Basic machine learning background with Python scikit-learn

Usman Roshan

Python

- An object oriented interpreter-based programming language
- Basic essentials
  - Data types are numbers, strings, lists, and dictionaries (hash-tables)
  - For loops and conditionals
  - Functions
  - Lists and hash-tables are references (like pointers in C)
  - All variables are passed by value

Python

- Simple Python programs

```
str = "1  4  3   5  7"

a = str.split();

l = len(a)
for i in range(0, l, 1):
      if(int(a[i]) > 3):
            print(a[i])
print('\n')
```

```
l = []
l2 = [1, 10]
l.append(l2)
l2 = [10, 100]
l.append(l2)
l2 = [20, 200]
l.append(l2)

rows = 3;
cols = 2;

for i in range(0, rows, 1):
       for j in range(0, cols, 1):
              print(l[i][j])
              print(" ")
       print('\n')
```

```
f = open("inputdata")

data = []
i=0;
l = f.readline()
while(l != ''):
      a = l.split()
      l2 = []
      for j in range(0, len(a), 1):
            l2.append(a[j])
      data.append(l2)
      l = f.readline()

rows = len(data)
cols = len(data[0])

print("row=", rows, "cols=", cols)

for i in range(0, rows, 1):
      print(data[i])
```

Data science

- Data science
  - Simple definition: reasoning and making decisions from data
  - Contains machine learning, statistics, algorithms, programming, big data
- Basic machine learning problems
  - Classification
    - Linear methods
    - Non-linear
  - Feature selection
  - Clustering (unsupervised learning)
  - Visualization with PCA

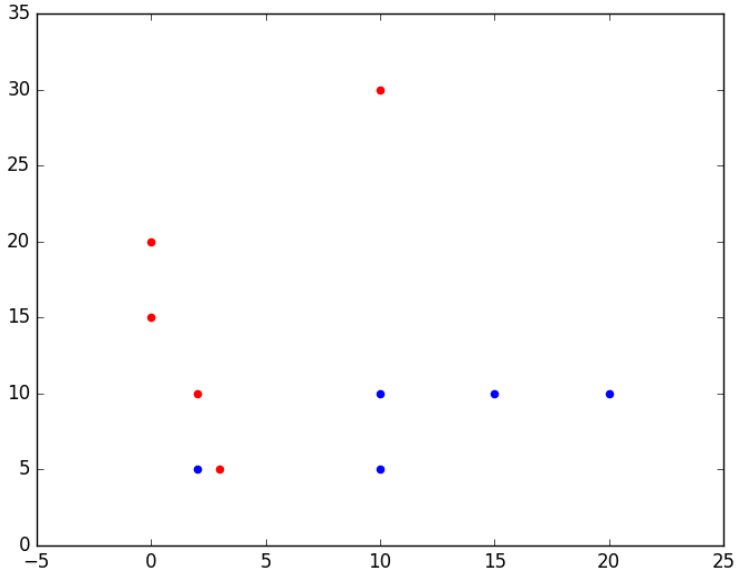
Python scikit-learn

- Popular machine learning toolkit in Python http://scikit-learn.org/stable/
- Requirements
  - Anaconda
  - Available from https://www.continuum.io/downloads
  - Includes numpy, scipy, and scikit-learn (former two are necessary for scikit-learn)

Data

- We think of data as vectors in a fixed dimensional space
- For example

Heart disease	Cigarettes per day	Exercise per day (mins)
1	10	10
1	2	5
1	20	10
1	10	5
1	15	10
0	10	30
0	2	10
0	3	5
0	0	20
0	0	15

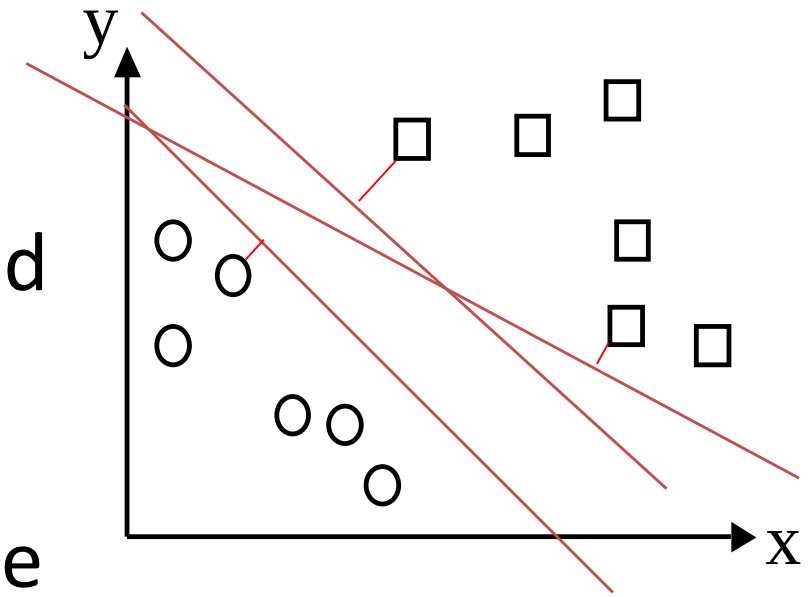


Classification

- Widely used task: given data determine the class it belongs to. The class will lead to a decision or outcome.
- Used in many different places:
  - DNA and protein sequence classification
  - Insurance
  - Weather
  - Experimental physics: Higgs Boson determination

Linear models

- We think of a linear model as a hyperplane in space
- The margin is the minimum distance of all closest point (misclassified have negative distance)
- The support vector machine is the hyperplane with largest margin



Support vector machine: optimally separating hyperplane

In practice we allow for error terms in case there is no hyperplane.

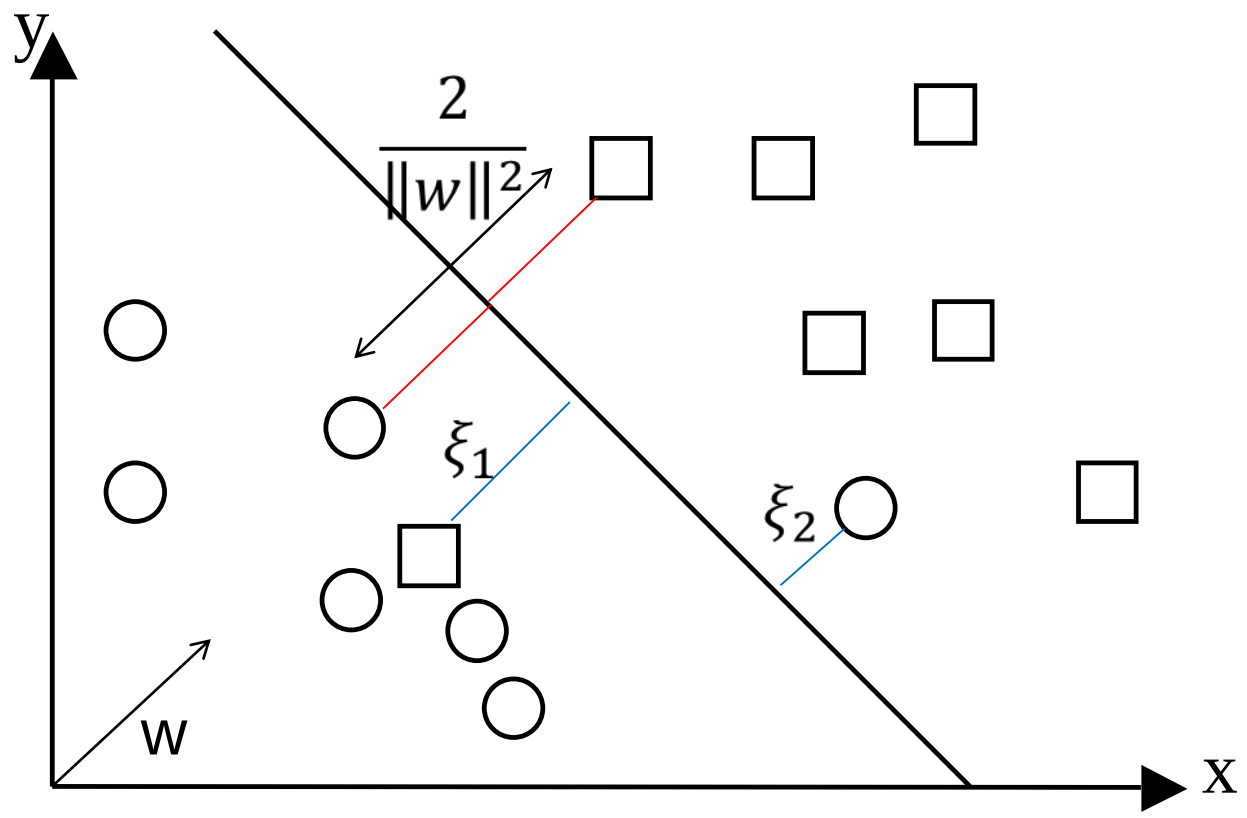
$$\min_{w,w_0,x_i}\left(\frac{1}{2}\|w\|^2 + C\sum_i x_i\right)$$

$$s.t.$$

$$y_i(w^T x + w_0) \geq 1 - x_i$$

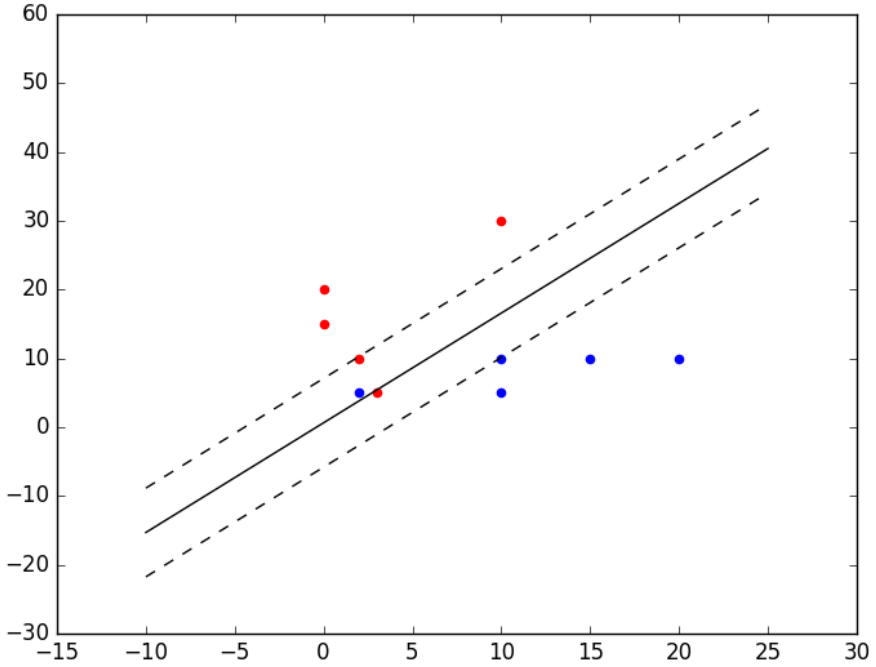
$$x_i \geq 0$$

Optimally separating hyperplane with errors



SVM on simple data

- Run SVM on example data shown earlier
- Solid line is SVM and dashed indicates margin



SVM in scikit-learn

- Dataset are taken from the UCI machine learning repository
- Learn an SVM model on training data
- Which parameter settings?
  - C: tradeoff between error and model complexity (margin)
  - max_iter: depth of the gradient descent algorithm
- Predict on test data

SVM in scikit-learn

Analysis of SVM program on breast cancer data

```
import numpy as np
from sklearn import svm

f = open("bc.train.0")
data = np.loadtxt(f)
train = data[:,1:]
trainlabels = data[:,0]

f = open("bc.test.0")
data = np.loadtxt(f)
test = data[:,1:]
testlabels = data[:,0]

clf = svm.LinearSVC()
clf.fit(train,trainlabels)
prediction = clf.predict(test)
print(prediction)

err = 0
for i in range(0, len(prediction), 1):
    if(prediction[i] != testlabels[i]):
        err += 1
err = err/len(testlabels)
```

Non-linear classification

- In practice some datasets may not be classifiable.
- Remember this may not be a big deal because the test error is more important than the train one

Non-linear classification

- Neural networks
  - Create a new representation of the data where it is linearly separable
  - Large networks leads to deep learning
- Decision trees
  - Use several linear hyperplanes arranged in a tree
  - Ensembles of decision trees are state of the art such as random forest and boosting

Decision tree

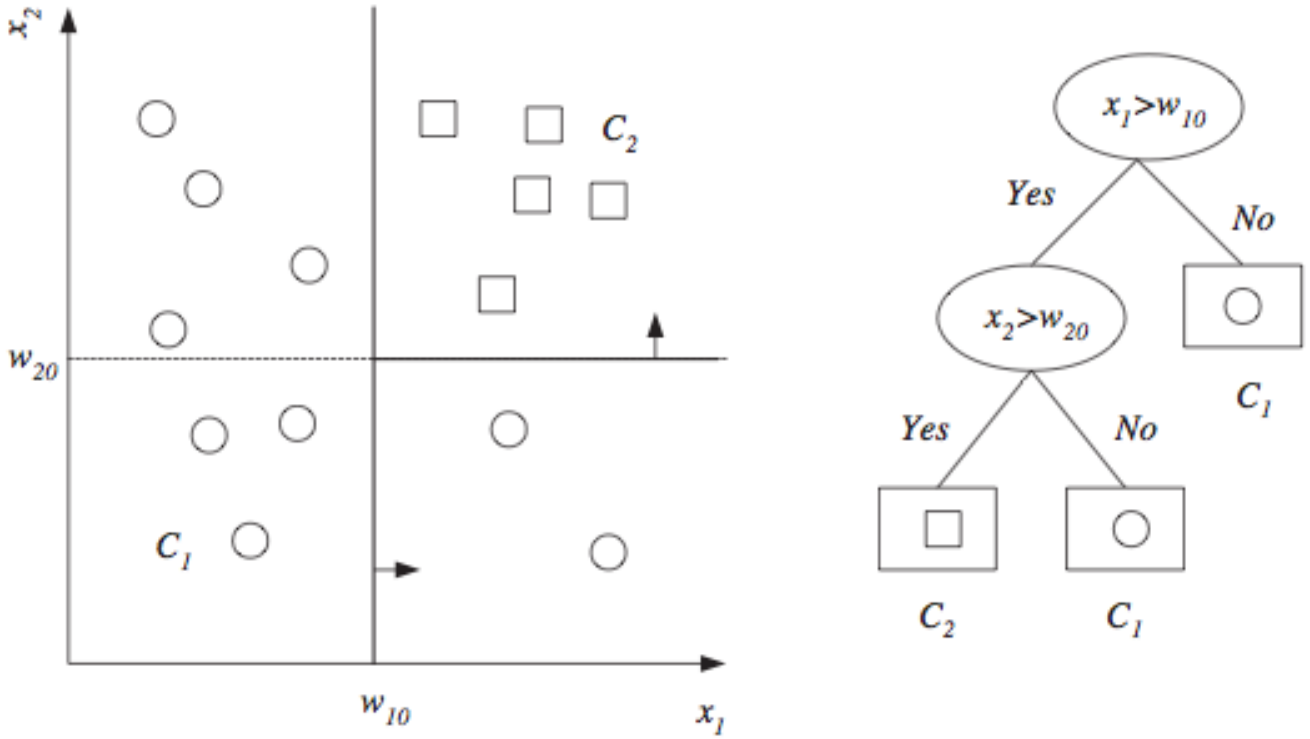


Figure 9.1 Example of a dataset and the corresponding decision tree. Oval nodes are the decision nodes and rectangles are leaf nodes. The univariate decision node splits along one axis, and successive splits are orthogonal to each other. After the first split, $\{\boldsymbol{x}|x_1 < w_{10}\}$ is pure and is not split further.

From Alpaydin, 2010

Combining classifiers by bagging

- A single decision tree can overfit the data and have poor generalization (high error on test data). We can relieve this by bagging
- Bagging
  - Randomly sample training data by bootstrapping
  - Determine classifier C_i on sampled data
  - Goto step 1 and repeat *m* times
  - For final classifier output the majority vote
- Similar to tree bagging
  - Compute decision trees on bootstrapped datasets
  - Return majority vote

Variance reduction by voting

- What is the variance of the output of k classifiers?

$$\mathrm{Var}(y) = \frac{1}{L^2}\mathrm{Var}\left(\sum_j d_j\right) = \frac{1}{L^2}\left[\sum_j \mathrm{Var}(d_j) + 2\sum_j \sum_{i<j} \mathrm{Cov}(d_j, d_i)\right]$$

- Thus we want classifiers to be independent to minimize variance
- Given independent binary classifiers each with accuracy > ½ the majority vote accuracy increases as we increase the number of classifiers (Hansen and Salamon, *IEEE Transactions of Pattern Analysis and Machine Intelligence*, 1990)

Random forest

- In addition to sampling datapoints (feature vectors) we also sample features (to increase independence among classifiers)
- Compute many decision trees and output majority vote
- Can also rank features
- Alternative to bagging is to select datapoints with different probabilities that change in the algorithm (called boosting)

Decision tree and random forest in scikit-learn

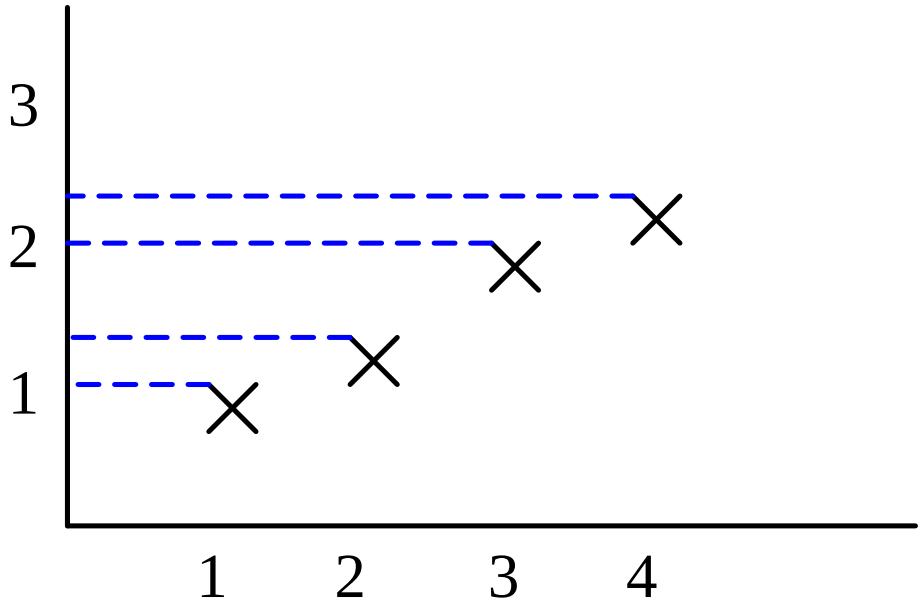
- Learn a decision tree and random forest on training data
- Which parameter settings?
  - Decision tree:
    - Depth of tree
  - Random forest:
    - Number of trees
    - Percentage of columns
- Predict on test data

Decision tree and random forest in scikit-learn

```
from sklearn import tree
from sklearn.ensemble import RandomForestClassifier
import numpy as np

f = open("bc.train.0")
data = np.loadtxt(f)
train = data[:,1:]
trainlabels = data[:,0]

f = open("bc.test.0")
data = np.loadtxt(f)
test = data[:,1:]
testlabels = data[:,0]

clf = tree.DecisionTreeClassifier()
clf.fit(train,trainlabels)
prediction = clf.predict(test)

rfc = RandomForestClassifier(n_estimators=100)
rfc.fit(train,trainlabels)
prediction2 = rfc.predict(test)

err = 0
err2 = 0
for i in range(0, len(prediction), 1):
    if(prediction[i] != testlabels[i]):
        err += 1
    if(prediction2[i] != testlabels[i]):
        err2 += 1
err = err/len(testlabels)
err2 = err2/len(testlabels)
print(err,err2)
```

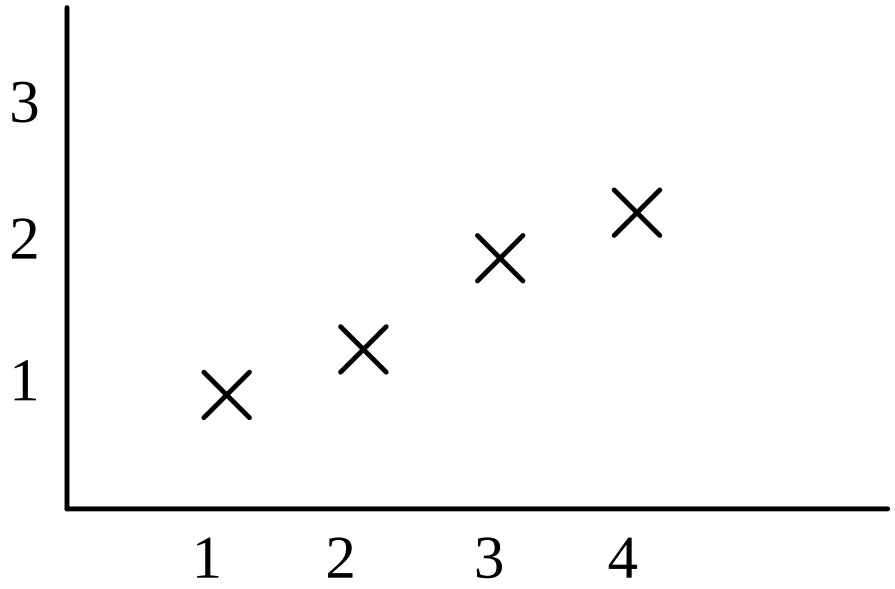
Data projection

- What is the mean and variance here?



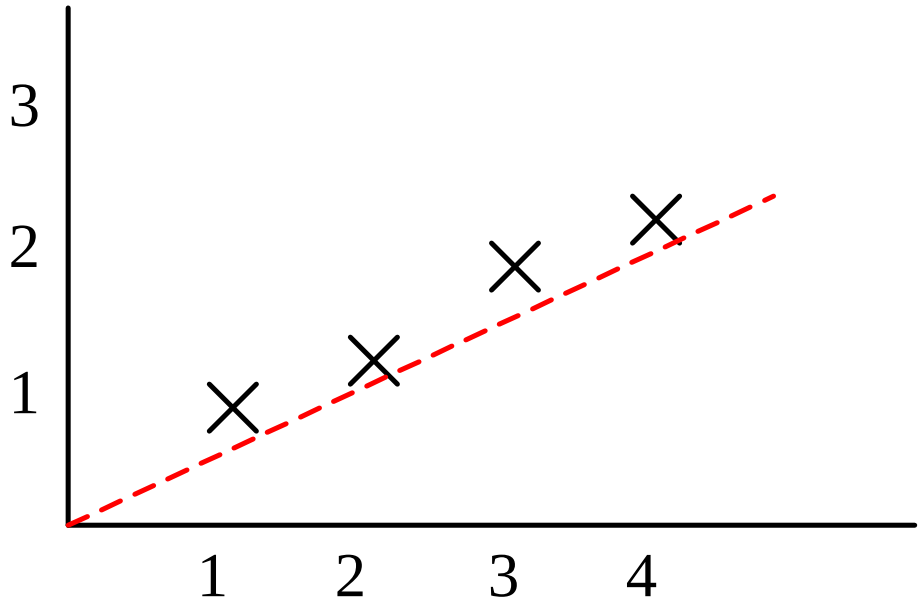
Data projection

- Which line maximizes variance?



Data projection

- Which line maximizes variance?



Principal component analysis

- Find vector w of length 1 that maximizes variance of projected data

PCA optimization problem

$$\arg\max_{w} \frac{1}{n} \sum_{i=1}^{n} (w^T x_i - w^T m)^2 \text{ subject to } w^T w = 1$$

The optimization criterion can be rewritten as

$$\arg\max_{w} \frac{1}{n} \sum_{i=1}^{n} (w^T (x_i - m))^2 =$$

$$\arg\max_{w} \frac{1}{n} \sum_{i=1}^{n} (w^T (x_i - m))^T (w^T (x_i - m)) =$$

$$\arg\max_{w} \frac{1}{n} \sum_{i=1}^{n} ((x_i - m)^T w)(w^T (x_i - m)) =$$

$$\arg\max_{w} \frac{1}{n} \sum_{i=1}^{n} w^T (x_i - m)(x_i - m)^T w =$$

$$\arg\max_{w} w^T \frac{1}{n} \sum_{i=1}^{n} (x_i - m)(x_i - m)^T w =$$

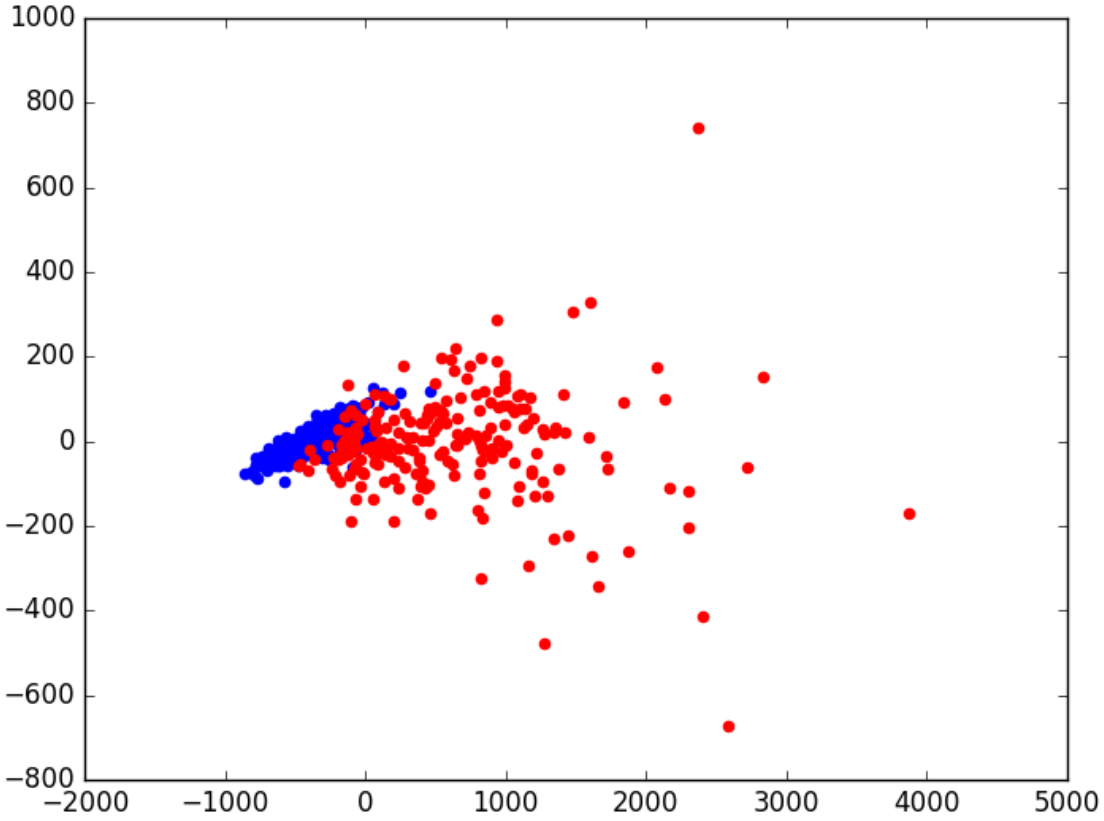
$$\arg\max_{w} w^T \Sigma w \text{ subject to } w^T w = 1$$

Dimensionality reduction and visualization with PCA

```
from matplotlib import pyplot as plt
from sklearn.decomposition import PCA

import numpy as np

f = open("bc")
data = np.loadtxt(f)

X = data[:,1:]
Y = data[:,0]

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

y1 = []
y2 = []
for i in range(0, len(newdata), 1):
        if(Y[i] == 1):
                y1.append(newdata[i][0])
                y2.append(newdata[i][1])

plt.scatter(y1, y2, color='blue')

y1 = []
y2 = []
for i in range(0, len(newdata), 1):
        if(Y[i] == -1):
                y1.append(newdata[i][0])
                y2.append(newdata[i][1])

plt.scatter(y1, y2, color='red')

plt.show()
```

Dimensionality reduction and visualization with PCA

PCA plot of breast cancer data (output of program in previous slide)



Unsupervised learning - clustering

- K-means: popular fast program for clustering data
- Objective: find k clusters that minimize Euclidean distance of points in each cluster to their centers (means)

$$\sum_{i=1}^{2} \sum_{x_j \in C_i} \| x_j - m_i \|^2$$

K-means algorithm for two clusters

Input: $x_i \in R^d, i = 1 \ldots n$

Algorithm:

1. Initialize: assign x_i to C_1 or C_2 with equal probability and compute means:

$$m_1 = \frac{1}{|C_1|} \sum_{x_i \in C_1} x_i \qquad m_2 = \frac{1}{|C_2|} \sum_{x_i \in C_2} x_i$$

2. Recompute clusters: assign x_i to C_1 if $\|x_i - m_1\| < \|x_i - m_2\|$, otherwise assign to C_2
3. Recompute means m_1 and m_2
4. Compute objective

$$\sum_{i=1}^{2} \sum_{x_j \in C_i} \| x_j - m_i \|^2$$

5. Compute objective of new clustering. If difference is smaller than α then stop, otherwise go to step 2.

K-means in scikit-learn

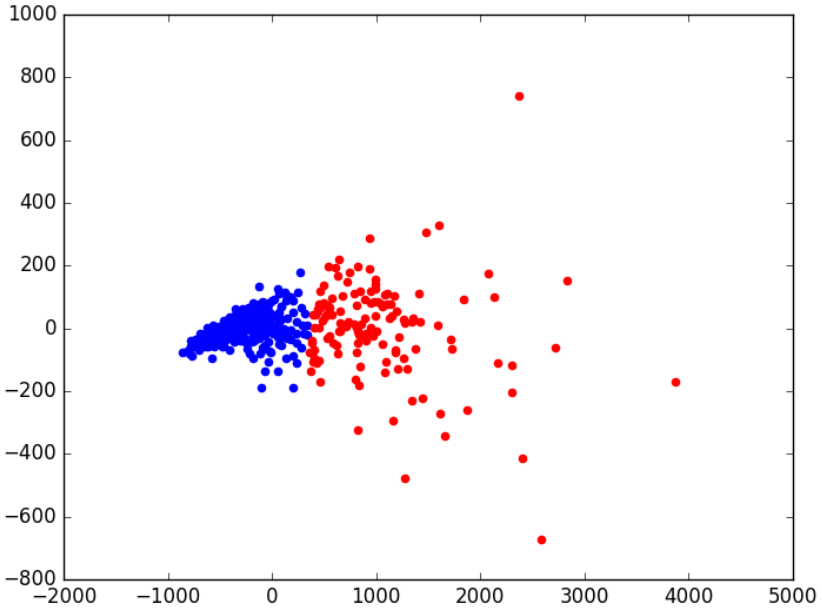
```
from sklearn.cluster import KMeans
from matplotlib import pyplot as plt
import numpy as np

f = open("bc")
data = np.loadtxt(f)

X = data[:,1:]
Y = data[:,0]

clustering = KMeans(n_clusters=2,init='random').fit(X)

err = 0
for i in range(0, len(X), 1):
        if(clustering.labels_[i] != Y[i]):
                err += 1
err /= len(X)
print(err)
```

K-means PCA plot in scikit-learn

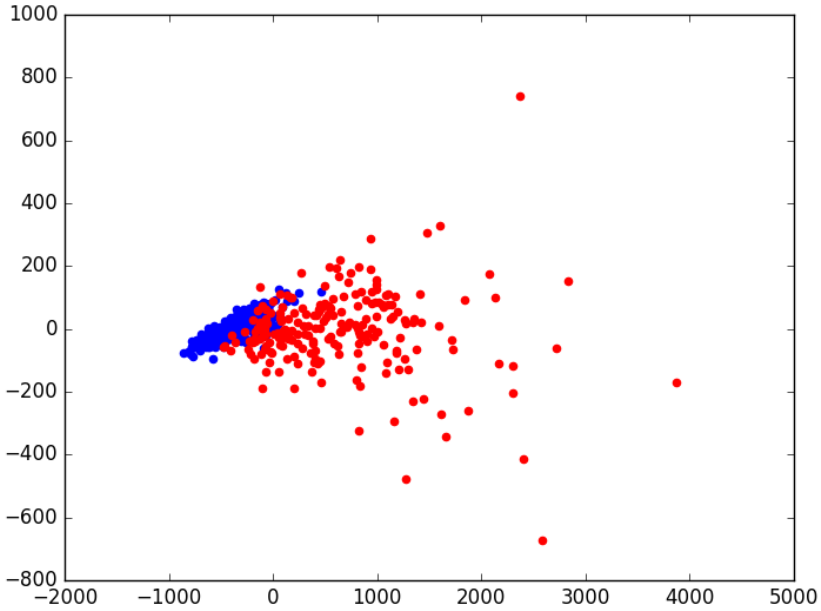
```
from sklearn.cluster import KMeans
from matplotlib import pyplot as plt
import numpy as np
from sklearn.decomposition import PCA

f = open("bc")
data = np.loadtxt(f)
X = data[:,1:]
Y = data[:,0]

pca = PCA(n_components=2)
pca.fit(X)
newdata = pca.transform(X)
clustering = KMeans(n_clusters=2,init='random').fit(X)

x = []
y = []
for i in range(0, len(newdata), 1):
        if(clustering.labels_[i] == 0):
                x.append(newdata[i][0])
                y.append(newdata[i][1])

plt.scatter(x, y, color='blue')

x = []
y = []
for i in range(0, len(newdata), 1):
        if(clustering.labels_[i] == 1):
                x.append(newdata[i][0])
                y.append(newdata[i][1])

plt.scatter(x, y, color='red')
plt.show()
```

K-means PCA plot in scikit-learn

PCA plot of breast cancer data colored by k-means labels



PCA plot of breast cancer data colored by true labels



Conclusion

- We saw basic data science and machine learning tasks in Python scikit-learn
- Can we handle very large datasets in Python scikit-learn? Yes
  - For space use array from numpy to use a byte for a char and 4 for float and int. Otherwise more space is used because Python is object oriented
  - For speed use stochastic gradient descent in scikit-learn (doesn't come with mini-batch though) and mini-batch k-means
- Deep learning stuff: Keras