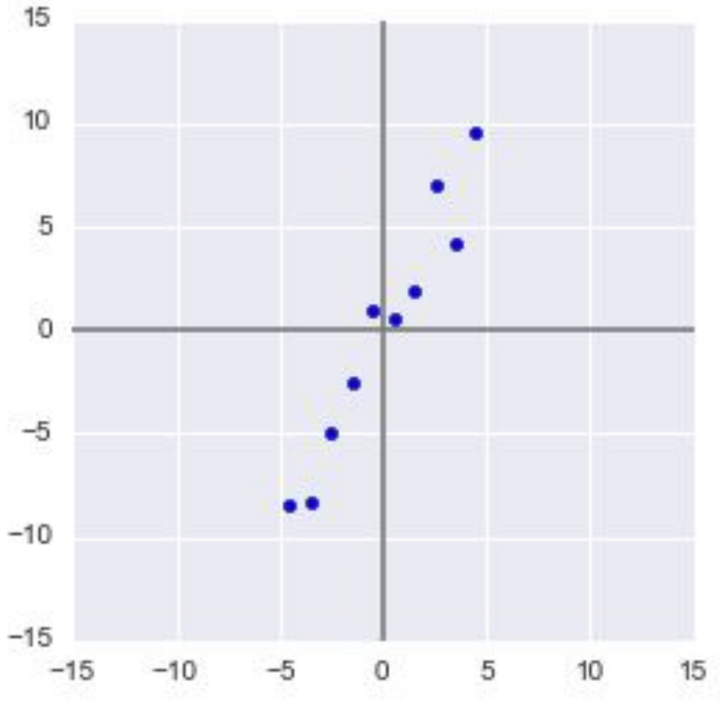
```
X.var()
((X - X.mean())**2).sum()/N
((X - X.mean())**2).mean()
(X**2).mean() - X.mean()**2

X.var(ddof=1)
((X - X.mean())**2).sum()/(N-1)
N/(N-1) * ((X - X.mean())**2).mean()
N/(N-1) * (X**2).mean() - N/(N-1) * X.mean()**2
```

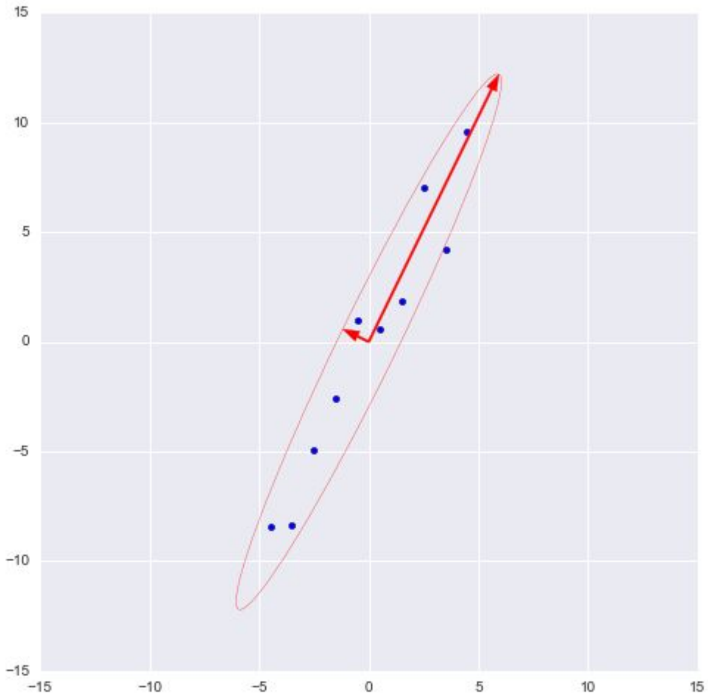
Project data. Some Intuition about projections





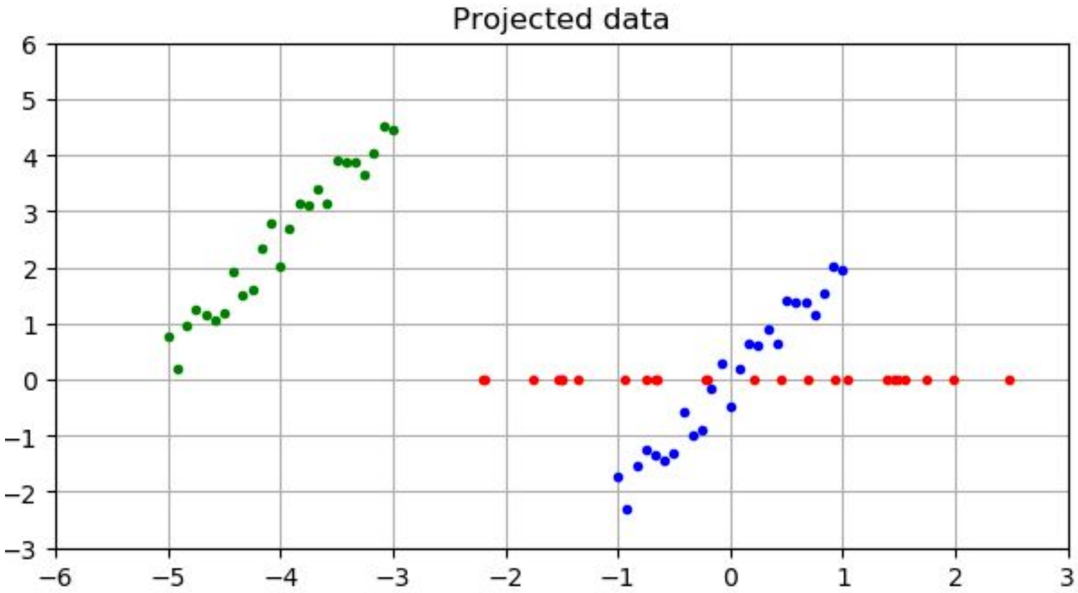
Project data. Some Intuition about eigenvectors





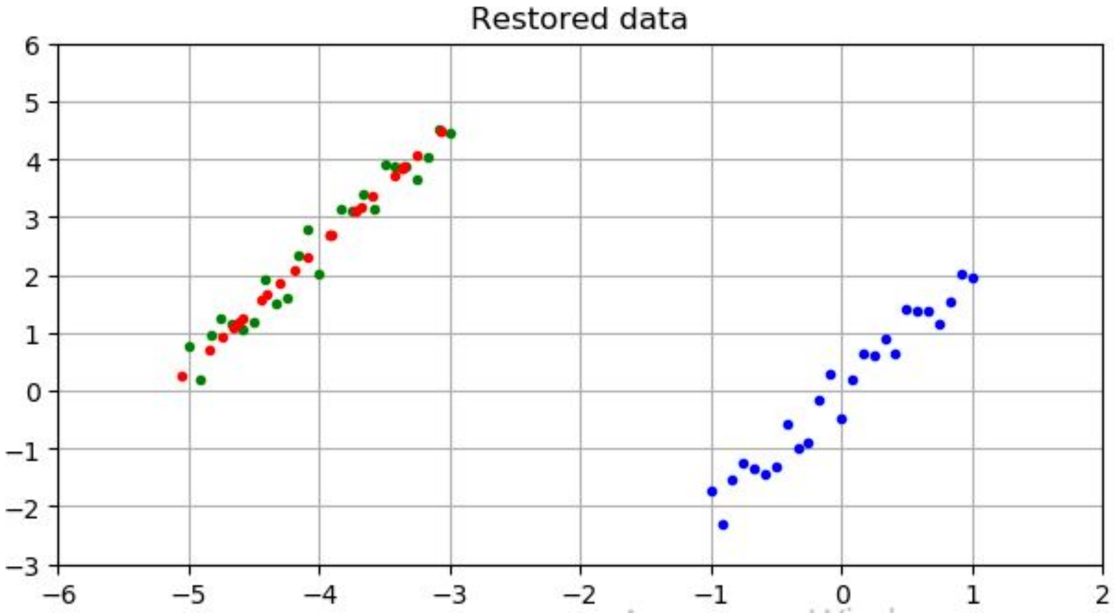
Then using the first PC we project it to new dimension

```
data_centered = np.stack((x_centered, y_centered),axis=-1)
projected_data  = np.dot(data_centered, eig_vectors[:,0])
# eig_vectors were sorted according to decreasing eigenvalues
```



Then if we want to restore data

```
# x_y_restored is a dot product of projection and transposed eigenvector(s)
x_restored = x_y_restored[:,0] + mean_vector[0]
y_restored = x_y_restored[:,1] + mean_vector[1]
```



Scikit-learn. Iris

Iris dataset has a size [150, 4]

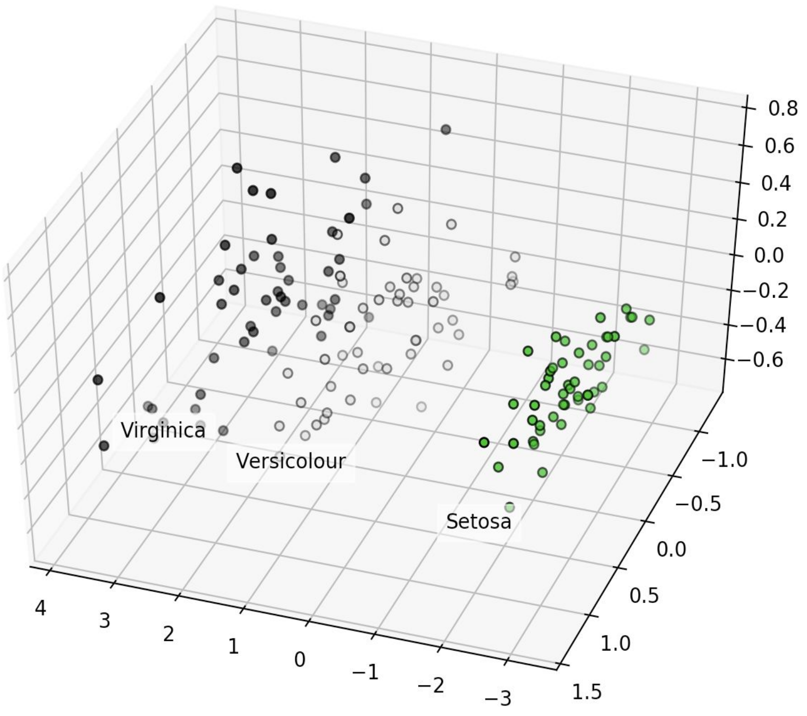
We will reduce it down to [150, 3] using PCA

Scikit-learn implementation

```
from sklearn.decomposition import PCA
pca = PCA(n_components=1)
x_PCA = pca.fit_transform(X)

print(x_PCA.T)
print(projected_data_local)
```

Scikit-learn. Iris

```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from sklearn import decomposition
from sklearn import datasets

np.random.seed(5)
iris = datasets.load_iris()
X = iris.data
y = iris.target
```



Scikit-learn. Iris

```
### MAIN PART ###

pca = decomposition.PCA(n_components=3)
pca.fit(X)
X_pca = pca.transform(X)
```

Scikit-learn. Iris

```
fig = plt.figure(1, figsize=(4, 3))
ax = Axes3D(fig, rect=[0, 0, .95, 1])

for name, label in [('Setosa', 0), ('Versicolour', 1), ('Virginica', 2)]:
    ax.text3D( X_pca[y == label, 0].mean(),
               X_pca[y == label, 1].mean() + 1.5,
               X_pca[y == label, 2].mean(),
               name)
y = np.choose(y, [1, 2, 0]).astype(np.float)
ax.scatter(X_pca[:, 0], X_pca[:, 1], X_pca[:, 2], c=y, cmap=plt.cm.nipy_spectral)

plt.show()
```

Homework

- Part I (from scratch)
  - Load Iris dataset
  - Center data
  - Project data to 2 dimensions
  - Plot results
  - Restore data
- Part II (sklearn API)
  - Do the same using sklearn libraries
  - Plot results (you should get the picture from the slide)
  - Compare results