# 1

# Introduction to Practical Machine Learning using Python

## Preparing, manipulating and visualizing data – NumPy, pandas and matplotlib tutorials

### Understanding the pandas module

### Exploring data

```
In [8]: obj = pd.Series([3,5,-2,1])
        obj

Out[8]: 0    3
        1    5
        2   -2
        3    1
        dtype: int64
```

```
In [9]: obj.values
Out[9]: array([ 3,  5, -2,  1])
In [10]: obj.index
Out[10]: Int64Index([0, 1, 2, 3], dtype='int64')
```

```
In [11]: obj *2
Out[11]: 0     6
         1    10
         2    -4
         3     2
         dtype: int64
In [12]: obj[obj>2]
Out[12]: 0    3
         1    5
         dtype: int64
```

In [19]:
```
data = {'a': 30, 'b': 70, 'c': 160, 'd': 5}
obj = pd.Series(data)
obj
```

Out[19]:
```
a     30
b     70
c    160
d      5
dtype: int64
```

In [20]:
```
index = ['a','b','c','d','g']
obj = pd.Series(data, index=index)
obj
```

Out[20]:
```
a     30
b     70
c    160
d      5
g    NaN
dtype: float64
```

```
In [16]: pd.isnull(obj)

Out[16]: a     False
         b     False
         c     False
         d     False
         dtype: bool

In [17]: pd.notnull(obj)

Out[17]: a     True
         b     True
         c     True
         d     True
         dtype: bool
```

```
In [4]: data = pd.read_csv("data_example/ad-dataset/ad.data",header=None)
```

In [5]:
```
data.describe()
```

Out[5]:

| | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | ... |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **count** | 3279.000000 | 3279.000000 | 3279.000000 | 3279.000000 | 3279.000000 | 3279.000000 | 3279.000000 | 3279.000000 | 3279.000000 | 3279.000000 | ... |
| **mean** | 0.004270 | 0.011589 | 0.004575 | 0.003355 | 0.003965 | 0.011589 | 0.003355 | 0.004880 | 0.009149 | 0.004575 | ... |
| **std** | 0.065212 | 0.107042 | 0.067491 | 0.057831 | 0.062850 | 0.107042 | 0.057831 | 0.069694 | 0.095227 | 0.067491 | ... |
| **min** | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | ... |
| **25%** | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | ... |
| **50%** | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | ... |
| **75%** | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | ... |
| **max** | 1.000000 | 1.000000 | 1.000000 | 1.000000 | 1.000000 | 1.000000 | 1.000000 | 1.000000 | 1.000000 | 1.000000 | ... |

8 rows × 1554 columns

In [25]:
```
data.columns
```

Out[25]:
```
Int64Index([   0,    1,    2,    3,    4,    5,    6,    7,    8,    9,
            ...
            1549, 1550, 1551, 1552, 1553, 1554, 1555, 1556, 1557, 1558],
           dtype='int64', length=1559)
```

In [26]:
```
data.dtypes
```

Out[26]:
```
0        object
1        object
2        object
3        object
4         int64
5         int64
          ...
1557      int64
1558     object
dtype: object
```

```
In [6]: data[1]

Out[6]: 0        125
        1        468
        29       234
                 ...
        3277       ?
        3278      40
        Name: 1, dtype: object
```

```
In [27]: data[[1,20]]
```

Out[27]:

| | 1 | 20 |
|---|---|---|
| **0** | 125 | 0 |
| **1** | 468 | 0 |
| **2** | 230 | 0 |
| **3** | 468 | 0 |
| **4** | 468 | 0 |
| **...** | ... | ... |
| **3277** | ? | 0 |
| **3278** | 40 | 0 |

3279 rows × 2 columns

```
In [28]: data[1].head()

Out[28]: 0     125
         1     468
         2     230
         3     468
         4     468
         Name: 1, dtype: object
```

```
In [29]: data[1].head(10)

Out[29]: 0     125
         1     468
         2     230
         3     468
         4     468
         5     468
         6     460
         7     234
         8     468
         9     468
         Name: 1, dtype: object
```

```
In [7]: data[1:3]
```

Out[7]:

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | ... | 1549 | 1550 | 1551 | 1552 | 1553 | 1554 | 1555 | 1556 | 1557 | 1558 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **1** | 57 | 468 | 8.2105 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ad. |
| **2** | 33 | 230 | 6.9696 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ad. |

2 rows × 1559 columns

## Manipulate data

```
In [31]: data[data[1]> 0].head(4)
```

Out[31]:

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | ... | 1549 | 1550 | 1551 | 1552 | 1553 | 1554 | 1555 | 1556 | 1557 | 1558 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 125 | 125 | 1.0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ad. |
| **1** | 57 | 468 | 8.2105 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ad. |
| **2** | 33 | 230 | 6.9696 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ad. |
| **3** | 60 | 468 | 7.8 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ad. |

4 rows × 1559 columns

In [32]:
```
data[(data[1]> 0) & (data[1558]=='ad.')].head(4)
```

Out[32]:

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | ... | 1549 | 1550 | 1551 | 1552 | 1553 | 1554 | 1555 | 1556 | 1557 | 1558 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 125 | 125 | 1.0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ad. |
| **1** | 57 | 468 | 8.2105 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ad. |
| **2** | 33 | 230 | 6.9696 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ad. |
| **3** | 60 | 468 | 7.8 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ad. |

4 rows × 1559 columns

In [33]:
```
data.ix[:3]
```

Out[33]:

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | ... | 1549 | 1550 | 1551 | 1552 | 1553 | 1554 | 1555 | 1556 | 1557 | 1558 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 125 | 125 | 1.0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ad. |
| **1** | 57 | 468 | 8.2105 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ad. |
| **2** | 33 | 230 | 6.9696 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ad. |
| **3** | 60 | 468 | 7.8 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ad. |

4 rows × 1559 columns

In [34]:
```
data.iloc[:3]
```

Out[34]:

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | ... | 1549 | 1550 | 1551 | 1552 | 1553 | 1554 | 1555 | 1556 | 1557 | 1558 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 125 | 125 | 1.0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ad. |
| **1** | 57 | 468 | 8.2105 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ad. |
| **2** | 33 | 230 | 6.9696 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ad. |

3 rows × 1559 columns

In [35]:
```
data.loc[:3]
```

Out[35]:

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | ... | 1549 | 1550 | 1551 | 1552 | 1553 | 1554 | 1555 | 1556 | 1557 | 1558 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 125 | 125 | 1.0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ad. |
| **1** | 57 | 468 | 8.2105 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ad. |
| **2** | 33 | 230 | 6.9696 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ad. |
| **3** | 60 | 468 | 7.8 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ad. |

4 rows × 1559 columns

click to scroll output; double click to hide

```
In [ ]:
```

```
In [37]: data.ix[3,1]=0
```

```
In [38]: import random
         data.ix[0] = [random.randint(0,1) for r in xrange(1558)]+['ad.']
```

```
In [40]: row = [random.randint(0,1) for r in xrange(1558)]+['ad.']
         data = data.append(pd.Series(row,index = data.columns),ignore_index=True)
```

```
In [70]: data.loc[len(data)] = row
```

```
In [41]: data['newcolumn'] = 'test value'
         data.columns
```

```
Out[41]: Index([          0,          1,          2,          3,          4,
                           5,          6,          7,          8,          9,
                 ...
                      1550,       1551,       1552,       1553,       1554,
                      1555,       1556,       1557,       1558, u'newcolumn'],
                dtype='object', length=1560)
```

```
In [56]: data = data.drop('newcolumn', 1)
         data.columns
```

```
Out[56]: Index([   0,    1,    2,    3,    4,    5,    6,    7,    8,    9,
                 ...
                 1549, 1550, 1551, 1552, 1553, 1554, 1555, 1556, 1557, 1558],
                dtype='object', length=1559)
```

```
In [42]: data.duplicated()
```

```
Out[42]: 0        False
         1        False
         2        False
         3        False
         4        False
                  ...
         3279     False
         dtype: bool
```

```
In [43]: data[1558].drop_duplicates()
```

```
Out[43]: 0         ad.
         459     nonad.
         Name: 1558, dtype: object
```

```
In [44]: data[1558].drop_duplicates().tolist()
```

```
Out[44]: ['ad.', 'nonad.']
```

```
In [46]: adindices = data[data .columns[-1]]== 'ad.'
         data.loc[adindices,data .columns[-1]]=1
         nonadindices = data[data .columns[-1]]=='nonad.'
         data.loc[nonadindices,data .columns[-1]]=0
```

```
In [47]: data[1558].dtypes
```

```
Out[47]: dtype('O')
```

In [63]:
```
data[data.columns[-1]]=data[data.columns[-1]].astype(float)
```

In [71]:
```
data=data.replace({'?': np.nan})
data=data.replace({'  ?': np.nan})
data=data.replace({'   ?': np.nan})
data=data.replace({'    ?': np.nan})
data=data.replace({'     ?': np.nan})
```

In [73]:
```
data=data.dropna()
```

In [74]:
```
data=data.fillna(-1)
```

In [82]:
```
data=data.apply(lambda x: pd.to_numeric(x))
```

In [83]:
```
data1 = pd.DataFrame(columns=[i for i in xrange(1559)])
data1.loc[len(data1)] = [random.randint(0,1) for r in xrange(1558)]+[1]
data1.loc[len(data1)] = [random.randint(0,1) for r in xrange(1558)]+[1]
```

In [85]:
```
print len(data)
datatot = pd.concat([data[:],data1[:]])
len(datatot)
```

```
2362
```

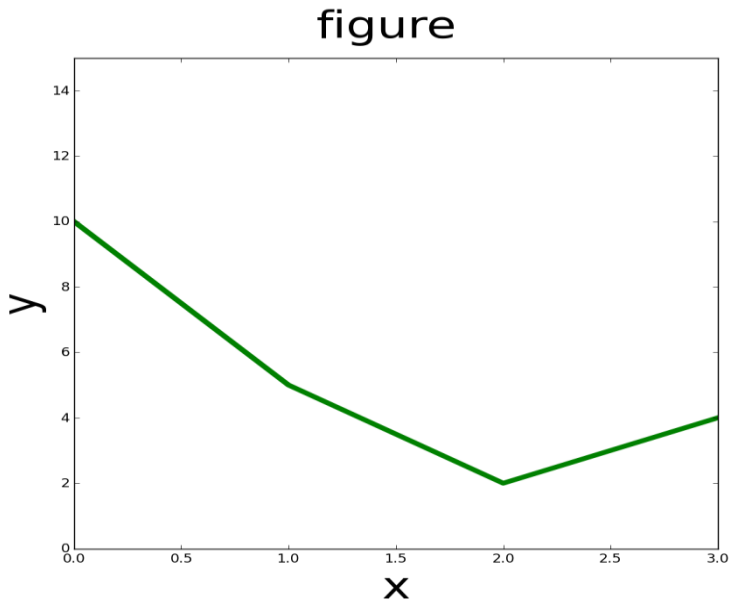
Out[85]:
```
2364
```

# Matplotlib tutorial

In [1]:
```
import matplotlib.pyplot as plt
```

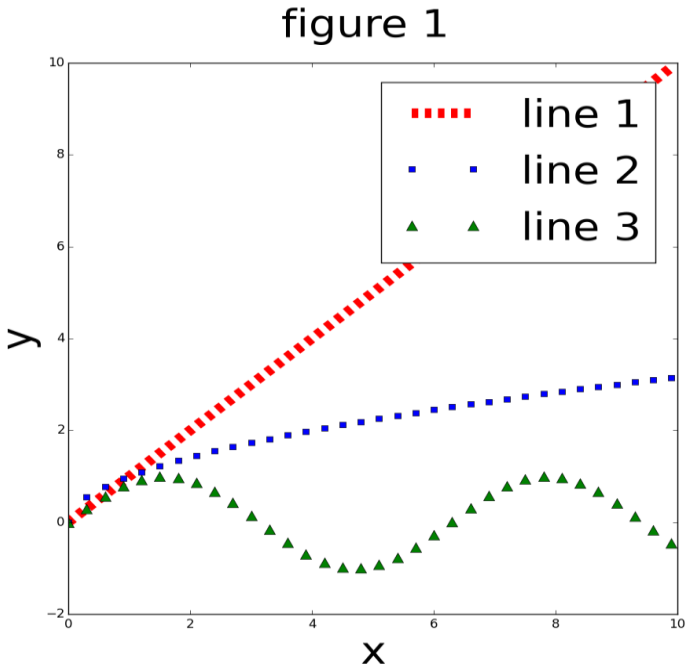
In [2]:
```
plt.plot([10,5,2,4],color='green',label='line 1', linewidth=5)
plt.ylabel('y',fontsize=40)
plt.xlabel('x',fontsize=40)
plt.axis([0,3, 0,15])
plt.show()
```

In [5]:
```
fig = plt.figure(figsize=(10,10))
ax = fig.add_subplot(111)
ax.set_xlabel('x',fontsize=40)
ax.set_ylabel('y',fontsize=40)
fig.suptitle('figure',fontsize=40)
ax.plot([10,5,2,4],color='green',label='line 1', linewidth=5)
fig.savefig('figure.png')
```



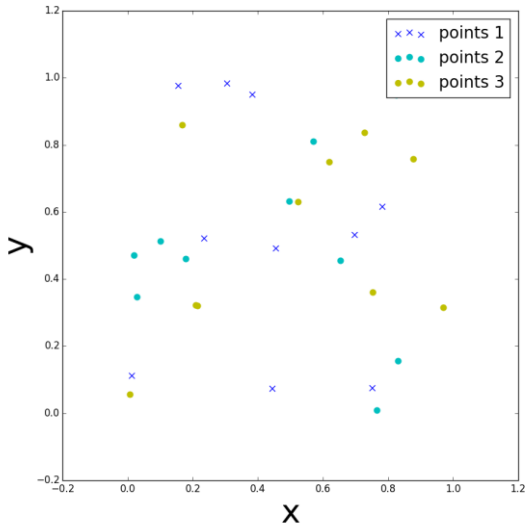

Example of simple plot

In [8]:
```
import numpy as np
fig = plt.figure(figsize=(10,10))
ax = fig.add_subplot(111)
r = np.arange(0., 10., 0.3)
p1, = ax.plot(r, r, 'r--',label='line 1', linewidth=10)
p2, = ax.plot(r, r**0.5, 'bs',label='line 2', linewidth=10)
p3, = ax.plot(r,np.sin(r),'g^', label='line 3', markersize=10)
handles, labels = ax.get_legend_handles_labels()
ax.legend(handles, labels,fontsize=40)
ax.set_xlabel('x',fontsize=40)
ax.set_ylabel('y',fontsize=40)
fig.suptitle('figure 1',fontsize=40)
fig.savefig('figure_multiplelines.png')
```




Example of plot with multiple lines

In [10]:
```
colors = ['b', 'c', 'y', 'm', 'r']
fig = plt.figure(figsize=(10,10))
ax = fig.add_subplot(111)
ax.scatter(np.random.random(10), np.random.random(10), marker='x', color=colors[0])
p1 = ax.scatter(np.random.random(10), np.random.random(10), marker='x', color=colors[0],s=50)
p2 = ax.scatter(np.random.random(10), np.random.random(10), marker='o', color=colors[1],s=50)
p3 = ax.scatter(np.random.random(10), np.random.random(10), marker='o', color=colors[2],s=50)
ax.legend((p1,p2,p3),('points 1','points 2','points 3'),fontsize=20)
ax.set_xlabel('x',fontsize=40)
ax.set_ylabel('y',fontsize=40)
fig.savefig('figure_scatterplot.png')
```




Scatter plot of randomly distributed points

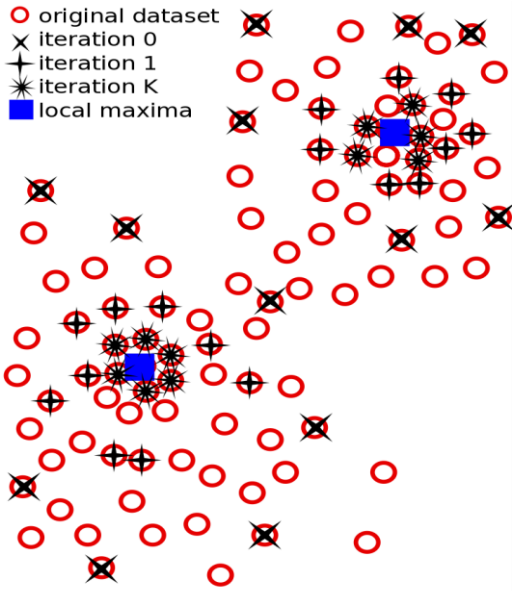
# 2

# Machine Learning Techniques – Unsupervised

## Learning Clustering algorithms

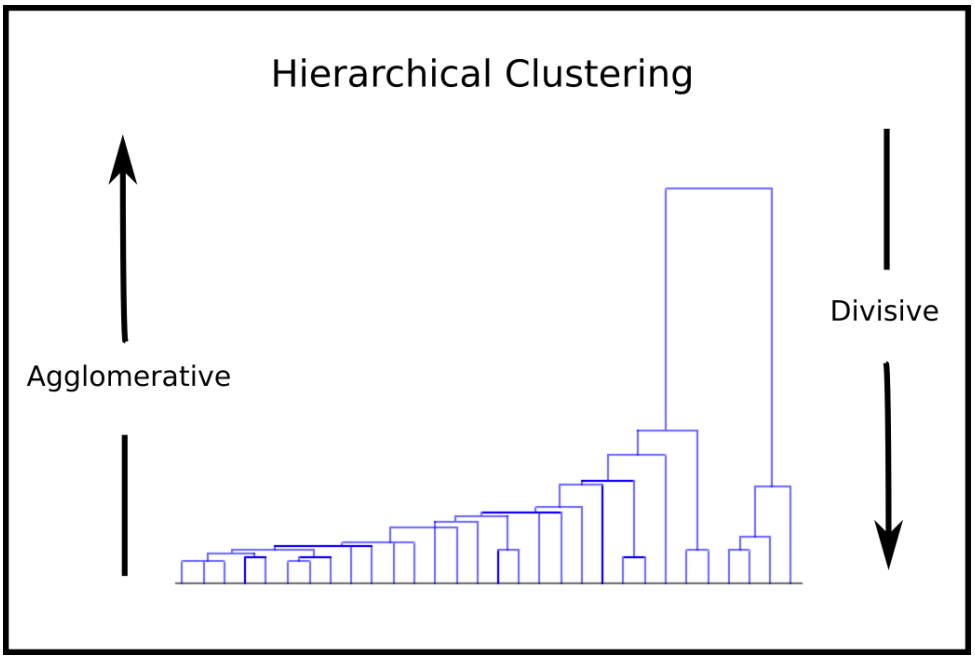
### Density methods

#### Mean – shift

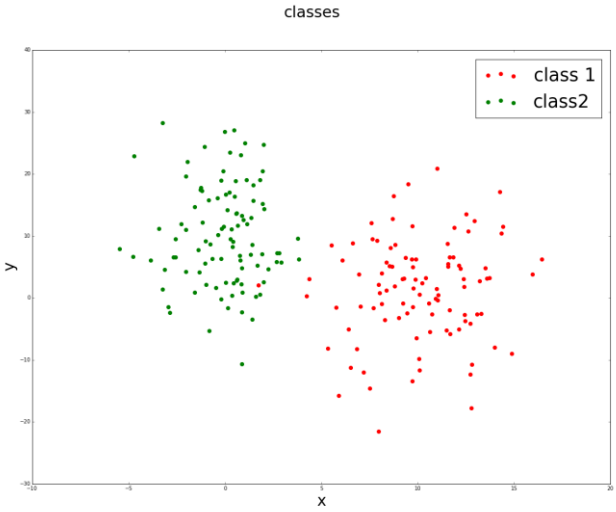



Sketch of the mean-shift evolution through iterations

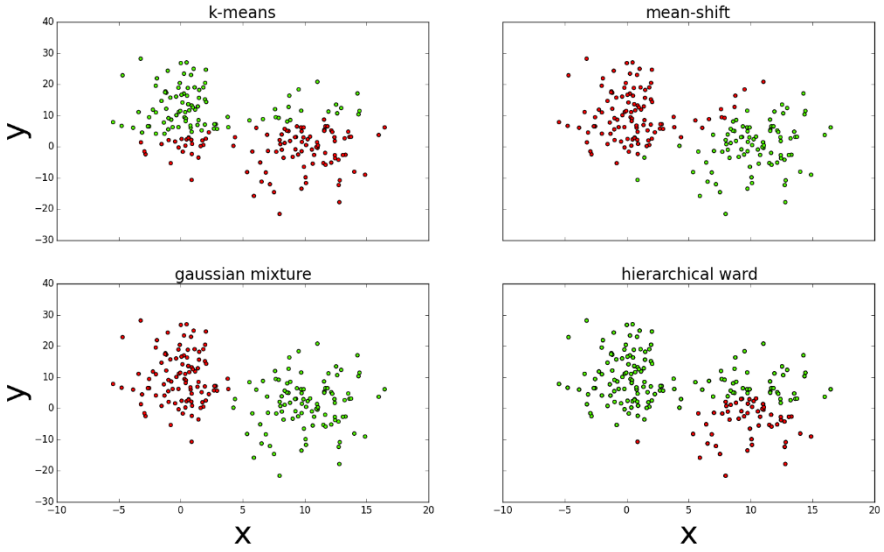
## Hierarchical methods



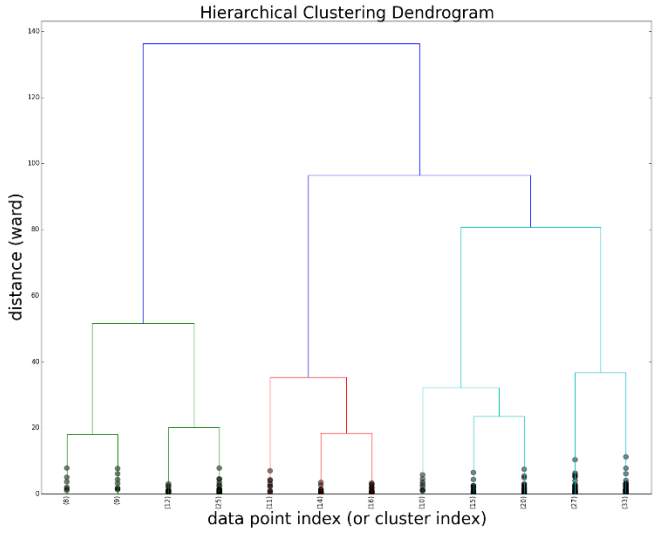
## Training and comparison of the clustering methods




Two multivariate normal classes with noise



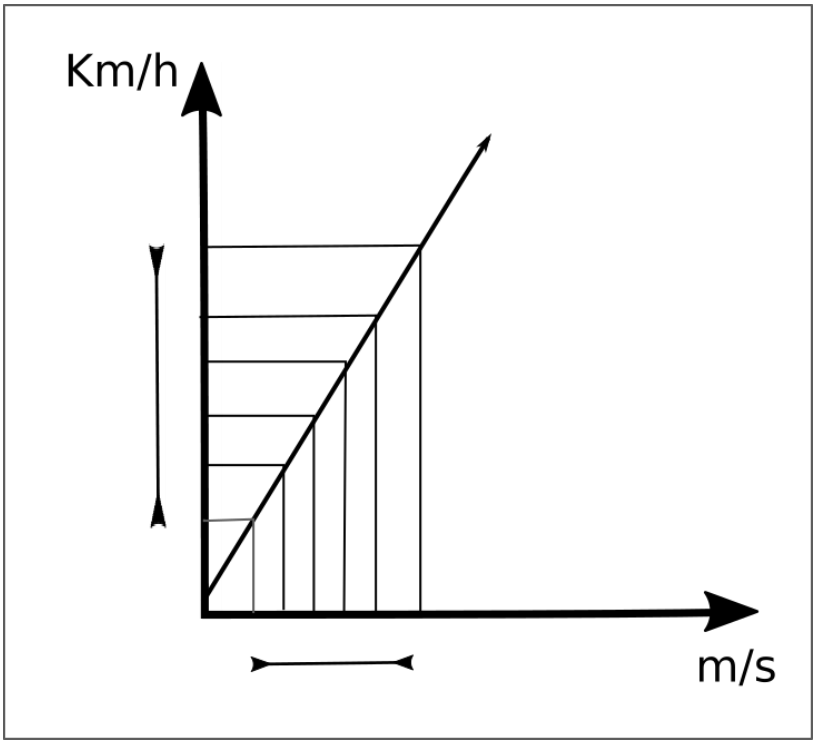
IClustering of the two multivariate classes using k-means, mean-shift, Gaussian mixture model, and hierarchical ward method




IHierarchical clustering dendrogram for the last 12 merges.

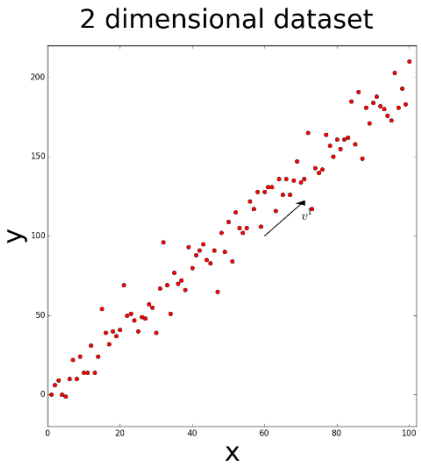
# Dimensionality reduction

## Principal Component Analysis (PCA)




The linear function between the velocity in m/s and Km/h

## PCA example

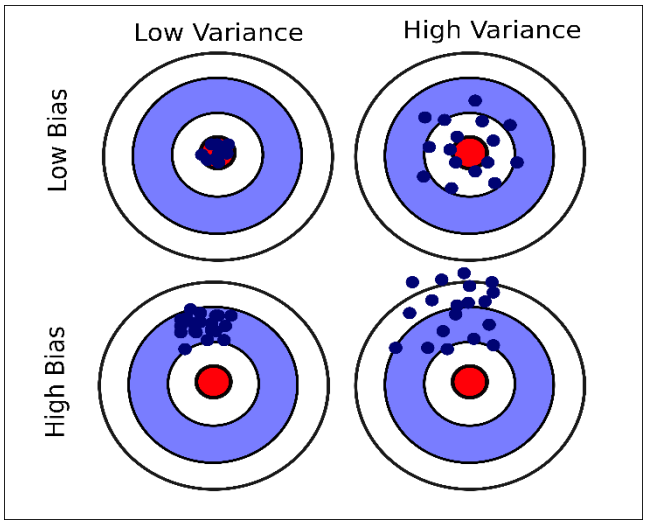



A two-dimensional dataset. The principal component direction v1 is indicated by an arrow.

# 3

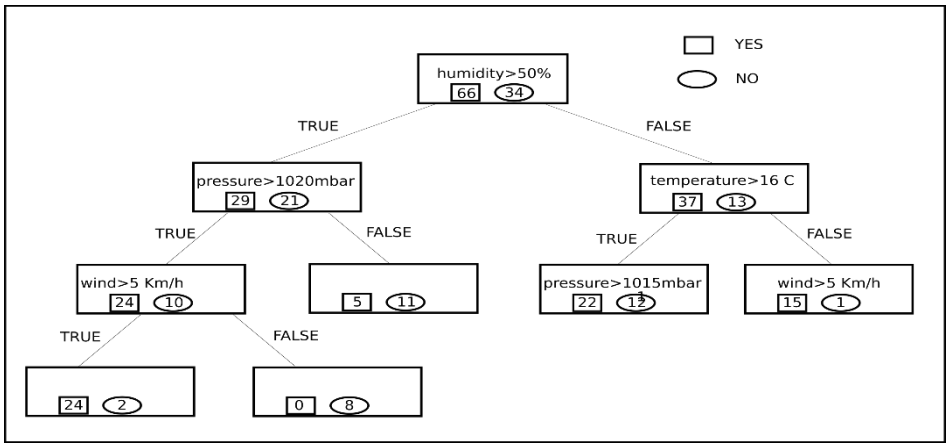
# Supervised Machine Learning

## Model error estimation



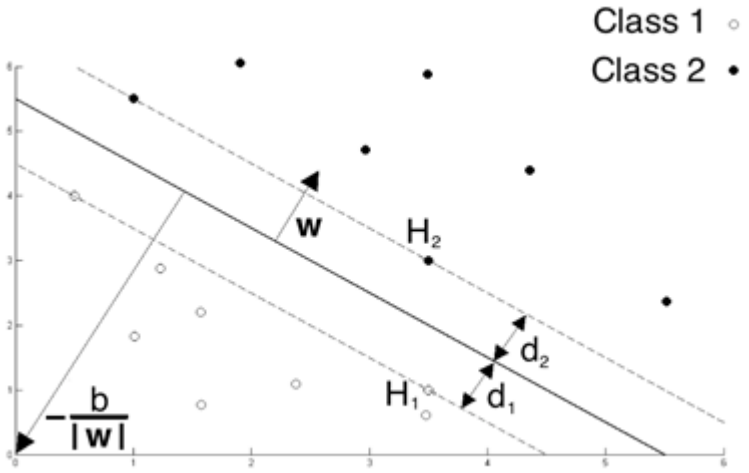

Variance and bias example.

# Decision trees

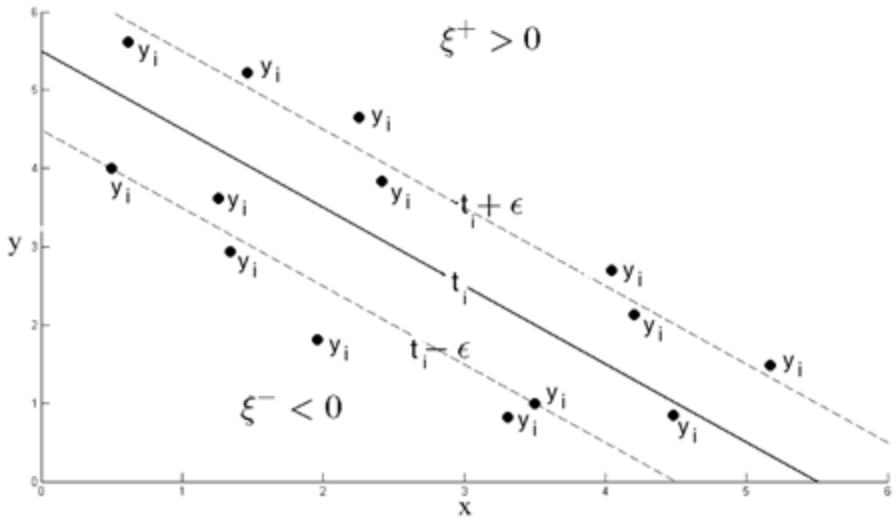



Decision tree for predicting whether to bring an umbrella or not based on a record of 100 days.

# Support vector machine

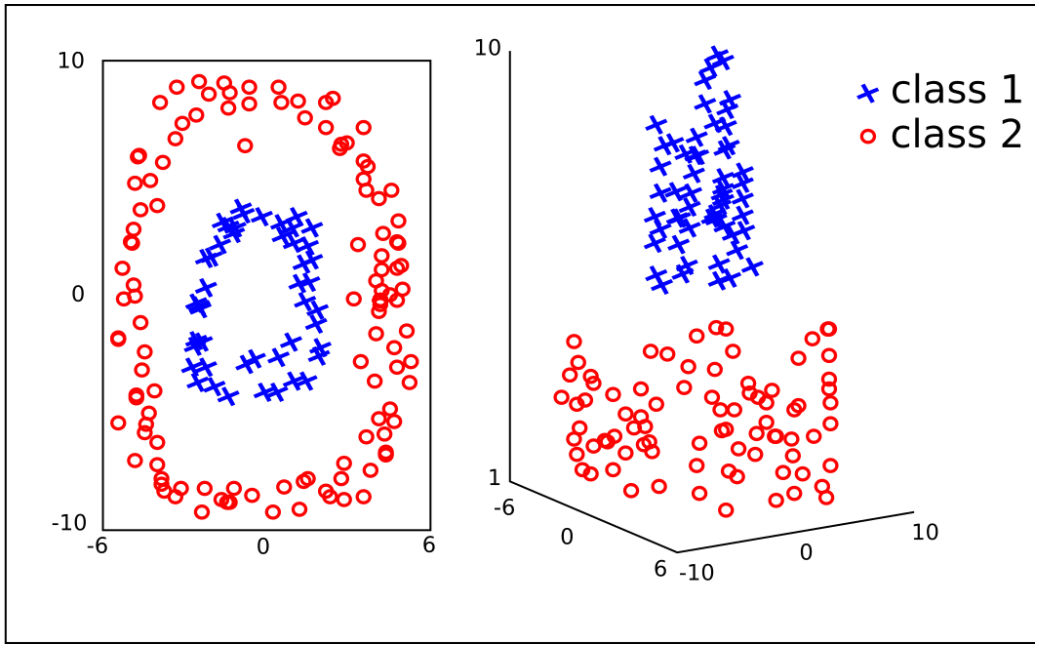



Sketch of the dataset separated in two classes (empty and filled circles) by the black line (decision boundary)




The predictions  lie around the true value

## Kernel trick




In a two-dimensional space, the dataset shown on the left is not separable. Mapping the dataset in a three-dimensional space, the two classes are separable.

# A comparison of methods

## Regression problem

In [4]:
```
import pandas as pd
import numpy as np
from sklearn import cross_validation
from sklearn import svm
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import Ridge
from sklearn.linear_model import Lasso
from sklearn.neighbors import KNeighborsRegressor
from sklearn.metrics import mean_squared_error
```

In [7]:

```
df = pd.read_csv('housing.csv',sep=',',header=None)
#shuffle the data
df = df.iloc[np.random.permutation(len(df))]
X= df[df.columns[:-1]].values
Y = df[df.columns[-1]].values

cv = 10
print 'linear regression'
lin = LinearRegression()
scores = cross_validation.cross_val_score(lin, X, Y, cv=cv)
print("mean R2: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
predicted = cross_validation.cross_val_predict(lin, X,Y, cv=cv)
print 'MSE:',mean_squared_error(Y,predicted)

print 'ridge regression'
ridge = Ridge(alpha=1.0)
scores = cross_validation.cross_val_score(ridge, X, Y, cv=cv)
print("mean R2: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
predicted = cross_validation.cross_val_predict(ridge, X,Y, cv=cv)
print 'MSE:',mean_squared_error(Y,predicted)

print 'lasso regression'
lasso = Lasso(alpha=0.1)
scores = cross_validation.cross_val_score(lasso, X, Y, cv=cv)
print("mean R2: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
predicted = cross_validation.cross_val_predict(lasso, X,Y, cv=cv)
print 'MSE:',mean_squared_error(Y,predicted)

print 'decision tree regression'
tree = DecisionTreeRegressor(random_state=0)
scores = cross_validation.cross_val_score(tree, X, Y, cv=cv)
print("mean R2: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
predicted = cross_validation.cross_val_predict(tree, X,Y, cv=cv)
print 'MSE:',mean_squared_error(Y,predicted)

print 'random forest regression'
forest = RandomForestRegressor(n_estimators=50, max_depth=None,min_samples_split=1,
                               random_state=0)
scores = cross_validation.cross_val_score(forest, X, Y, cv=cv)
print("mean R2: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
predicted = cross_validation.cross_val_predict(forest, X,Y, cv=cv)
print 'MSE:',mean_squared_error(Y,predicted)
#svm
print 'linear support vector machine'
svm_lin = svm.SVR(epsilon=0.2,kernel='linear',C=1)
scores = cross_validation.cross_val_score(svm_lin, X, Y, cv=cv)
print("mean R2: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
predicted = cross_validation.cross_val_predict(svm_lin, X,Y, cv=cv)
print 'MSE:',mean_squared_error(Y,predicted)

print 'support vector machine rbf'
clf = svm.SVR(epsilon=0.2,kernel='rbf',C=1.)
scores = cross_validation.cross_val_score(clf, X, Y, cv=cv)
print("mean R2: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
predicted = cross_validation.cross_val_predict(clf, X,Y, cv=cv)
print 'MSE:',mean_squared_error(Y,predicted)

print 'knn'
knn = KNeighborsRegressor()
scores = cross_validation.cross_val_score(knn, X, Y, cv=cv)
print("mean R2: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
predicted = cross_validation.cross_val_predict(knn, X,Y, cv=cv)
print 'MSE:',mean_squared_error(Y,predicted)
```

```
linear regression
mean R2: 0.72 (+/- 0.15)
MSE: 23.5515499366
ridge regression
mean R2: 0.72 (+/- 0.16)
MSE: 23.7397585761
lasso regression
mean R2: 0.71 (+/- 0.17)
MSE: 24.734860679
decision tree regression
mean R2: 0.75 (+/- 0.24)
MSE: 19.8023913043
random forest regression
mean R2: 0.87 (+/- 0.12)
MSE: 10.9910313913
linear support vector machine
mean R2: 0.70 (+/- 0.25)
MSE: 25.833801836
support vector machine rbf
mean R2: -0.01 (+/- 0.11)
MSE: 83.8283880541
knn
mean R2: 0.54 (+/- 0.23)
MSE: 37.8792632411
```

In [9]:

```
from sklearn.feature_selection import RFE
best_features=4
print 'feature selection on linear regression'
rfe_lin = RFE(lin,best_features).fit(X,Y)
mask = np.array(rfe_lin.support_)
scores = cross_validation.cross_val_score(lin, X[:,mask], Y, cv=cv)
print("R2: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
predicted = cross_validation.cross_val_predict(lin, X[:,mask],Y, cv=cv)
print 'MSE:',mean_squared_error(Y,predicted)

print 'feature selection ridge regression'
rfe_ridge = RFE(ridge,best_features).fit(X,Y)
mask = np.array(rfe_ridge.support_)
scores = cross_validation.cross_val_score(ridge, X[:,mask], Y, cv=cv)
print("R2: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
predicted = cross_validation.cross_val_predict(ridge, X[:,mask],Y, cv=cv)
print 'MSE:',mean_squared_error(Y,predicted)

print 'feature selection on lasso regression'
rfe_lasso = RFE(lasso,best_features).fit(X,Y)
mask = np.array(rfe_lasso.support_)
scores = cross_validation.cross_val_score(lasso, X[:,mask], Y, cv=cv)
print("R2: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
predicted = cross_validation.cross_val_predict(lasso, X[:,mask],Y, cv=cv)
print 'MSE:',mean_squared_error(Y,predicted)

print 'feature selection on decision tree'
rfe_tree = RFE(tree,best_features).fit(X,Y)
mask = np.array(rfe_tree.support_)
scores = cross_validation.cross_val_score(tree, X[:,mask], Y, cv=cv)
print("R2: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
predicted = cross_validation.cross_val_predict(tree, X[:,mask],Y, cv=cv)
print 'MSE:',mean_squared_error(Y,predicted)

print 'feature selection on random forest'
rfe_forest = RFE(forest,best_features).fit(X,Y)
mask = np.array(rfe_forest.support_)
scores = cross_validation.cross_val_score(forest, X[:,mask], Y, cv=cv)
print("R2: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
predicted = cross_validation.cross_val_predict(forest, X[:,mask],Y, cv=cv)
print 'MSE:',mean_squared_error(Y,predicted)

print 'feature selection on linear support vector machine'
rfe_svm = RFE(svm_lin,best_features).fit(X,Y)
mask = np.array(rfe_svm.support_)
scores = cross_validation.cross_val_score(svm_lin, X[:,mask], Y, cv=cv)
print("R2: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
predicted = cross_validation.cross_val_predict(svm_lin, X,Y, cv=cv)
print 'MSE:',mean_squared_error(Y,predicted)
```

```
feature selection on linear regression
R2: 0.61 (+/- 0.31)
MSE: 33.182126206
feature selection ridge regression
R2: 0.61 (+/- 0.32)
MSE: 33.2543979822
feature selection on lasso regression
R2: 0.68 (+/- 0.20)
MSE: 27.4174043724
feature selection on decision tree
R2: 0.70 (+/- 0.35)
MSE: 24.1185968379
feature selection on random forest
R2: 0.84 (+/- 0.14)
MSE: 13.6755712332
feature selection on linear support vector machine
R2: 0.60 (+/- 0.33)
```

# Classification problem

In [1]:
```
import pandas as pd
import numpy as np
from sklearn import cross_validation
from sklearn import svm
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.naive_bayes import MultinomialNB
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import f1_score
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
```

In [10]:
```
#read data in
df = pd.read_csv('data_cars.csv',header=None)
for i in range(len(df.columns)):
    df[i] = df[i].astype('category')
df.head()
```

Out[10]:

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|---|
| **0** | vhigh | vhigh | 2 | 2 | small | low | unacc |
| **1** | vhigh | vhigh | 2 | 2 | small | med | unacc |
| **2** | vhigh | vhigh | 2 | 2 | small | high | unacc |
| **3** | vhigh | vhigh | 2 | 2 | med | low | unacc |
| **4** | vhigh | vhigh | 2 | 2 | med | med | unacc |

In [14]:

```
#map catgories to values
map0 = dict( zip( df[0].cat.categories, range( len(df[0].cat.categories ))))
#print map0
map1 = dict( zip( df[1].cat.categories, range( len(df[1].cat.categories ))))
map2 = dict( zip( df[2].cat.categories, range( len(df[2].cat.categories ))))
map3 = dict( zip( df[3].cat.categories, range( len(df[3].cat.categories ))))
map4 = dict( zip( df[4].cat.categories, range( len(df[4].cat.categories ))))
map5 = dict( zip( df[5].cat.categories, range( len(df[5].cat.categories ))))
map6 = dict( zip( df[6].cat.categories, range( len(df[6].cat.categories ))))

cat_cols = df.select_dtypes(['category']).columns
df[cat_cols] = df[cat_cols].apply(lambda x: x.cat.codes)

df = df.iloc[np.random.permutation(len(df))]
print df.head()
```

```
      0  1  2  3  4  5  6
570   0  0  1  0  1  1  2
951   2  3  3  0  0  1  2
1633  1  1  0  1  1  2  0
412   3  1  3  0  0  2  2
156   3  0  1  2  1  1  2
```

In [40]:

```
df_f1 = pd.DataFrame(columns=['method']+sorted(map6, key=map6.get))
df_precision = pd.DataFrame(columns=['method']+sorted(map6, key=map6.get))
df_recall = pd.DataFrame(columns=['method']+sorted(map6, key=map6.get))
def CalcMeasures(method,y_pred,y_true,df_f1=df_f1
                 ,df_precision=df_precision,df_recall=df_recall):

    df_f1.loc[len(df_f1)] = [method]+list(f1_score(y_pred,y_true,average=None))
    df_precision.loc[len(df_precision)] = [method]+list(precision_score(y_pred,y_true,average=None))
    df_recall.loc[len(df_recall)] = [method]+list(recall_score(y_pred,y_true,average=None))

X= df[df.columns[:-1]].values
Y = df[df.columns[-1]].values
```

In [41]:

```
cv = 10
method = 'linear support vector machine'
clf = svm.SVC(kernel='linear',C=50)
y_pred = cross_validation.cross_val_predict(clf, X,Y, cv=cv)
CalcMeasures(method,y_pred,Y)

method = 'rbf support vector machine'
clf = svm.SVC(kernel='rbf',C=50)
y_pred = cross_validation.cross_val_predict(clf, X,Y, cv=cv)
CalcMeasures(method,y_pred,Y)

method = 'poly support vector machine'
clf = svm.SVC(kernel='poly',C=50)
y_pred = cross_validation.cross_val_predict(clf, X,Y, cv=cv)
CalcMeasures(method,y_pred,Y)

method = 'decision tree'
clf = DecisionTreeClassifier(random_state=0)
y_pred = cross_validation.cross_val_predict(clf, X,Y, cv=cv)
CalcMeasures(method,y_pred,Y)

method = 'random forest'
clf = RandomForestClassifier(n_estimators=50,random_state=0,max_features=None)
y_pred = cross_validation.cross_val_predict(clf, X,Y, cv=cv)
CalcMeasures(method,y_pred,Y)

method = 'naive bayes'
clf = MultinomialNB()
y_pred = cross_validation.cross_val_predict(clf, X,Y, cv=cv)
CalcMeasures(method,y_pred,Y)

method = 'logistic regression'
clf = LogisticRegression()
y_pred = cross_validation.cross_val_predict(clf, X,Y, cv=cv)
CalcMeasures(method,y_pred,Y)

method = 'k nearest neighbours'
clf = KNeighborsClassifier(weights='distance',n_neighbors=5)
y_pred = cross_validation.cross_val_predict(clf, X,Y, cv=cv)
CalcMeasures(method,y_pred,Y)
```

In [45]:
```
df_f1
```

Out[45]:

| | method | acc | good | unacc | vgood |
|---|---|---|---|---|---|
| **0** | linear support vector machine | 0.271318 | 0.000000 | 0.846757 | 0.000000 |
| **1** | rbf support vector machine | 0.990921 | 1.000000 | 0.997933 | 0.984375 |
| **2** | poly support vector machine | 0.788918 | 0.841270 | 0.938010 | 0.800000 |
| **3** | decision tree | 0.957309 | 0.882353 | 0.989238 | 0.946565 |
| **4** | random forest | 0.963918 | 0.915493 | 0.991275 | 0.961832 |
| **5** | naive bayes | 0.040404 | 0.000000 | 0.825701 | 0.000000 |
| **6** | logistic regression | 0.265781 | 0.000000 | 0.820967 | 0.078947 |
| **7** | k nearest neighbours | 0.801609 | 0.534653 | 0.952988 | 0.666667 |

In [46]:
```
df_precision
```

Out[46]:

| | method | acc | good | unacc | vgood |
|---|---|---|---|---|---|
| **0** | linear support vector machine | 0.182292 | 0.000000 | 0.981818 | 0.000000 |
| **1** | rbf support vector machine | 0.994792 | 1.000000 | 0.997521 | 0.969231 |
| **2** | poly support vector machine | 0.778646 | 0.768116 | 0.950413 | 0.738462 |
| **3** | decision tree | 0.963542 | 0.869565 | 0.987603 | 0.953846 |
| **4** | random forest | 0.973958 | 0.942029 | 0.985950 | 0.969231 |
| **5** | naive bayes | 0.020833 | 0.000000 | 0.998347 | 0.000000 |
| **6** | logistic regression | 0.208333 | 0.000000 | 0.919008 | 0.046154 |
| **7** | k nearest neighbours | 0.778646 | 0.391304 | 0.988430 | 0.507692 |

In [47]:
```
df_recall
```

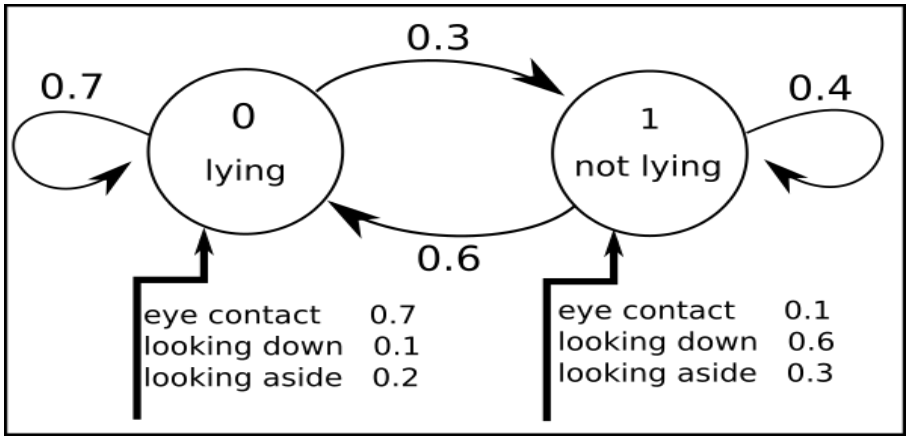
Out[47]:

| | method | acc | good | unacc | vgood |
|---|---|---|---|---|---|
| **0** | linear support vector machine | 0.530303 | 0.000000 | 0.744361 | 0.000000 |
| **1** | rbf support vector machine | 0.987080 | 1.000000 | 0.998346 | 1.000000 |
| **2** | poly support vector machine | 0.799465 | 0.929825 | 0.925926 | 0.872727 |
| **3** | decision tree | 0.951157 | 0.895522 | 0.990879 | 0.939394 |
| **4** | random forest | 0.954082 | 0.890411 | 0.996658 | 0.954545 |
| **5** | naive bayes | 0.666667 | 0.000000 | 0.703963 | 0.000000 |
| **6** | logistic regression | 0.366972 | 0.000000 | 0.741828 | 0.272727 |
| **7** | k nearest neighbours | 0.825967 | 0.843750 | 0.920000 | 0.970588 |

```
In [42]: labels_counts=df[6].value_counts()
         pd.Series(map6).map(labels_counts)
```

```
Out[42]: acc        384
         good        69
         unacc     1210
         vgood       65
         dtype: int64
```

# Hidden Markov model




Salesman behavior – two states hidden Markov model

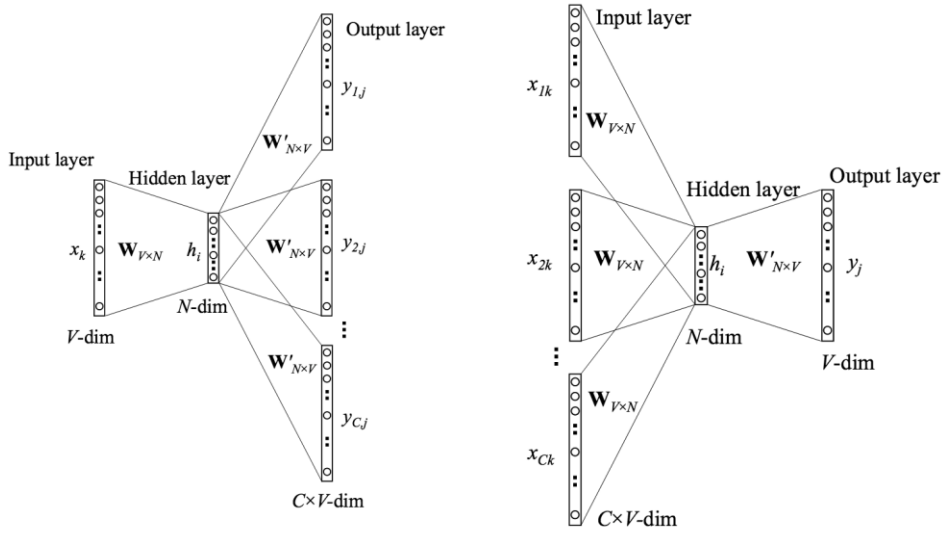
# 4
# Web Mining Techniques

## Natural language processing

```
>>> import nltk
>>> from nltk.tokenize import WordPunctTokenizer
>>> nltk.download('stopwords')
[nltk_data] Downloading package 'stopwords' to
[nltk_data]     /Users/andrea/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
True
>>> from nltk.corpus import stopwords
>>> stopwords = stopwords.words('english')
>>> tknzr = WordPunctTokenizer()
>>> from nltk.stem.porter import PorterStemmer
>>> stemmer = PorterStemmer()
>>> text = 'The European languages are members of the same family. Many words in a language trans
late into familiar words in another. For science, music, sport, etc, Europe uses the same vocabul
ary.  Everyone realizes why a new common language would be desirable: one could refuse to pay tra
nslators.'
>>> words = tknzr.tokenize(text)
>>> words
['The', 'European', 'languages', 'are', 'members', 'of', 'the', 'same', 'family', '.', 'Many', 'w
ords', 'in', 'a', 'language', 'translate', 'into', 'familiar', 'words', 'in', 'another', '.', 'Fo
r', 'science', ',', 'music', ',', 'sport', ',', 'etc', ',', 'Europe', 'uses', 'the', 'same', 'voc
abulary', '.', 'Everyone', 'realizes', 'why', 'a', 'new', 'common', 'language', 'would', 'be', 'd
esirable', ':', 'one', 'could', 'refuse', 'to', 'pay', 'translators', '.']
>>> words_clean = [w.lower() for w in words if w not in stopwords]
>>> words_clean
['the', 'european', 'languages', 'members', 'family', '.', 'many', 'words', 'language', 'translat
e', 'familiar', 'words', 'another', '.', 'for', 'science', ',', 'music', ',', 'sport', ',', 'etc'
, ',', 'europe', 'uses', 'vocabulary', '.', 'everyone', 'realizes', 'new', 'common', 'language',
'would', 'desirable', ':', 'one', 'could', 'refuse', 'pay', 'translators', '.']
>>> words_clean_stem = [stemmer.stem(w) for w in words_clean]
>>> words_clean_stem
['the', 'european', 'languag', 'member', 'famili', '.', 'mani', 'word', 'languag', 'translat', 'f
amiliar', 'word', 'anoth', '.', 'for', 'scienc', ',', 'music', ',', 'sport', ',', 'etc', ',', 'eu
rop', 'use', 'vocabulari', '.', 'everyon', 'realiz', 'new', 'common', 'languag', 'would', 'desir'
, ':', 'one', 'could', 'refus', 'pay', 'translat', '.']
```

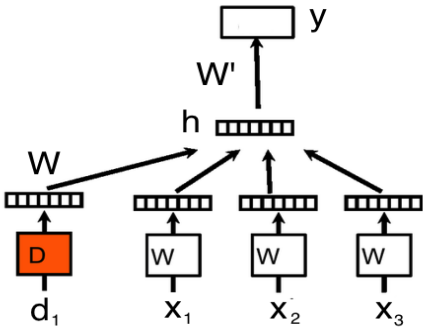
# Information retrieval models

## Word2vec – continuous bag of words and skip-gram architectures




skip-gram (left) and CBOW (right) architectures of the word2vec algorithm; figures taken from word2vec Parameter Learning Explained by X Rong (2015)

## Doc2Vec extension



A distributed memory model example with a context of three words (window=3); figure taken from Distributed Representations of Sentences and Documents by Le and Mikolov (2014)

# Movie review query example

In [1]:
```
#import files
import os
import numpy as np
#get titles
from BeautifulSoup import BeautifulSoup
moviehtmldir = './movie/'
moviedict = {}
for filename in [f for f in os.listdir(moviehtmldir) if f[0]!='.']:
    id = filename.split('.')[0]
    f = open(moviehtmldir+'/'+filename)
    parsed_html = BeautifulSoup(f.read())
    try:
        title = parsed_html.body.h1.text

    except:
        title = 'none'
    moviedict[id] = title
```

In [99]:
```
import nltk
from nltk.corpus import stopwords
from nltk.tokenize import WordPunctTokenizer
tknzr = WordPunctTokenizer()
nltk.download('stopwords')
stoplist = stopwords.words('english')
from nltk.stem.porter import PorterStemmer
stemmer = PorterStemmer()
def ListDocs(dirname):
    docs = []
    titles = []
    for filename in [f for f in os.listdir(dirname) if str(f)[0]!='.']:
        f = open(dirname+'/'+filename,'r')
        id = filename.split('.')[0].split('_')[1]
        titles.append(moviedict[id])
        docs.append(f.read())
    return docs,titles

dir = './review_polarity/txt_sentoken/'
pos_textreviews,pos_titles = ListDocs(dir+'pos/')
neg_textreviews,neg_titles = ListDocs(dir+'neg/')
tot_textreviews = pos_textreviews+neg_textreviews
tot_titles = pos_titles+neg_titles
```

In [4]:
```
#test tf-idf
from sklearn.feature_extraction.text import TfidfVectorizer

def PreprocessTfidf(texts,stoplist=[],stem=False):
    newtexts = []
    for text in texts:
        if stem:
            tmp = [w for w in tknzr.tokenize(text) if w not in stoplist]
        else:
            tmp = [stemmer.stem(w) for w in [w for w in tknzr.tokenize(text) if w not in stoplist]]
        newtexts.append(' '.join(tmp))
    return newtexts
vectorizer = TfidfVectorizer(min_df=1)
processed_reviews = PreprocessTfidf(tot_textreviews,stoplist,True)
mod_tfidf = vectorizer.fit(processed_reviews)
vec_tfidf = mod_tfidf.transform(processed_reviews)
tfidf = dict(zip(vectorizer.get_feature_names(),vectorizer.idf_))
```

In [5]:
```
#test LSA
import gensim
from gensim import models
class GenSimCorpus(object):
           def __init__(self, texts, stoplist=[],stem=False):
               self.texts = texts
               self.stoplist = stoplist
               self.stem = stem
               self.dictionary = gensim.corpora.Dictionary(self.iter_docs(texts, stoplist))


           def __len__(self):
               return len(self.texts)
           def __iter__(self):
               for tokens in self.iter_docs(self.texts, self.stoplist):
                   yield self.dictionary.doc2bow(tokens)
           def iter_docs(self,texts, stoplist):
               for text in texts:
                   if self.stem:
                      yield (stemmer.stem(w) for w in [x for x in tknzr.tokenize(text) if x not in stoplist])
                   else:
                      yield (x for x in tknzr.tokenize(text) if x not in stoplist)

corpus = GenSimCorpus(tot_textreviews,stoplist,True)
dict_corpus = corpus.dictionary
ntopics = 10
lsi =  models.LsiModel(corpus, num_topics=ntopics, id2word=dict_corpus)
```

In [6]:
```
U = lsi.projection.u
Sigma = np.eye(ntopics)*lsi.projection.s
#calculate V
V = gensim.matutils.corpus2dense(lsi[corpus], len(lsi.projection.s)).T / lsi.projection.s
dict_words = {}
for i in range(len(dict_corpus)):
    dict_words[dict_corpus[i]] = i
```

In [7]:
```
from collections import namedtuple

def PreprocessDoc2Vec(text,stop=[],stem=False):
    words = tknzr.tokenize(text)
    if stem:
       words_clean = [stemmer.stem(w) for w in [i.lower() for i in words if i not in stop]]
    else:
       words_clean = [i.lower() for i in words if i not in stop]
    return words_clean

Review = namedtuple('Review','words tags')
dir = './review_polarity/txt_sentoken/'
do2vecstem = False
reviews_pos = []
cnt = 0
for filename in [f for f in os.listdir(dir+'pos/') if str(f)[0]!='.']:
    f = open(dir+'pos/'+filename,'r')
    reviews_pos.append(Review(PreprocessDoc2Vec(f.read(),stoplist,do2vecstem),['pos_'+str(cnt)]))
    cnt+=1

reviews_neg = []
cnt= 0
for filename in [f for f in os.listdir(dir+'neg/') if str(f)[0]!='.']:
    f = open(dir+'neg/'+filename,'r')
    reviews_neg.append(Review(PreprocessDoc2Vec(f.read(),stoplist,do2vecstem),['neg_'+str(cnt)]))
    cnt+=1

tot_reviews = reviews_pos + reviews_neg
```

In [8]:
```
#define doc2vec
from gensim.models import Doc2Vec
import multiprocessing

cores = multiprocessing.cpu_count()
vec_size = 500
model_d2v = Doc2Vec(dm=1, dm_concat=0, size=vec_size, window=10, negative=0, hs=0, min_count=1, workers=cores)

#build vocab
model_d2v.build_vocab(tot_reviews)
#train
numepochs= 20
for epoch in range(numepochs):
    try:
        print 'epoch %d' % (epoch)
        model_d2v.train(tot_reviews)
        model_d2v.alpha *= 0.99
        model_d2v.min_alpha = model_d2v.alpha
    except (KeyboardInterrupt, SystemExit):
        break
```

In [9]:
```
#query
query = ['science','future','action']
```

In [10]:
```
#similar tfidf
#sparse matrix so the metrics transform into regular vectors before computing cosine
from sklearn.metrics.pairwise import cosine_similarity
query_vec = mod_tfidf.transform(PreprocessTfidf([' '.join(query)],stoplist,True))
sims= cosine_similarity(query_vec,vec_tfidf)[0]
indxs_sims = sims.argsort()[::-1]
for d in list(indxs_sims)[:5]:
    print 'sim:',sims[d],' title:',tot_titles[d]
```

In [11]:
```
#LSA query
def TransformWordsListtoQueryVec(wordslist,dict_words,stem=False):
    q = np.zeros(len(dict_words.keys()))
    for w in wordslist:
        if stem:
            q[dict_words[stemmer.stem(w)]]=1.
        else:
            q[dict_words[w]] = 1.
    return q

q = TransformWordsListtoQueryVec(query,dict_words,True)

qk =   np.dot(np.dot(q,U),Sigma)

sims = np.zeros(len(tot_textreviews))
for d in range(len(V)):
    sims[d]=np.dot(qk,V[d])
indxs_sims = np.argsort(sims)[::-1]
for d in list(indxs_sims)[:5]:
    print 'sim:',sims[d],' doc:',tot_titles[d]
```

In [12]:
```
#doc2vec query
#force inference to get the same result
model_d2v.random = np.random.RandomState(1)
query_docvec = model_d2v.infer_vector(PreprocessDoc2Vec(' '.join(query),stoplist,do2vecstem))

reviews_related = model_d2v.docvecs.most_similar([query_docvec], topn=5)
for review in reviews_related:
    print 'relevance:',review[1],'  title:',tot_titles[review[0]]
```

```
sim: 0.177948650457  title: No Telling (1991)
sim: 0.177821146567  title: Total Recall (1990)
sim: 0.173783798661  title: Time Machine, The (1960)
sim: 0.163031796224  title: Bicentennial Man (1999)
sim: 0.160582512878  title: Andromeda Strain, The (1971)
```

```
sim: 4.0370254245  doc: Star Wars: Episode I - The Phantom Menace (1999)
sim: 3.41798397445  doc: Alien&#179; (1992)
sim: 3.41131742531  doc: Rocky Horror Picture Show, The (1975)
sim: 2.99980957062  doc: Starship Troopers (1997)
sim: 2.86164366049  doc: Wild Things (1998)
```

```
relevance: 0.129549503326   title: Lost World: Jurassic Park, The (1997)
relevance: 0.124721623957   title: In the Heat of the Night (1967)
relevance: 0.122562259436   title: Charlie's Angels (2000)
relevance: 0.119273915887   title: Batman & Robin (1997)
relevance: 0.118506141007   title: Pok&#233;mon: The Movie 2000 (2000)
```

# Postprocessing information

## Latent Dirichlet allocation

### Example

In [6]:
```
#LDA
import gensim.models
from gensim import models

from nltk.tokenize import RegexpTokenizer
tknzr = RegexpTokenizer(r'((?<=[^\w\s])\w(?=[^\w\s])|(\W))+', gaps=True)

from nltk.stem.porter import PorterStemmer
stemmer = PorterStemmer()
class GenSimCorpus(object):
           def __init__(self, texts, stoplist=[],bestwords=[],stem=False):
               self.texts = texts
               self.stoplist = stoplist
               self.stem = stem
               self.bestwords = bestwords
               self.dictionary = gensim.corpora.Dictionary(self.iter_docs(texts, stoplist))

           def __len__(self):
               return len(self.texts)
           def __iter__(self):
               for tokens in self.iter_docs(self.texts, self.stoplist):
                   yield self.dictionary.doc2bow(tokens)
           def iter_docs(self,texts, stoplist):
               for text in texts:
                   if self.stem:
                      yield (stemmer.stem(w) for w in [x for x in tknzr.tokenize(text) if x not in stoplist])
                   else:
                      if len(self.bestwords)>0:
                         yield (x for x in tknzr.tokenize(text) if x in self.bestwords)
                      else:
                         yield (x for x in tknzr.tokenize(text) if x not in stoplist)

num_topics = 10
corpus = GenSimCorpus(tot_textreviews, stoplist,[],False)
dict_lda = corpus.dictionary
```

In [7]:
```
import copy
#filter out very common words like mobie and film or very unfrequent terms
out_ids = [tokenid for tokenid, docfreq in dict_lda.dfs.iteritems() if docfreq > 1000 or docfreq < 3 ]
dict_lfq = copy.deepcopy(dict_lda)
dict_lfq.filter_tokens(out_ids)
dict_lfq.compactify()
corpus = [dict_lfq.doc2bow(tknzr.tokenize(text)) for text in tot_textreviews]
```

In [8]:
```
lda_lfq = models.LdaModel(corpus, num_topics=num_topics, id2word=dict_lfq,passes=10, iterations=50,alpha=0.01,eta=0.01)
for t in range(num_topics):
    print 'topic ',t,'  words: ',lda_lfq.print_topic(t,topn=10)
    print
```

```
topic  0   words:  0.009*best + 0.008*life + 0.008*although + 0.008*great + 0.007*director + 0.006*own + 0.006*see +
0.006*town + 0.006*doesn + 0.005*still

topic  1   words:  0.014*see + 0.010*know + 0.008*bad + 0.008*off + 0.008*think + 0.007*plot + 0.007*could +
0.007*re + 0.007*life + 0.007*m

topic  2   words:  0.011*disney + 0.009*off + 0.009*action + 0.009*plot + 0.008*love + 0.008*life + 0.007*wild +
0.007*could + 0.006*mulan + 0.006*new

topic  3   words:  0.009*scene + 0.008*life + 0.007*new + 0.007*know + 0.007*doesn + 0.007*off + 0.007*could +
0.006*bad + 0.006*director + 0.006*see

topic  4   words:  0.014*truman + 0.009*life + 0.009*best + 0.008*doesn + 0.007*scene + 0.007*own + 0.007*world +
0.007*sandler + 0.007*see + 0.006*new

topic  5   words:  0.009*bad + 0.008*big + 0.008*off + 0.007*plot + 0.007*doesn + 0.007*director + 0.007*scene +
0.007*go + 0.006*see + 0.006*better

topic  6   words:  0.013*plot + 0.012*action + 0.012*alien + 0.011*bad + 0.009*new + 0.008*off + 0.008*planet +
0.008*see + 0.007*could + 0.006*scene

topic  7   words:  0.013*action + 0.009*plot + 0.007*war + 0.007*off + 0.007*see + 0.007*re + 0.007*van +
0.006*director + 0.006*great + 0.006*made

topic  8   words:  0.012*love + 0.009*best + 0.008*see + 0.007*could + 0.007*life + 0.006*new + 0.006*scene +
0.006*off + 0.006*go + 0.006*re

topic  9   words:  0.016*life + 0.010*world + 0.007*scene + 0.007*could + 0.006*mother + 0.006*own + 0.006*love +
0.006*role + 0.006*off + 0.006*father
```

In [9]:

```
#topics for each doc
def GenerateDistrArrays(corpus):
         for i,dist in enumerate(corpus[:10]):
             dist_array = np.zeros(num_topics)
             for d in dist:
                 dist_array[d[0]] =d[1]
             if dist_array.argmax() == 6 :
                print tot_titles[i]
corpus_lda = lda_lfq[corpus]
GenerateDistrArrays(corpus_lda)
```

# Opinion mining (sentiment analysis)

In [10]:
```
import nltk
from nltk.corpus import stopwords
from nltk.tokenize import WordPunctTokenizer
tknzr = WordPunctTokenizer()

from nltk.tokenize import RegexpTokenizer
tknzr = RegexpTokenizer(r'((?<=[^\w\s])\w(?=[^\w\s])|(\W))+', gaps=True)

nltk.download('stopwords')
stoplist = stopwords.words('english')
from nltk.stem.porter import PorterStemmer
stemmer = PorterStemmer()


from collections import namedtuple

def PreprocessReviews(text,stop=[],stem=False):
    #print profile
    words = tknzr.tokenize(text)
    if stem:
       words_clean = [stemmer.stem(w) for w in [i.lower() for i in words if i not in stop]]
    else:
       words_clean = [i.lower() for i in words if i not in stop]
    return words_clean

Review = namedtuple('Review','words title tags')
dir = './review_polarity/txt_sentoken/'
do2vecstem = True
reviews_pos = []
cnt = 0
for filename in [f for f in os.listdir(dir+'pos/') if str(f)[0]!='.']:
    f = open(dir+'pos/'+filename,'r')
    id = filename.split('.')[0].split('_')[1]
    reviews_pos.append(Review(PreprocessReviews(f.read(),stoplist,do2vecstem),moviedict[id],['pos_'+str(cnt)]))
    cnt+=1

reviews_neg = []
cnt= 0
for filename in [f for f in os.listdir(dir+'neg/') if str(f)[0]!='.']:
    f = open(dir+'neg/'+filename,'r')
    id = filename.split('.')[0].split('_')[1]
    reviews_neg.append(Review(PreprocessReviews(f.read(),stoplist,do2vecstem),moviedict[id],['neg_'+str(cnt)]))
    cnt+=1

tot_reviews = reviews_pos + reviews_neg
```

In [11]:
```
#split in test training sets
def word_features(words):
    return dict([(word, True) for word in words])
negfeatures = [(word_features(r.words), 'neg') for r in reviews_neg]
posfeatures = [(word_features(r.words), 'pos') for r in reviews_pos]
portionpos = int(len(posfeatures)*0.8)
portionneg = int(len(negfeatures)*0.8)
print portionpos,'-',portionneg
trainfeatures = negfeatures[:portionneg] + posfeatures[:portionpos]
print len(trainfeatures)
testfeatures = negfeatures[portionneg:] + posfeatures[portionpos:]
#shuffle(testfeatures)
```

In [12]:
```
from nltk.classify import NaiveBayesClassifier
#training naive bayes
classifier = NaiveBayesClassifier.train(trainfeatures)
##testing
err = 0
print 'test on: ',len(testfeatures)
for r in testfeatures:
    sent = classifier.classify(r[0])
    if sent != r[1]:
       err +=1.
print 'error rate: ',err/float(len(testfeatures))
```

In [16]:
```
import itertools
from nltk.collocations import BigramCollocationFinder
from nltk.metrics import BigramAssocMeasures
from random import shuffle

#train bigram:
def bigrams_words_features(words, nbigrams=200,measure=BigramAssocMeasures.chi_sq):
    bigram_finder = BigramCollocationFinder.from_words(words)
    bigrams = bigram_finder.nbest(measure, nbigrams)
    return dict([(ngram, True) for ngram in itertools.chain(words, bigrams)])

negfeatures = [(bigrams_words_features(r.words,500), 'neg') for r in reviews_neg]
posfeatures = [(bigrams_words_features(r.words,500), 'pos') for r in reviews_pos]
portionpos = int(len(posfeatures)*0.8)
portionneg = int(len(negfeatures)*0.8)
print portionpos,'-',portionneg
trainfeatures = negfeatures[:portionpos] + posfeatures[:portionneg]
print len(trainfeatures)
classifier = NaiveBayesClassifier.train(trainfeatures)
##test bigram
testfeatures = negfeatures[portionneg:] + posfeatures[portionpos:]
shuffle(testfeatures)
err = 0
print 'test on: ',len(testfeatures)
for r in testfeatures:
    sent = classifier.classify(r[0])
    #print r[1],'-pred: ',sent
    if sent != r[1]:
       err +=1.
print 'error rate: ',err/float(len(testfeatures))
```

In [21]:
```
import nltk.classify.util, nltk.metrics
tot_poswords = [val for l in [r.words for r in reviews_pos] for val in l]
tot_negwords = [val for l in [r.words for r in reviews_neg] for val in l]
from nltk.probability import FreqDist, ConditionalFreqDist
word_fd = FreqDist()
label_word_fd = ConditionalFreqDist()

for word in tot_poswords:
    word_fd[word.lower()] +=1
    label_word_fd['pos'][word.lower()] +=1

for word in tot_negwords:
    word_fd[word.lower()] +=1
    label_word_fd['neg'][word.lower()] +=1
pos_words = len(tot_poswords)
neg_words = len(tot_negwords)

tot_words = pos_words + neg_words
#select the best words in terms of information contained in the two classes pos and neg
word_scores = {}

for word, freq in word_fd.iteritems():
    pos_score = BigramAssocMeasures.chi_sq(label_word_fd['pos'][word],
                (freq, pos_words), tot_words)
    neg_score = BigramAssocMeasures.chi_sq(label_word_fd['neg'][word],
                (freq, neg_words), tot_words)
    word_scores[word] = pos_score + neg_score
print 'total: ',len(word_scores)
best = sorted(word_scores.iteritems(), key=lambda (w,s): s, reverse=True)[:10000]
bestwords = set([w for w, s in best])
```

In [22]:
```
#training naive bayes with chi square feature selection of best words
def best_words_features(words):
    return dict([(word, True) for word in words if word in bestwords])

negfeatures = [(best_words_features(r.words), 'neg') for r in reviews_neg]
posfeatures = [(best_words_features(r.words), 'pos') for r in reviews_pos]
portionpos = int(len(posfeatures)*0.8)
portionneg = int(len(negfeatures)*0.8)
print portionpos,'-',portionneg
trainfeatures = negfeatures[:portionpos] + posfeatures[:portionneg]
print len(trainfeatures)
classifier = NaiveBayesClassifier.train(trainfeatures)
##test with feature chi square selection
testfeatures = negfeatures[portionneg:] + posfeatures[portionpos:]
shuffle(testfeatures)
err = 0
print 'test on: ',len(testfeatures)
for r in testfeatures:
    sent = classifier.classify(r[0])
    #print r[1],'-pred: ',sent
    if sent != r[1]:
       err +=1.
print 'error rate: ',err/float(len(testfeatures))
```

In [23]:
```
#split train,test sets
trainingsize = 2*int(len(reviews_pos)*0.8)

train_d2v = np.zeros((trainingsize, vec_size))
train_labels = np.zeros(trainingsize)
test_size = len(tot_reviews)-trainingsize
test_d2v = np.zeros((test_size, vec_size))
test_labels = np.zeros(test_size)

cnt_train = 0
cnt_test = 0
for r in reviews_pos:
    name_pos = r.tags[0]
    if int(name_pos.split('_')[1])>= int(trainingsize/2.):
        test_d2v[cnt_test] = model_d2v.docvecs[name_pos]
        test_labels[cnt_test] = 1
        cnt_test +=1
    else:
        train_d2v[cnt_train] = model_d2v.docvecs[name_pos]
        train_labels[cnt_train] = 1
        cnt_train +=1

for r in reviews_neg:
    name_neg = r.tags[0]
    if int(name_neg.split('_')[1])>= int(trainingsize/2.):
        test_d2v[cnt_test] = model_d2v.docvecs[name_neg]
        test_labels[cnt_test] = 0
        cnt_test +=1
    else:
        train_d2v[cnt_train] = model_d2v.docvecs[name_neg]
        train_labels[cnt_train] = 0
        cnt_train +=1
```

In [27]:
```
#train log regre
from sklearn.linear_model import LogisticRegression
classifier = LogisticRegression()
classifier.fit(train_d2v, train_labels)
print 'accuracy:',classifier.score(test_d2v,test_labels)

from sklearn.svm import SVC
clf = SVC()
clf.fit(train_d2v, train_labels)
print 'accuracy:',clf.score(test_d2v,test_labels)
```

# 5
# Recommendation Systems

## Utility matrix

In [34]:
```
import numpy as np
import pandas as pd
import copy
import collections
from scipy import linalg
import math
from collections import defaultdict
```

In [35]:
```
#data
df = pd.read_csv('./data/ml-100k/u.data',sep='\t',header=None)
#movie list
df_info = pd.read_csv('./data/ml-100k/u.item',sep='|',header=None)
movielist = [df_info[1].tolist()[indx]+';'+str(indx+1) for indx in xrange(len(df_info[1].tolist()))]
nmovies = len(movielist)
nusers = len(df[0].drop_duplicates().tolist())

min_ratings = 50
movies_rated  = list(df[1])
counts = collections.Counter(movies_rated)
dfout = pd.DataFrame(columns=['user']+movielist)

toremovelist = []
for i in range(1,nusers):
    tmpmovielist = [0 for j in range(nmovies)]
    dftmp =df[df[0]==i]
    for k in dftmp.index:
        if counts[dftmp.ix[k][1]]>= min_ratings:
            tmpmovielist[dftmp.ix[k][1]-1] = dftmp.ix[k][2]

        else:
            toremovelist.append(dftmp.ix[k][1])

    dfout.loc[i] = [i]+tmpmovielist

toremovelist = list(set(toremovelist))
dfout.drop(dfout.columns[toremovelist], axis=1, inplace=True)
dfout.to_csv('data/utilitymatrix.csv',index=None)
```

In [38]:
```
df = pd.read_csv('data/utilitymatrix.csv')
df.head(2)
```

Out[38]:

| | user | Toy Story (1995);1 | GoldenEye (1995);2 | Four Rooms (1995);3 | Get Shorty (1995);4 | Copycat (1995);5 | Twelve Monkeys (1995);7 | Babe (1995);8 | Dead Man Walking (1995);9 | Richard III (1995);10 | ... | Cool Runnings (1993);1035 | Hamlet (1996);1039 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 1 | 5 | 3 | 4 | 3 | 3 | 4 | 1 | 5 | 3 | ... | 0 | 0 |
| **1** | 2 | 4 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 2 | ... | 0 | 0 |

2 rows × 604 columns

In [4]:
```
def imputation(inp,Ri):
    Ri = Ri.astype(float)
    def userav():
        for i in xrange(len(Ri)):
            Ri[i][Ri[i]==0] = sum(Ri[i])/float(len(Ri[i][Ri[i]>0]))
        return Ri
    def itemav():
        for i in xrange(len(Ri[0])):
            Ri[:,i][Ri[:,i]==0] = sum(Ri[:,i])/float(len(Ri[:,i][Ri[:,i]>0]))
        return Ri
    switch = {'useraverage':userav(),'itemaverage':itemav()}
    return switch[inp]
```

# Similarities measures

In [8]:
```
from scipy.stats import pearsonr
from scipy.spatial.distance import cosine
def sim(x,y,metric='cos'):
    if metric == 'cos':
        return 1.-cosine(x,y)
    else:#correlation
        return pearsonr(x,y)[0]
```

# Collaborative Filtering methods

## Memory-based collaborative filtering

### User based collaborative filtering

In [51]:
```
def CF_userbased(u_vec,K,data,indxs=False):
    def FindKNeighbours(r,data,K):
        neighs = []
        cnt=0
        for u in xrange(len(data)):
            if data[u,r]>0 and cnt<K:
               neighs.append(data[u])
               cnt +=1
            elif cnt==K:
               break
        return np.array(neighs)

    def CalcRating(u_vec,r,neighs):
        rating = 0.
        den = 0.
        for j in xrange(len(neighs)):
            rating += neighs[j][-1]*float(neighs[j][r]-neighs[j][neighs[j]>0][:-1].mean())
            den += abs(neighs[j][-1])
        if den>0:
            rating = np.round(u_vec[u_vec>0].mean()+(rating/den),0)
        else:
            rating = np.round(u_vec[u_vec>0].mean(),0)
        if rating>5:
            return 5.
        elif rating<1:
            return 1.
        return rating
    #add similarity col
    data = data.astype(float)
    nrows = len(data)
    ncols = len(data[0])
    data_sim = np.zeros((nrows,ncols+1))
    data_sim[:,:-1] = data
    #calc similarities:
    for u in xrange(nrows):
        if np.array_equal(data_sim[u,:-1],u_vec)==False: #list(data_sim[u,:-1]) != list(u_vec):
           data_sim[u,ncols] = sim(data_sim[u,:-1],u_vec,'pearson')
        else:
           data_sim[u,ncols] = 0.
    #order by similarity:
    data_sim =data_sim[data_sim[:,ncols].argsort()][::-1]
    #find the K users for each item not rated:
    u_rec = np.zeros(len(u_vec))
    for r in xrange(ncols):
        if u_vec[r]==0:
           neighs = FindKNeighbours(r,data_sim,K)
           #calc the predicted rating
           u_rec[r] = CalcRating(u_vec,r,neighs)
    if indxs:
           #take out the rated movies
           seenindxs = [indx for indx in xrange(len(u_vec)) if u_vec[indx]>0]
           u_rec[seenindxs] = -1
           recsvec = np.argsort(u_rec)[::-1][np.argsort(u_rec)>0]

           return recsvec
    return u_rec
```

## Item based collaborative filtering

In [32]:
```
class CF_itembased(object):
    def __init__(self,data):
        #calc item similarities matrix
        nitems = len(data[0])
        self.data = data
        self.simmatrix = np.zeros((nitems,nitems))
        for i in xrange(nitems):
            for j in xrange(nitems):
                if j>=i:#triangular matrix
                    self.simmatrix[i,j] = sim(data[:,i],data[:,j])
                else:
                    self.simmatrix[i,j] = self.simmatrix[j,i]

    def GetKSimItemsperUser(self,r,K,u_vec):
        items = np.argsort(self.simmatrix[r])[::-1]
        items = items[items!=r]
        cnt=0
        neighitems = []
        for i in items:
            if u_vec[i]>0 and cnt<K:
                neighitems.append(i)
                cnt+=1
            elif cnt==K:
                break
        return neighitems

    def CalcRating(self,r,u_vec,neighitems):
        rating = 0.
        den = 0.
        for i in neighitems:
            rating +=  self.simmatrix[r,i]*u_vec[i]
            den += abs(self.simmatrix[r,i])
        if den>0:
            rating = np.round(rating/den,0)
        else:
            rating = np.round(self.data[:,r][self.data[:,r]>0].mean(),0)
        return rating

    def CalcRatings(self,u_vec,K,indxs=False):
        #u_rec = copy.copy(u_vec)
        u_rec = np.zeros(len(u_vec))
        for r in xrange(len(u_vec)):
            if u_vec[r]==0:
                neighitems = self.GetKSimItemsperUser(r,K,u_vec)
                #calc predicted rating
                u_rec[r] = self.CalcRating(r,u_vec,neighitems)
        if indxs:
            #take out the rated movies
            seenindxs = [indx for indx in xrange(len(u_vec)) if u_vec[indx]>0]
            u_rec[seenindxs]=-1
            recsvec = np.argsort(u_rec)[::-1][np.argsort(u_rec)>0]

            return recsvec
        return u_rec
```

## Simplest item-based collaborative filtering: slope one

In [34]:

```
class SlopeOne(object):
    def __init__(self,Umatrix):
        #calc item similarities matrix
        nitems = len(Umatrix[0])
        self.difmatrix = np.zeros((nitems,nitems))
        self.nratings = np.zeros((nitems,nitems))
        def diffav_n(x,y):
            xy = np.vstack((x, y)).T
            xy = xy[(xy[:,0]>0) & (xy[:,1]>0)]
            nxy = len(xy)
            if nxy == 0:
                #print 'no common'
                return [1000.,0]
            return [float(sum(xy[:,0])-sum(xy[:,1]))/nxy,nxy]

        for i in xrange(nitems):
            for j in xrange(nitems):
                if j>=i:#triangular matrix
                   self.difmatrix[i,j],self.nratings[i,j] = diffav_n(Umatrix[:,i],Umatrix[:,j])
                else:
                   self.difmatrix[i,j] = -self.difmatrix[j,i]
                   self.nratings[i,j] = self.nratings[j,i]

    def GetKSimItemsperUser(self,r,K,u_vec):
        items = np.argsort(self.difmatrix[r])
        items = items[items!=r]
        cnt=0
        neighitems = []
        for i in items:
            if u_vec[i]>0 and cnt<K:
               neighitems.append(i)
               cnt+=1
            elif cnt==K:
               break
        return neighitems

    def CalcRating(self,r,u_vec,neighitems):
        rating = 0.
        den = 0.
        for i in neighitems:
            if abs(self.difmatrix[r,i])!=1000:
               rating +=  (self.difmatrix[r,i]+u_vec[i])*self.nratings[r,i]
               den += self.nratings[r,i]
        if den==0:
            #print 'no similar diff'
            return 0.
        rating = np.round(rating/den,0)
        if rating >5:
            return 5.
        elif rating <1.:
            return 1.
        return rating

    def CalcRatings(self,u_vec,K):
        #u_rec = copy.copy(u_vec)
        u_rec = np.zeros(len(u_vec))
        for r in xrange(len(u_vec)):
            if u_vec[r]==0:
               neighitems = self.GetKSimItemsperUser(r,K,u_vec)
               #calc predicted rating
               u_rec[r] = self.CalcRating(r,u_vec,neighitems)
        return u_rec
```

# Model-based Collaborative Filtering

## Alternative least square (ALS)

In [12]:
```
def ALS(Umatrix, K, iterations=50, l=0.001, tol=0.001):

    nrows = len(Umatrix)
    ncols = len(Umatrix[0])
    P = np.random.rand(nrows,K)
    Q = np.random.rand(ncols,K)
    Qt = Q.T
    err = 0.
    Umatrix = Umatrix.astype(float)
    mask = Umatrix>0.
    mask[mask==True]=1
    mask[mask==False]=0
    mask = mask.astype(np.float64, copy=False)
    for it in xrange(iterations):
        for u, mask_u in enumerate(mask):
            P[u] = np.linalg.solve(np.dot(Qt, np.dot(np.diag(mask_u), Qt.T)) + l*np.eye(K),
                                   np.dot(Qt, np.dot(np.diag(mask_u), Umatrix[u].T))).T
        for i, mask_i in enumerate(mask.T):
            Qt[:,i] = np.linalg.solve(np.dot(P.T, np.dot(np.diag(mask_i), P)) + l*np.eye(K),
                                   np.dot(P.T, np.dot(np.diag(mask_i), Umatrix[:,i])))
        err=np.sum((mask*(Umatrix - np.dot(P, Qt)))**2)
        if err < tol:
            break
    return np.round(np.dot(P,Qt),0)
```

## Stochastic gradient descent (SGD)

In [11]:
```
def SGD(Umatrix, K, iterations=100, alpha=0.00001, l=0.001, tol=0.001):

    nrows = len(Umatrix)
    ncols = len(Umatrix[0])
    P = np.random.rand(nrows,K)
    Q = np.random.rand(ncols,K)
    Qt = Q.T
    cost=-1
    for it in xrange(iterations):
        for i in xrange(nrows):
            for j in xrange(ncols):
                if Umatrix[i][j] > 0:
                   eij = Umatrix[i][j] -np.dot(P[i,:],Qt[:,j])
                   for k in xrange(K):
                       P[i][k] += alpha*(2*eij*Qt[k][j]-l*P[i][k])
                       Qt[k][j] += alpha*(2*eij*P[i][k]-l*Qt[k][j])
        cost = 0
        for i in xrange(nrows):
            for j in xrange(ncols):
                if Umatrix[i][j]>0:
                   cost += pow(Umatrix[i][j]-np.dot(P[i,:],Qt[:,j]),2)
                   for k in xrange(K):
                       cost += float(l/2.0)*(pow(P[i][k],2)+pow(Qt[k][j],2))
        if cost < tol:
            break
    return np.round(np.dot(P,Qt),0)
```

## Non-negative matrix factorization (NMF)

In [13]:
```
from sklearn.decomposition import NMF
def NMF_alg(Umatrix,K,inp='none',l=0.001):
    R_tmp = copy.copy(Umatrix)
    R_tmp = R_tmp.astype(float)
    #imputation
    if inp != 'none':
        R_tmp = imputation(inp,Umatrix)
    nmf = NMF(n_components=K,alpha=l)
    P = nmf.fit_transform(R_tmp)
    R_tmp = np.dot(P,nmf.components_)
    return R_tmp
```

## Singular value decomposition (SVD)

In [14]:
```
from sklearn.decomposition import TruncatedSVD
def SVD(Umatrix,K,inp='none'):
    R_tmp = copy.copy(Umatrix)
    R_tmp = R_tmp.astype(float)
    #imputation
    if inp != 'none':
        R_tmp = imputation(inp,Umatrix)

    means = np.array([ R_tmp[i][R_tmp[i]>0].mean() for i in xrange(len(R_tmp))]).reshape(-1,1)
    R_tmp = R_tmp-means
    svd = TruncatedSVD(n_components=K, random_state=4)
    R_k = svd.fit_transform(R_tmp)
    R_tmp = svd.inverse_transform(R_k)
    R_tmp = means+R_tmp

    return np.round(R_tmp,0)
```

In [15]:
```
def SVD_EM(Umatrix,K,inp='none',iterations=50,tol=0.001):
    R_tmp = copy.copy(Umatrix)
    R_tmp = R_tmp.astype(float)
    nrows = len(Umatrix)
    ncols = len(Umatrix[0])
    #imputation
    if inp != 'none':
        R_tmp = imputation(inp,Umatrix)
    #define svd
    svd = TruncatedSVD(n_components=K, random_state=4)
    err = -1
    for it in xrange(iterations):
        #m-step
        R_k = svd.fit_transform(R_tmp)
        R_tmp = svd.inverse_transform(R_k)
        #e-step and error evaluation
        err = 0
        for i in xrange(nrows):
            for j in xrange(ncols):
                if Umatrix[i][j]>0:
                   err += pow(Umatrix[i][j]-R_tmp[i][j],2)
                   R_tmp[i][j] = Umatrix[i][j]

        if err < tol:
            print it,'toll reached!'
            break
    return np.round(R_tmp,0)
```

# Content-based Filtering (CBF) methods

In [ ]:
```
#matrix movies's content
movieslist = [int(m.split(';')[-1]) for m in dfout.columns[1:]]
moviescats = ['unknown','Action','Adventure','Animation','Children\'s','Comedy','Crime','Documentary',
              'Drama','Fantasy','Film-Noir','Horror','Musical','Mystery',
              'Romance','Sci-Fi','Thriller','War','Western']
dfout_movies =  pd.DataFrame(columns=['movie_id']+moviescats)
startcatsindx = 5
cnt= 0
for m in movieslist:
    dfout_movies.loc[cnt] = [m]+df_info.iloc[m-1][startcatsindx:].tolist()
    cnt +=1
print dfout_movies.head()

dfout_movies.to_csv('data/movies_content.csv',index=None)
```

# Item's features average method

In [16]:
```
class CBF_averageprofile(object):
    def __init__(self,Movies,Movieslist):
        #calc user profiles:
        self.nfeatures = len(Movies[0])
        self.Movieslist = Movieslist
        self.Movies = Movies

    def GetRecMovies(self,u_vec,indxs=False):
        #generate user profile
        nmovies = len(u_vec)
        nfeatures = self.nfeatures
        mean_u = u_vec[u_vec>0].mean()
        diff_u = u_vec-mean_u
        features_u = np.zeros(nfeatures).astype(float)
        cnts = np.zeros(nfeatures)
        for m in xrange(nmovies):
            if u_vec[m]>0:#u has rated m
               features_u += self.Movies[m]*(diff_u[m])
               cnts += self.Movies[m]
        #average:
        for m in xrange(nfeatures):
            if cnts[m]>0:
               features_u[m] = features_u[m]/float(cnts[m])

        #calc sim:
        sims = np.zeros(nmovies)
        for m in xrange(nmovies):
            if u_vec[m]==0:#sim only for movies not yet rated by the user
               sims[m] = sim(features_u,self.Movies[m])
        #order movies
        order_movies_indxs = np.argsort(sims)[::-1]
        if indxs:
            return order_movies_indxs
        return self.Movieslist[order_movies_indxs]
```

## Regularized linear regression method

In [35]:
```
class CBF_regression(object):
    def __init__(self,Movies,Umatrix,alpha=0.01,l=0.0001,its=50,tol=0.001):
        #calc parameters:
        self.nfeatures = len(Movies[0])+1#intercept
        nusers = len(Umatrix)
        nmovies = len(Umatrix[0])
        #add intercept col
        movies_feats = np.ones((nmovies,self.nfeatures))
        movies_feats[:,1:] = Movies
        self.movies_feats = movies_feats.astype(float)

        #set Umatrix as float
        self.Umatrix = Umatrix.astype(float)
        #initialize the matrix:
        Pmatrix = np.random.rand(nusers,self.nfeatures)
        Pmatrix[:,0]=1.
        err = 0.
        cost = -1
        for it in xrange(its):
            print 'it:',it,' -- ',cost
            for u in xrange(nusers):
                for f in xrange(self.nfeatures):
                    if f==0:#no regularization
                        for m in xrange(nmovies):
                            if self.Umatrix[u,m]>0:
                               diff = np.dot(Pmatrix[u],self.movies_feats[m])-self.Umatrix[u,m]
                               Pmatrix[u,f] += -alpha*(diff*self.movies_feats[m][f])
                    else:
                        for m in xrange(nmovies):
                            if self.Umatrix[u,m]>0:
                               diff = np.dot(Pmatrix[u],self.movies_feats[m])-self.Umatrix[u,m]
                               Pmatrix[u,f] += -alpha*(diff*self.movies_feats[m][f] +l*Pmatrix[u][f])

            cost = 0
            for u in xrange(nusers):
                for m in xrange(nmovies):
                    if self.Umatrix[u][m]>0:
                       cost += 0.5*pow(Umatrix[u][m]-np.dot(Pmatrix[u],self.movies_feats[m]),2)
                for f in xrange(1,self.nfeatures):
                    cost += float(l/2.0)*(pow(Pmatrix[u][f],2))
            if cost < tol:
               print 'err',cost
               break
        self.Pmatrix = Pmatrix

    def CalcRatings(self,u_vec):
        #find u_vec
        s = 0.
        u_feats = np.zeros(len(self.Pmatrix[0]))
        #in case the user is not present in the utility matrix find the most similar
        for u in xrange(len(self.Umatrix)):
            #print self.Umatrix[u]
            tmps = sim(self.Umatrix[u],u_vec)
            if tmps > s:
                s = tmps
                u_feats = self.Pmatrix[u]
            if s == 1.:
                break
        new_vec = np.zeros(len(u_vec))
        for r in xrange(len(u_vec)):
            if u_vec[r]==0:
                new_vec[r] = np.dot(u_feats,self.movies_feats[r])
        return new_vec
```

# Association rules learning recommendation system

In [36]:

```
class AssociationRules(object):
    def __init__(self,Umatrix,Movieslist,min_support=0.1,min_confidence=0.1,likethreshold=3):
        self.min_support = min_support
        self.min_confidence = min_confidence
        self.Movieslist = Movieslist
        #transform utility matrix to sets of liked items
        nitems = len(Umatrix[0])
        transactions = []
        for u in Umatrix:
            s = [i for i in xrange(len(u)) if u[i]>likethreshold]
            if len(s)>0:
                transactions.append(s)
        #find sets of 2 items
        flat = [item for sublist in transactions for item in sublist]
        inititems = map(frozenset,[ [item] for item in frozenset(flat)])
        set_trans = map(set, transactions)
        sets_init, self.dict_sets_support = self.filterSet(set_trans, inititems)
        setlen = 2
        items_tmp = self.combine_lists(sets_init, setlen)
        self.freq_sets, sup_tmp = self.filterSet(set_trans, items_tmp)
        self.dict_sets_support.update(sup_tmp)
        self.ass_matrix = np.zeros((nitems,nitems))
        for freqset in self.freq_sets:
            #print 'freqset',freqset
            list_setitems = [frozenset([item]) for item in freqset]
            #print "freqSet", freqset, 'H1', list_setitems
            self.calc_confidence_matrix(freqset, list_setitems)

    def filterSet(self,set_trans, likeditems):
        itemscnt = {}
        for id in set_trans:
            for item in likeditems:
                if item.issubset(id):
                    itemscnt.setdefault(item, 0)
                    itemscnt[item] += 1
        num_items = float(len(set_trans))
        freq_sets = []
        dict_sets = {}
        for key in itemscnt:
            support = itemscnt[key] / num_items
            if support >= self.min_support:
                freq_sets.insert(0, key)
            dict_sets[key] = support
        return freq_sets, dict_sets

    def combine_lists(self,freq_sets, setlen):
        setitems_list = []
        nsets = len(freq_sets)
        for i in range(nsets):
            for j in range(i + 1, nsets):
                setlist1 = list(freq_sets[i])[:setlen - 2]
                setlist2 = list(freq_sets[j])[:setlen - 2]
                if set(setlist1) == set(setlist2):
                    setitems_list.append(freq_sets[i].union(freq_sets[j]))
        return setitems_list

    def calc_confidence_matrix(self,freqset, list_setitems):
        for target in list_setitems:
            confidence = self.dict_sets_support[freqset] / self.dict_sets_support[freqset - target]
            if confidence >= self.min_confidence:
                self.ass_matrix[list(freqset - target)[0]][list(target)[0]] = confidence

    def GetRecItems(self,u_vec,indxs=False):
        vec_recs = np.dot(u_vec,self.ass_matrix)
        sortedweight = np.argsort(vec_recs)
        seenindxs = [indx for indx in xrange(len(u_vec)) if u_vec[indx]>0]
        seenmovies = np.array(self.Movieslist)[seenindxs]
        #remove seen items
        recitems = np.array(self.Movieslist)[sortedweight]
        recitems = [m for m in recitems if m not in seenmovies]
        if indxs:
            vec_recs[seenindxs]=-1
            recsvec = np.argsort(vec_recs)[::-1][np.argsort(vec_recs)>0]
            return recsvec
        return recitems[::-1]
```

# Log-likelihood ratios recommendation system method

In [19]:

```
class LogLikelihood(object):
    def __init__(self,Umatrix,Movieslist,likethreshold=3):
        self.Movieslist = Movieslist
        #calculate loglikelihood ratio for each pair
        self.nusers = len(Umatrix)
        self.Umatrix =Umatrix
        self.likethreshold = likethreshold
        self.likerange = range(self.likethreshold+1,5+1)
        self.dislikerange = range(1,self.likethreshold+1)
        self.loglikelihood_ratio()

    def calc_k(self,a,b):
        tmpk = [[0 for j in range(2)] for i in range(2)]
        for ratings in self.Umatrix:
            if ratings[a] in self.likerange and ratings[b] in self.likerange:
               tmpk[0][0] += 1
            if ratings[a] in self.likerange and ratings[b] in self.dislikerange:
                tmpk[0][1] += 1
            if ratings[a] in self.dislikerange and ratings[b] in self.likerange:
                tmpk[1][0] += 1
            if ratings[a] in self.dislikerange and ratings[b] in self.dislikerange:
                tmpk[1][1] += 1
        return tmpk

    def calc_llr(self,k_matrix):
        Hcols=Hrows=Htot=0.0
        if sum(k_matrix[0])+sum(k_matrix[1])==0:
            return 0.
        invN = 1.0/(sum(k_matrix[0])+sum(k_matrix[1]))
        for i in range(0,2):
            if((k_matrix[0][i]+k_matrix[1][i])!=0.0):
               Hcols += invN*(k_matrix[0][i]+k_matrix[1][i])*math.log((k_matrix[0][i]+k_matrix[1][i])*invN )#sum of row
            if((k_matrix[i][0]+k_matrix[i][1])!=0.0):
               Hrows += invN*(k_matrix[i][0]+k_matrix[i][1])*math.log((k_matrix[i][0]+k_matrix[i][1])*invN )#sum of col
            for j in range(0,2):
                if(k_matrix[i][j]!=0.0):
                   Htot +=invN*k_matrix[i][j]*math.log(invN*k_matrix[i][j])
        return 2.0*(Htot-Hcols-Hrows)/invN

    def loglikelihood_ratio(self):
        nitems = len(self.Movieslist)
        self.items_llr= pd.DataFrame(np.zeros((nitems,nitems))).astype(float)
        for i in xrange(nitems):
            for j in xrange(nitems):
                if(j>=i):
                   tmpk=self.calc_k(i,j)
                   self.items_llr.ix[i,j] = self.calc_llr(tmpk)
                else:
                   self.items_llr.ix[i,j] = self.items_llr.iat[j,i]

    def GetRecItems(self,u_vec,indxs=False):
        items_weight = np.dot(u_vec,self.items_llr)
        sortedweight = np.argsort(items_weight)
        seenindxs = [indx for indx in xrange(len(u_vec)) if u_vec[indx]>0]
        seenmovies = np.array(self.Movieslist)[seenindxs]
        #remove seen items
        recitems = np.array(self.Movieslist)[sortedweight]
        recitems = [m for m in recitems if m not in seenmovies]
        if indxs:
            items_weight[seenindxs]=-1
            recsvec = np.argsort(items_weight)[::-1][np.argsort(items_weight)>0]
            return recsvec
        return recitems[::-1]
```

# Hybrid recommendation systems

In [37]:
```
class Hybrid_cbf_cf(object):
    def __init__(self,Movies,Movieslist,Umatrix):
        #calc user profiles:
        self.nfeatures = len(Movies[0])
        self.Movieslist = Movieslist
        self.Movies = Movies.astype(float)
        self.Umatrix_mfeats = np.zeros((len(Umatrix),len(Umatrix[0])+self.nfeatures))
        means = np.array([ Umatrix[i][Umatrix[i]>0].mean() for i in xrange(len(Umatrix))]).reshape(-1,1)
        diffs = np.array([ [Umatrix[i][j]-means[i] if Umatrix[i][j]>0 else 0.
                            for j in xrange(len(Umatrix[i]))  ] for i in xrange(len(Umatrix))])
        self.Umatrix_mfeats[:,:len(Umatrix[0])] = Umatrix#diffs
        self.nmovies = len(Movies)
        #calc item features for each user
        for u in xrange(len(Umatrix)):
            u_vec = Umatrix[u]
            self.Umatrix_mfeats[u,len(Umatrix[0]):] = self.GetUserItemFeatures(u_vec)

    def GetUserItemFeatures(self,u_vec):
        mean_u = u_vec[u_vec>0].mean()
        #diff_u = u_vec-mean_u
        features_u = np.zeros(self.nfeatures).astype(float)
        cnts = np.zeros(self.nfeatures)
        for m in xrange(self.nmovies):
            if u_vec[m]>0:#u has rated m
               features_u += self.Movies[m]*u_vec[m]#self.Movies[m]*(diff_u[m])
               cnts += self.Movies[m]
        #average:
        for m in xrange(self.nfeatures):
            if cnts[m]>0:
               features_u[m] = features_u[m]/float(cnts[m])
        return features_u
```

```
        def CalcRating(u_vec,r,neighs):
            rating = 0.
            den = 0.
            for j in xrange(len(neighs)):
                rating += neighs[j][-1]*float(neighs[j][r]-neighs[j][neighs[j]>0][:-1].mean())
                den += abs(neighs[j][-1])
            if den>0:
                rating = np.round(u_vec[u_vec>0].mean()+(rating/den),0)
            else:
                rating = np.round(u_vec[u_vec>0].mean(),0)
            if rating>5:
                return 5.
            elif rating<1:
                return 1.
            return rating
        #add similarity col
        nrows = len(self.Umatrix_mfeats)
        ncols = len(self.Umatrix_mfeats[0])
        data_sim = np.zeros((nrows,ncols+1))
        data_sim[:,:-1] = self.Umatrix_mfeats
        u_rec = np.zeros(len(u_vec))
        #calc similarities:
        mean = u_vec[u_vec>0].mean()
        u_vec_feats = u_vec#np.array([u_vec[i]-mean if u_vec[i]>0 else 0 for i in xrange(len(u_vec))])
        u_vec_feats = np.append(u_vec_feats,self.GetUserItemFeatures(u_vec))
```

```
        for u in xrange(nrows):
            if np.array_equal(data_sim[u,:-1],u_vec)==False: #list(data_sim[u,:-1]) != list(u_vec):
                data_sim[u,ncols] = sim(data_sim[u,:-1],u_vec_feats)
            else:
                data_sim[u,ncols] = 0.
        #order by similarity:
        data_sim[:,:-1] = self.Umatrix_mfeats
        u_rec = np.zeros(len(u_vec))
        #calc similarities:
        mean = u_vec[u_vec>0].mean()
        u_vec_feats = u_vec#np.array([u_vec[i]-mean if u_vec[i]>0 else 0 for i in xrange(len(u_vec))])
        u_vec_feats = np.append(u_vec_feats,self.GetUserItemFeatures(u_vec))

        for u in xrange(nrows):
            if np.array_equal(data_sim[u,:-1],u_vec)==False: #list(data_sim[u,:-1]) != list(u_vec):
                data_sim[u,ncols] = sim(data_sim[u,:-1],u_vec_feats)
            else:
                data_sim[u,ncols] = 0.
        #order by similarity:
        data_sim =data_sim[data_sim[:,ncols].argsort()][::-1]
        #find the K users for each item not rated:

        for r in xrange(self.nmovies):
            if u_vec[r]==0:
                neighs = FindKNeighbours(r,data_sim,K)
                #calc the predicted rating
                u_rec[r] = CalcRating(u_vec,r,neighs)
        return u_rec
```

In [22]:
```
class Hybrid_svd(object):
    def __init__(self,Movies,Movieslist,Umatrix,K,inp):
        #calc user profiles:
        self.nfeatures = len(Movies[0])
        self.Movieslist = Movieslist
        self.Movies = Movies.astype(float)

        R_tmp = copy.copy(Umatrix)
        R_tmp = R_tmp.astype(float)
        #imputation

        if inp != 'none':
            R_tmp = imputation(inp,Umatrix)
        Umatrix_mfeats = np.zeros((len(Umatrix),len(Umatrix[0])+self.nfeatures))
        means = np.array([ Umatrix[i][Umatrix[i]>0].mean() for i in xrange(len(Umatrix))]).reshape(-1,1)
        diffs = np.array([ [float(Umatrix[i][j]-means[i])
                            if Umatrix[i][j]>0 else float(R_tmp[i][j]-means[i]) for j in xrange(len(Umatrix[i]))  ]
                          for i in xrange(len(Umatrix))])
        Umatrix_mfeats[:,:len(Umatrix[0])] = diffs#R_tmp
        self.nmovies = len(Movies)
        #calc item features for each user
        for u in xrange(len(Umatrix)):
            u_vec = Umatrix[u]
            Umatrix_mfeats[u,len(Umatrix[0]):] = self.GetUserItemFeatures(u_vec)

        #calc svd
        svd = TruncatedSVD(n_components=K, random_state=4)
        R_k = svd.fit_transform(Umatrix_mfeats)
        R_tmp = means+svd.inverse_transform(R_k)
        self.matrix = np.round(R_tmp[:,:self.nmovies],0)


    def GetUserItemFeatures(self,u_vec):
        mean_u = u_vec[u_vec>0].mean()
        diff_u = u_vec-mean_u
        features_u = np.zeros(self.nfeatures).astype(float)
        cnts = np.zeros(self.nfeatures)
        for m in xrange(self.nmovies):
            if u_vec[m]>0:#u has rated m
                features_u += self.Movies[m]*(diff_u[m])#self.Movies[m]*u_vec[m]
                cnts += self.Movies[m]
        #average:
        for m in xrange(self.nfeatures):
            if cnts[m]>0:
                features_u[m] = features_u[m]/float(cnts[m])
        return features_u
```

# Evaluation of the recommendation systems

In [42]:
```
def cross_validation(df,k):
    val_num = int(len(df)/float(k))
    print val_num
    df_trains = []
    df_vals = []
    for i in xrange(k):
        start_val = (k-i-1)*val_num
        end_val = start_val+val_num
        df_trains.append(pd.concat([df[:start_val],df[end_val:]]))
        df_vals.append(df[start_val:end_val])

    return df_trains,df_vals
```

In [23]:
```
import random
def HideRandomRatings(u_vec, ratiovals=0.5):
    u_test = np.zeros(len(u_vec))
    u_vals = np.zeros(len(u_vec))
    cnt = 0
    nratings = len(u_vec[u_vec>0])
    for i in xrange(len(u_vec)):
        if u_vec[i]>0:
            if bool(random.getrandbits(1)) or cnt>=int(nratings*rationvals):
                u_test[i]=u_vec[i]
            else:#random choice to hide the rating:
                cnt +=1
                u_vals[i]=u_vec[i]
    return u_test,u_vals
```

In [24]:
```
#load data
df = pd.read_csv('data/utilitymatrix.csv')
print df.head(4)
df_movies = pd.read_csv('data/movies_content.csv')
movies = df_movies.values[:,1:]
print 'check:::',len(df.columns[1:]),'--',len(df_movies)
movieslist = list(df.columns[1:])
#k-fold cv 5 folds
nfolds = 5
df_trains,df_vals = cross_validation(df,nfolds)
```

In [31]:
```
nmovies = len(df_vals[0].values[:,1:][0])
s = []
tests_vecs_folds = []
for i in xrange(nfolds):
    u_vecs = df_vals[i].values[:,1:]
    vtests = np.empty((0,nmovies),float)
    vvals = np.empty((0,nmovies),float)
    for u_vec in u_vecs:
        u_test,u_vals = HideRandomRatings(u_vec)
        vvals = np.vstack([vvals,u_vals])
        vtests = np.vstack([vtests,u_test])
    vals_vecs_folds.append(vvals)
    tests_vecs_folds.append(vtests)
```

## Root mean square error (RMSE) evaluation

In [43]:
```
def SE(u_preds,u_vals):
    nratings = len(u_vals)
    se = 0.
    cnt = 0
    for i in xrange(nratings):
        if u_vals[i]>0:
            se +=  (u_vals[i]-u_preds[i])*(u_vals[i]-u_preds[i])
            cnt += 1
    return se,cnt
```

In [40]:

```
err_itembased = 0.
cnt_itembased = 0
err_userbased = 0.
cnt_userbased = 0
err_slopeone = 0.
cnt_slopeone = 0
err_cbfcf = 0.
cnt_cbfcf = 0
for i in xrange(nfolds):
    Umatrix = df_trains[i].values[:,1:]
    cfitembased = CF_itembased(Umatrix)
    cfslopeone = SlopeOne(Umatrix)
    cbfcf = Hybrid_cbf_cf(movies,movieslist,Umatrix)
    print 'fold:',i+1
    vec_vals = vals_vecs_folds[i]
    vec_tests = tests_vecs_folds[i]
    for j in xrange(len(vec_vals)):
        u_vals = vec_vals[j]
        u_test = vec_tests[j]
        #cbfcf
        u_preds = cbfcf.CalcRatings(u_test,5)
        e,c =  SE(u_preds,u_vals)
        err_cbfcf +=e
        cnt_cbfcf +=c
        #cf_userbased
        u_preds = CF_userbased(u_test,5,Umatrix)
        e,c =  SE(u_preds,u_vals)
        err_userbased +=e
        cnt_userbased +=c
        #cf_itembased
        u_preds = cfitembased.CalcRatings(u_test,5)
        e,c =  SE(u_preds,u_vals)
        err_itembased +=e
        cnt_itembased +=c
        #slope one
        u_preds = cfslopeone.CalcRatings(u_test,5)
        e,c =  SE(u_preds,u_vals)
        err_slopeone +=e
        cnt_slopeone +=c
rmse_userbased = np.sqrt(err_userbased/float(cnt_userbased))
rmse_itembased = np.sqrt(err_itembased/float(cnt_itembased))
rmse_slopeone = np.sqrt(err_slopeone/float(cnt_slopeone))
print 'user_userbased rmse:',rmse_userbased,'--',cnt_userbased
print 'user_itembased rmse:',rmse_itembased,'--',cnt_itembased
print 'slope one rmse:',rmse_slopeone,'--',cnt_slopeone

rmse_cbfcf = np.sqrt(err_cbfcf/float(cnt_cbfcf))
print 'cbfcf rmse:',rmse_cbfcf,'---',cnt_cbfcf
```

In [63]:

```
err_svd = 0.
cnt_svd = 0
err_svd_em = 0.
cnt_svd_em = 0
err_als = 0.
cnt_als = 0
err_cbfreg = 0.
cnt_cbfreg = 0
for i in xrange(nfolds):
    Umatrix = df_trains[i].values[:,1:]
    print 'fold:',i+1
    teststartindx = len(Umatrix)
    vals_vecs = vals_vecs_folds[i]
    tests_vecs = tests_vecs_folds[i]
    for k in xrange(len(vals_vecs)):
        u_vals = vals_vecs[k]
        u_test = tests_vecs[k]
        #add test vector to utility matrix
        Umatrix = np.vstack([Umatrix,u_test])

    #svd_em_matrix = Hybrid_svd(movies,movieslist,Umatrix,20,'useraverage').matrix#SVD_EM(Umatrix,20,'useraverage',1)
    svd_matrix = SVD(Umatrix,20,'itemaverage')
    cbf_reg = CBF_regression(movies,Umatrix)
    #als_umatrix = SGD(Umatrix,20,50)#ALS(Umatrix,20,50)#NMF_alg(Umatrix,20,'itemaverage',0.001)
    #evaluate errors
    for indx in xrange(len(vals_vecs)):
        #e,c =  SE(als_umatrix[teststartindx+indx],vals_vecs[indx])
        #err_als += e
        #cnt_als += c
        u_preds = cbf_reg.CalcRatings(Umatrix[teststartindx+indx])
        e,c = SE(u_preds,vals_vecs[indx])
        err_cbfreg +=e
        cnt_cbfreg +=c

        e,c = SE(svd_matrix[teststartindx+indx],vals_vecs[indx])
        err_svd +=e
        cnt_svd +=c
        #e,c = SE(svd_em_matrix[teststartindx+indx],vals_vecs[indx])
        #err_svd_em +=e
        #cnt_svd_em +=c

if cnt_svd==0: cnt_svd=1
if cnt_svd_em==0: cnt_svd_em=1
if cnt_als==0: cnt_als=1
if cnt_cbfreg==0: cnt_cbfreg=1

rmse_als = np.sqrt(err_als/float(cnt_als))
rmse_svd = np.sqrt(err_svd/float(cnt_svd))
rmse_svd_em = np.sqrt(err_svd_em/float(cnt_svd_em))
rmse_cbfreg = np.sqrt(err_cbfreg/float(cnt_cbfreg))

print 'svd rmse:',rmse_svd,'--',cnt_svd
#print 'svd_em rmse:',rmse_svd_em,'--',cnt_svd_em
#print 'als rmse:',rmse_als,'--',cnt_als
print 'cbfreg rmse:',rmse_cbfreg,'--',cnt_cbfreg
```

# Classification metrics

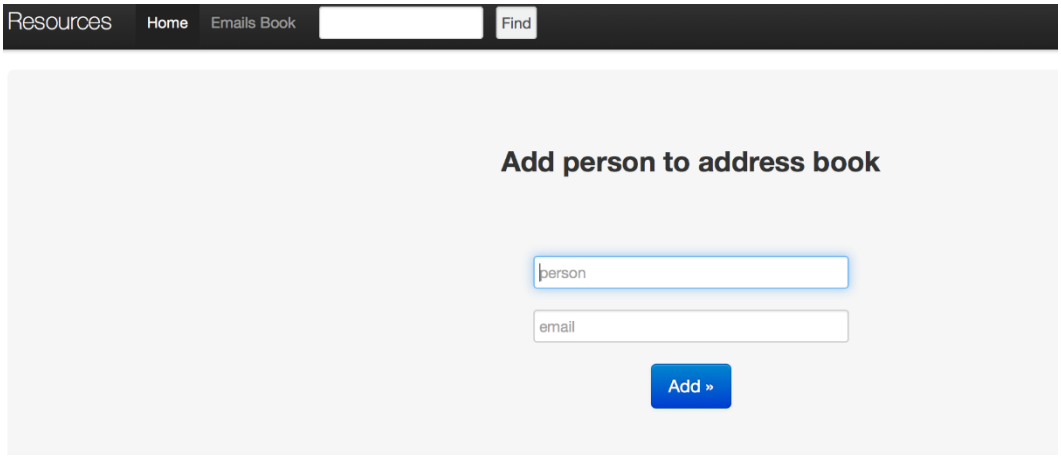
In [33]:
```
def ClassificationMetrics(vec_vals,vec_recs,likethreshold=3,shortlist=50,ratingsval=False,vec_test=None):
    #convert vals in indxs vec
    indxs_like = [i for i in xrange(len(vec_vals)) if vec_vals[i]>likethreshold]
    indxs_dislike = [i for i in xrange(len(vec_vals)) if vec_vals[i]<=likethreshold and vec_vals[i]>0]
    cnt = len(indxs_like)+len(indxs_dislike)
    indxs_rec = []
    if ratingsval:
        #convert ratings into items's list
        if vec_test==None:
            raise 'Error no test vector'
        indxs_rec = [i for i in xrange(len(vec_recs)) if vec_recs[i]>likethreshold and vec_test[i]<1][:shortlist]
    else:
        #consider only the first slot of recs
        indxs_rec = vec_recs[:shortlist]

    tp = len(set(indxs_rec).intersection(set(indxs_like)))
    fp = len(set(indxs_rec).intersection(set(indxs_dislike)))
    fn = len(set(indxs_like)^(set(indxs_rec).intersection(set(indxs_like))))
    precision = 0.
    if tp+fp>0:
        precision = float(tp)/(tp+fp)
    recall = 0.
    if tp+fn>0:
        recall = float(tp)/(tp+fn)
    f1 = 0.
    if recall+precision >0:
        f1 = 2.*precision*recall/(precision+recall)

    return np.array([precision,recall,f1]),cnt
```

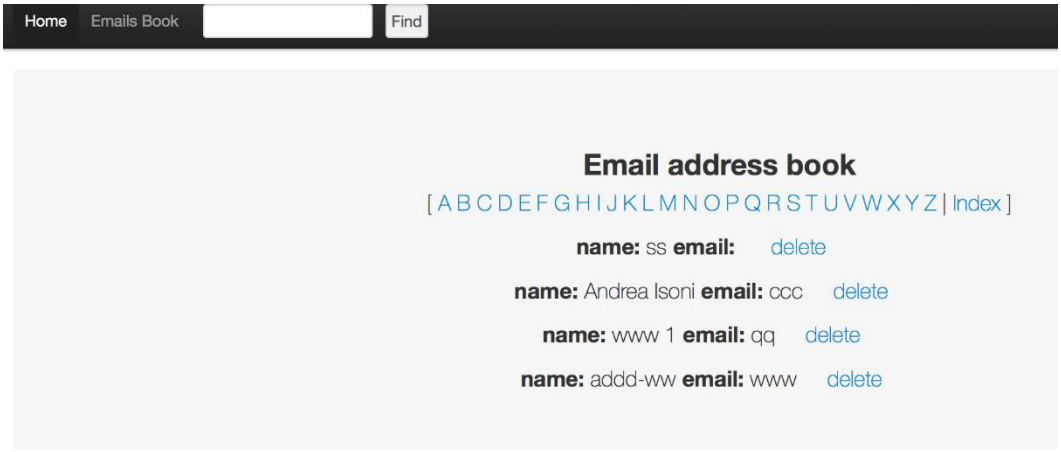
# 6
# Getting Started with Django

## Writing an app – most important features

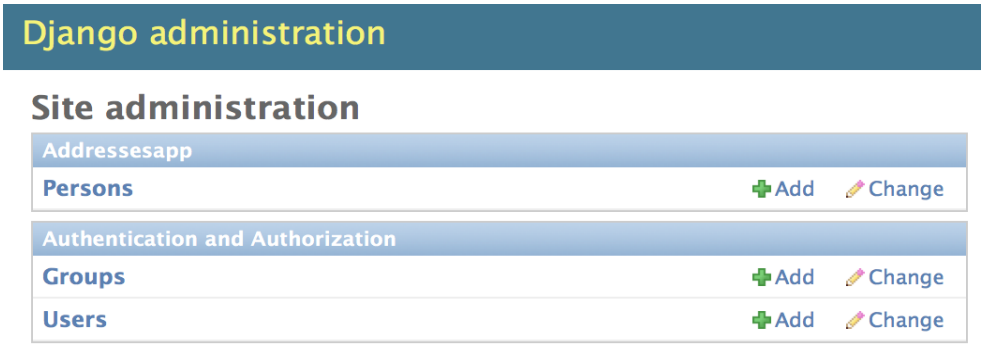
### URL and views behind HTML web pages

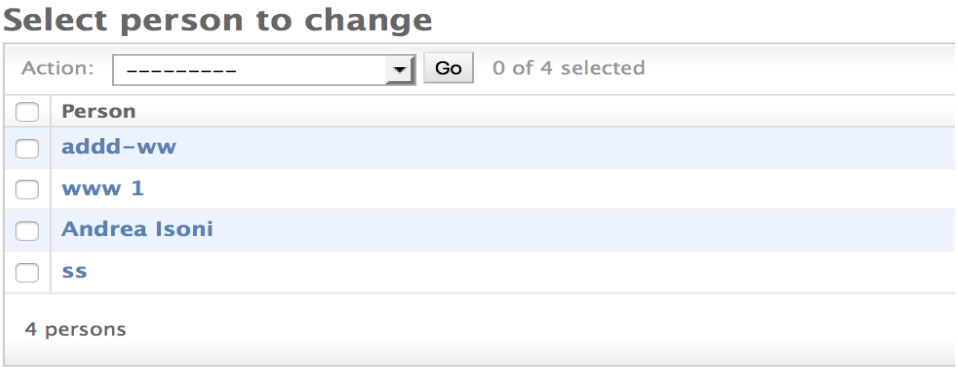
### HTML pages



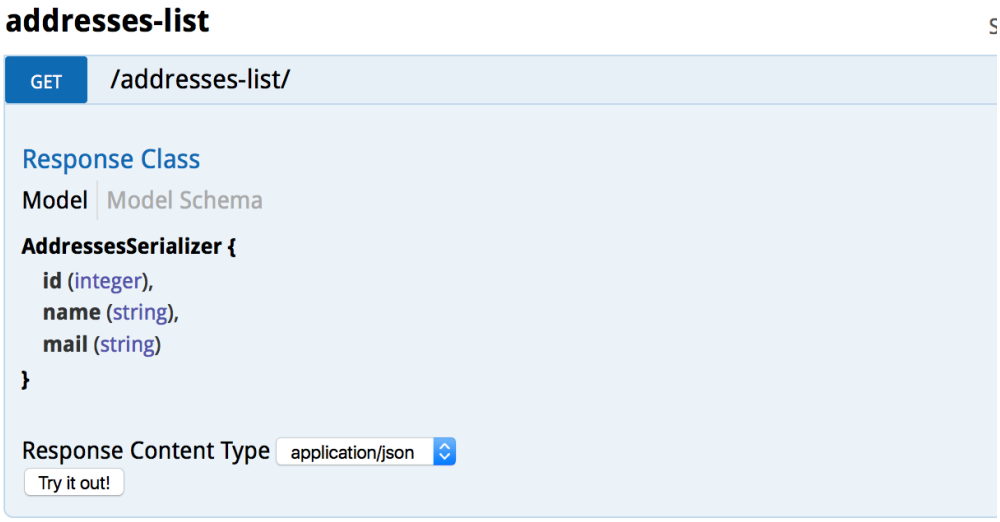

Home
Emails Book
Find
Email address book
[ A B C D E F G H I J K L M N O P Q R S T U V W X Y Z | Index ]
name: ss email: delete
name: Andrea Isoni email: ccc delete
name: www 1 email: qq delete
name: add-ww email: www delete

# Admin


Django administration
Site administration
Addressesapp
Persons
Add
Change
Authentication and Authorization
Groups
Add
Change
Users
Add
Change



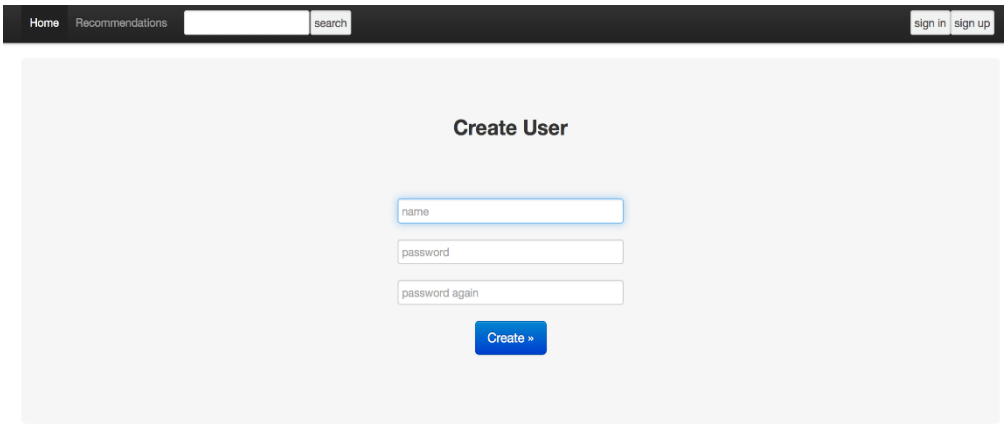

## RESTful application programming interfaces (APIs)

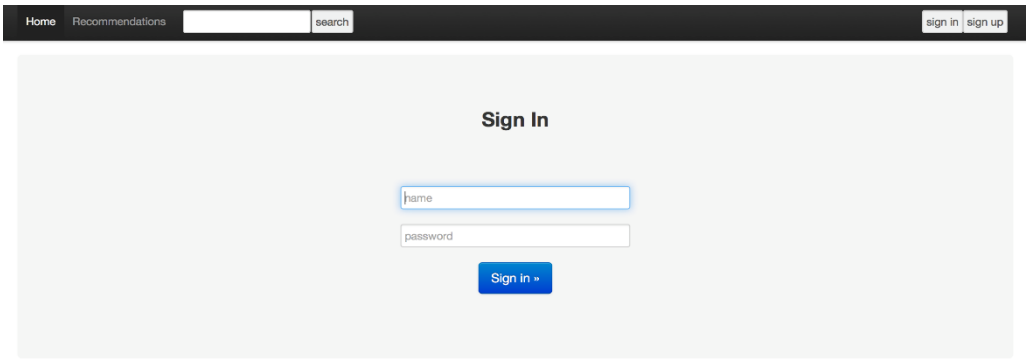



# 7

# Movie Recommendation System Web Application

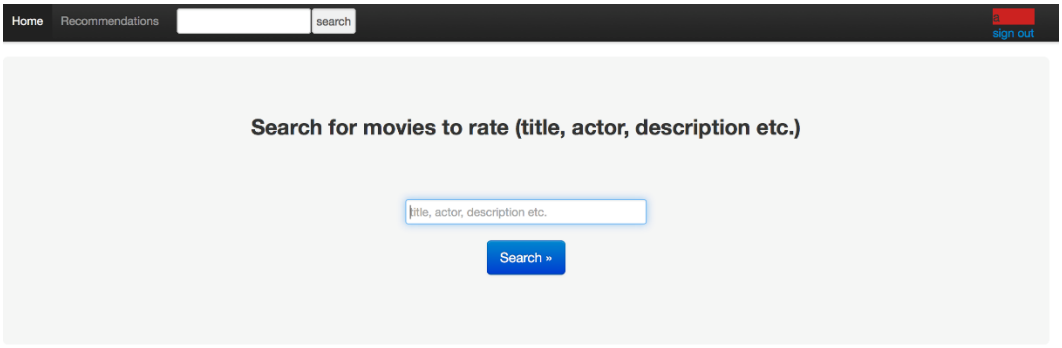
## User sign up login/logout implementation

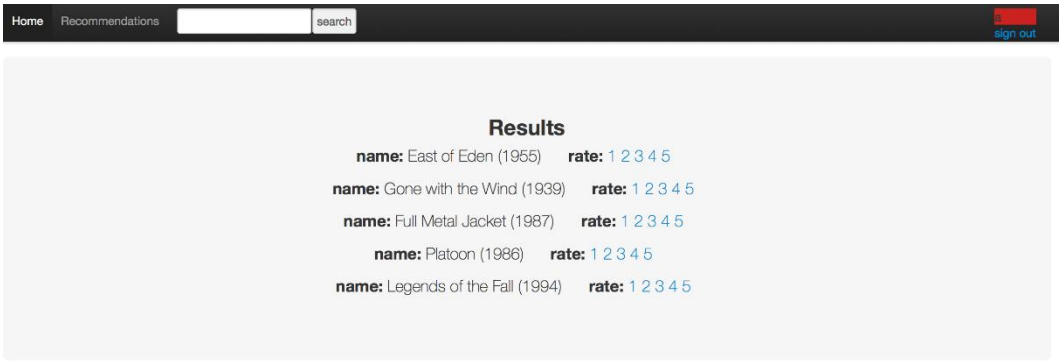







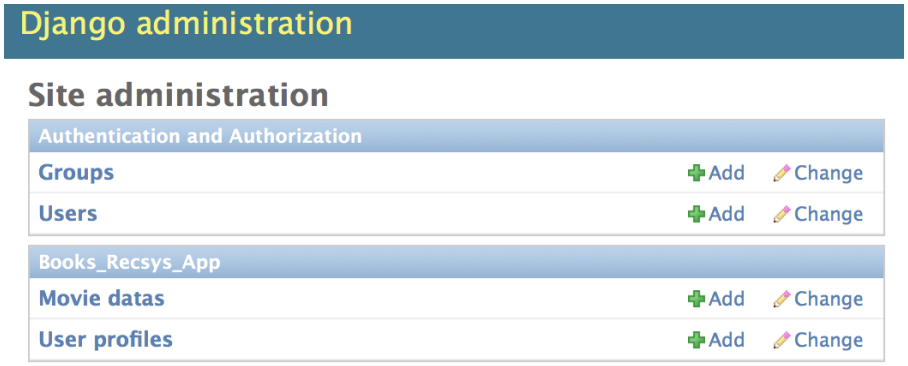
# Information retrieval system (movies query)






# Recommendation systems

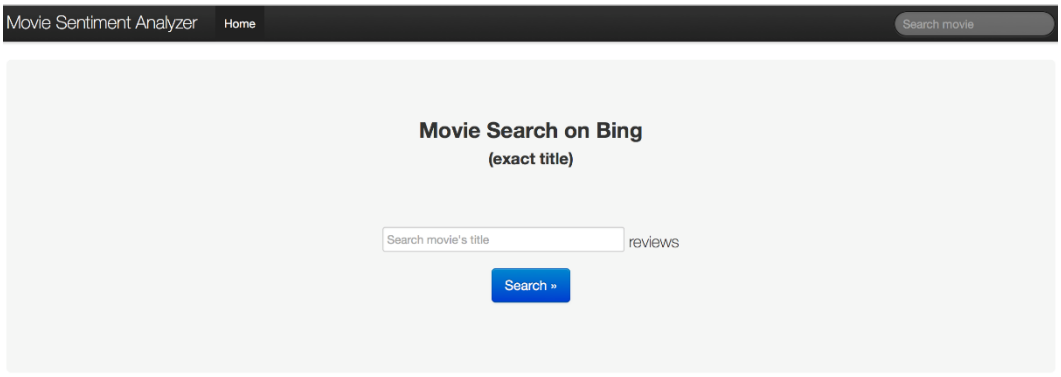

# Admin interface and API



# 8

# Sentiment Analyser application on Movie Reviews

## Application's usage overview



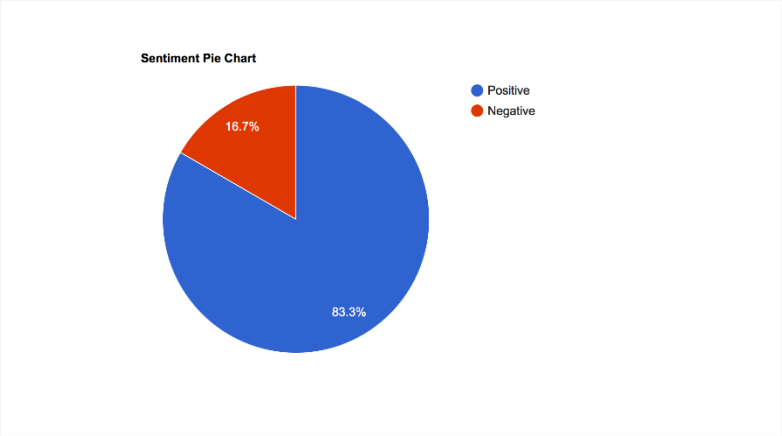
Movie Sentiment Analyzer Home batman vs superman dawn

## Movie Sentiment Analysis

**Reviews Classified : 18**

**Positive Reviews : 15**

**Negative Reviews : 3**




calculate page rank

scrape and calculate page rank (may take a long time)

Movie Sentiment Analyzer Home About Search/@movie

## PageRank of movie reviews

**Url: http://indianexpress.com/photos/entertainment-gallery/batman-vs-superman-dawn-of-justice-movie-review-in-pics/ pagerank: 1.0**

**Url: http://timesofindia.indiatimes.com/entertainment/english/movie-reviews/Batman-v-Superman-Dawn-of-Justice/movie-review/51539160.cms pagerank: 1.0**

**Url: http://timesofindia.indiatimes.com/entertainment/english/movie-reviews/Batman-v-Superman-Dawn-of-Justice/movie-review/51539160.cms?tabtype=spoiler pagerank: 1.0**

**Url: http://wabi.tv/2016/03/27/batman-v-superman-dawn-of-justice-movie-review/ pagerank: 1.0**

**Url: http://worldviewreviews.com/2016/03/26/batman-v-superman-review/ pagerank: 1.0**

**Url: http://www.rottentomatoes.com/m/batman_v_superman_dawn_of_justice/ pagerank: 1.0**

**Url: https://www.commonsensemedia.org/movie-reviews/batman-v-superman-dawn-of-justice pagerank: 1.0**

**Url: http://moviefloss.com/batman-vs-superman-dawn-of-justice-movie-review/ pagerank: 1.0**

**Url: http://www.sakshipost.com/index.php?option=com_content&view=article&id=77578&catid=24&Itemid=178%20&pfrom=home-sakshi-post pagerank: 1.0**

**Url: http://edition.cnn.com/2016/03/23/entertainment/batman-v-superman-review-thr-feat/ pagerank: 1.0**

**Url: http://www.thereelword.net/batman-v-superman-dawn-of-justice-movie-review/ pagerank: 1.0**

**Url: http://www.rogerebert.com/reviews/batman-v-superman-dawn-of-justice-2016 pagerank: 0.00335595495563**

# Admin and API