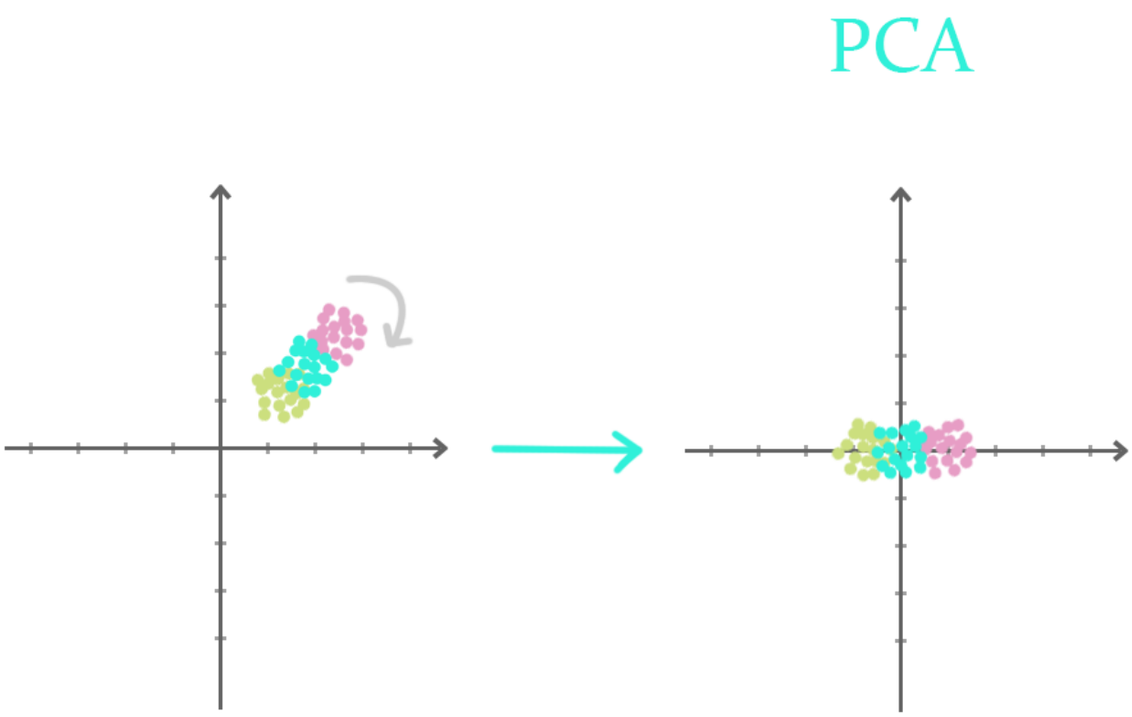
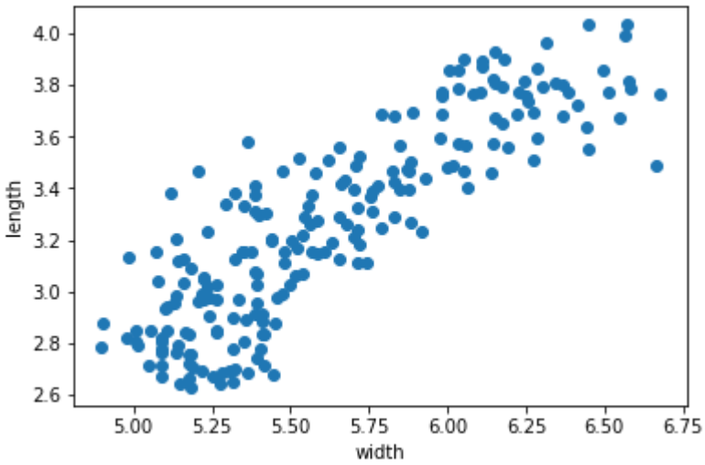
PCA Transformation : Principal Component Analysis

- Decorrelation :
  - rotates the samples to be aligned with axes
  - shift data samples so they have means zero
- Reduce dimension

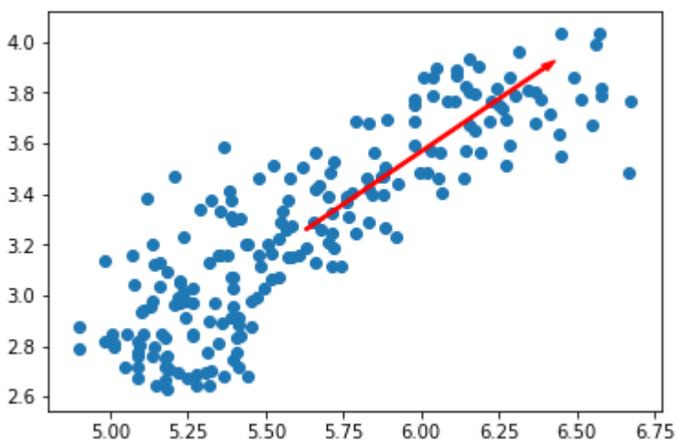


In [17]:
```
import numpy as np
from scipy.stats import pearsonr

%cd c://
seed = np.genfromtxt('seeds-width-vs-length.csv',delimiter=',',dtype=None)

width = seed[:,0]
length = seed[:,1]

correlation,pValue = pearsonr(width, length)
print(correlation)


plt.plot(width, length, 'o')
plt.xlabel('width')
plt.ylabel('length')
plt.show()
```

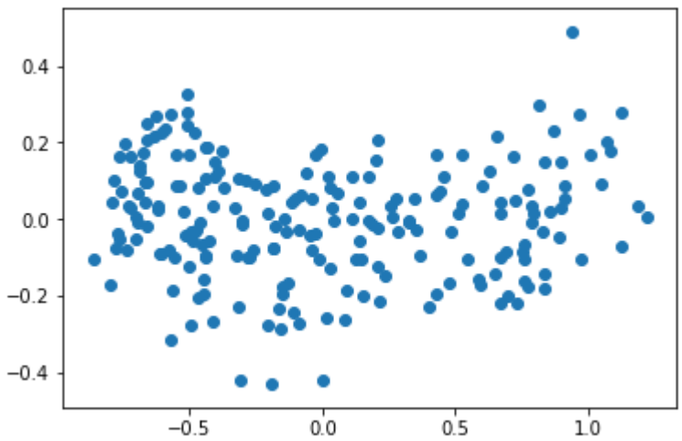
```
c:\
0.860414937714
```



In [18]:
```
from sklearn.decomposition import PCA

pca = PCA()
pca.fit(seed)
transformed = pca.transform(seed)

mean = pca.mean_
FPC = pca.components_[0,:]

plt.scatter(width, length)
plt.arrow(mean[0], mean[1], FPC[0], FPC[1], color='red',width=0.01)
#plt.axis('equal')# Keep axes on same scale
plt.show()
```

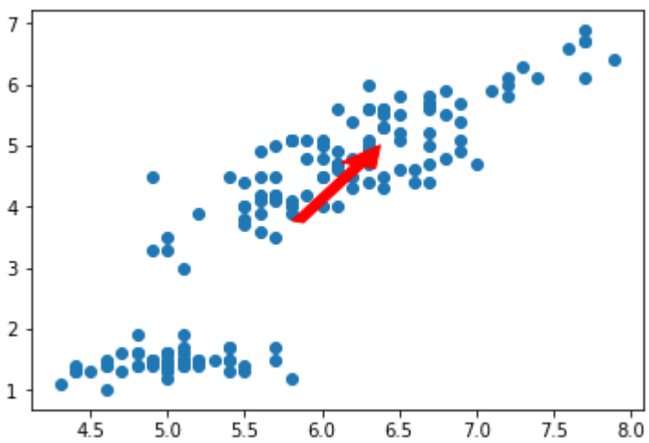


In [19]:
```
plt.scatter(transformed[:,0],transformed[:,1],)
plt.show()
```

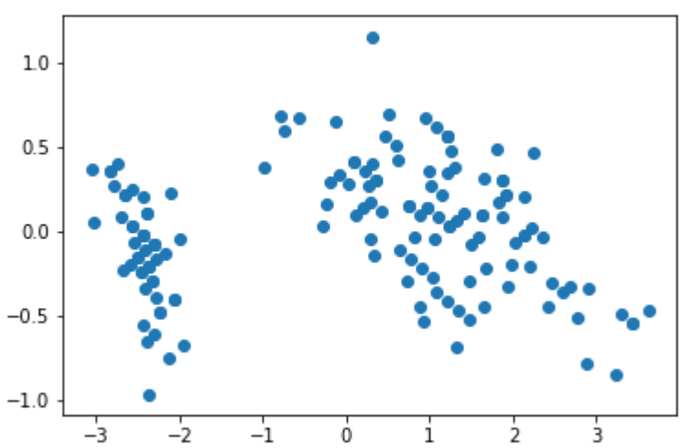


Iris Dataset

In [20]:
```
iris_model = PCA()
iris_model.fit(iris.data[:,[0,2]])
transform = iris_model.transform(iris.data[:,[0,2]])

mean = iris_model.mean_
First_PC = iris_model.components_[0,:]
First_PC

plt.scatter(iris.data[:,0],iris.data[:,2])
plt.arrow(mean[0], mean[1], First_PC[0], First_PC[1],color='red',width=0.08)
plt.show()
```



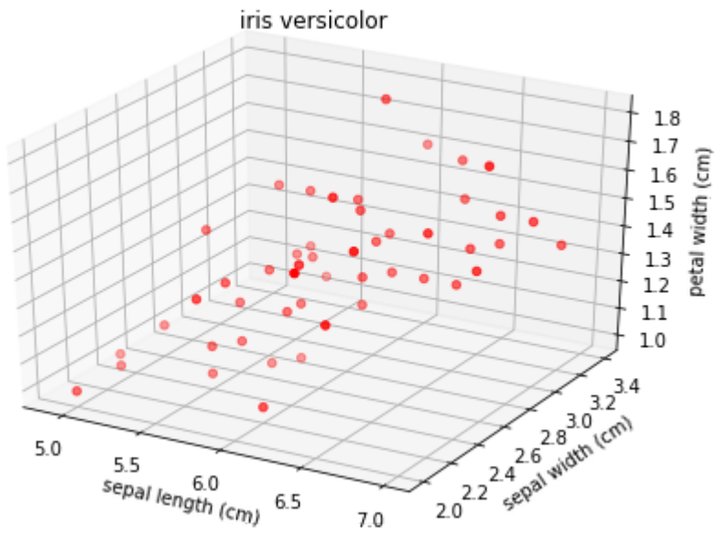
In [21]:
```
plt.scatter(transform[:,0], transform[:,1])
plt.show()
```



Intrinsic dimension

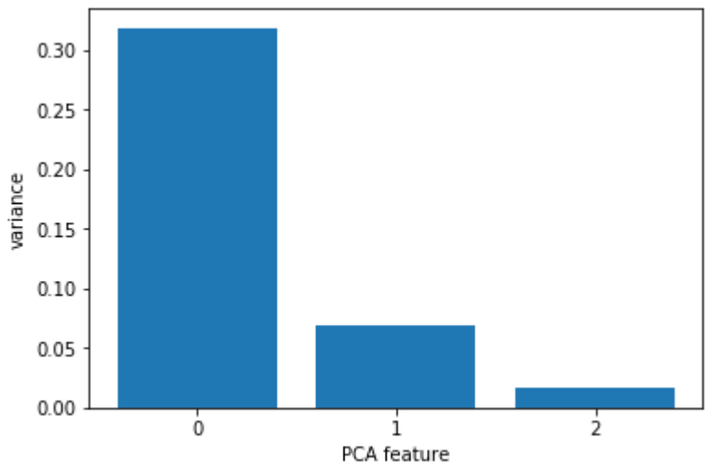
the intrinsic dimension is number of features needed to approximate the dataset

In [22]:
```
from mpl_toolkits.mplot3d import Axes3D
fig = plt.figure()
ax = Axes3D(fig)

versicolor = iris.data[50:100]
versicolor = versicolor[:,[0,1,3]]

ax.scatter(versicolor[:,0], versicolor[:,1], versicolor[:,2],c='r')

ax.set_title('iris versicolor')

ax.set_xlabel('sepal length (cm)')
ax.set_ylabel('sepal width (cm)')
ax.set_zlabel('petal width (cm)')

plt.show()
```



In [23]:
```
fig2 = plt.figure()
ax2 = Axes3D(fig2)


# Create a PCA instance: pca
VC_pca = PCA()
VC_pca.fit(versicolor)
transformed = VC_pca.transform(versicolor)

# Plot the explained variances
PC = VC_pca.components_
nfeatures = range(VC_pca.n_components_)

#ax2.scatter(transformed[:,0], transformed[:,1], transformed[:,2],c='b')
#ax2.set_title('iris versicolor')
#ax2.set_xlabel('sepal length (cm)')
#ax2.set_ylabel('sepal width (cm)')
#ax2.set_zlabel('petal width (cm)')
#plt.show()
```

In [63]:
```
plt.bar(nfeatures, VC_pca.explained_variance_)
plt.xlabel('PCA feature')
plt.ylabel('variance')
plt.xticks(nfeatures)
plt.show()
```



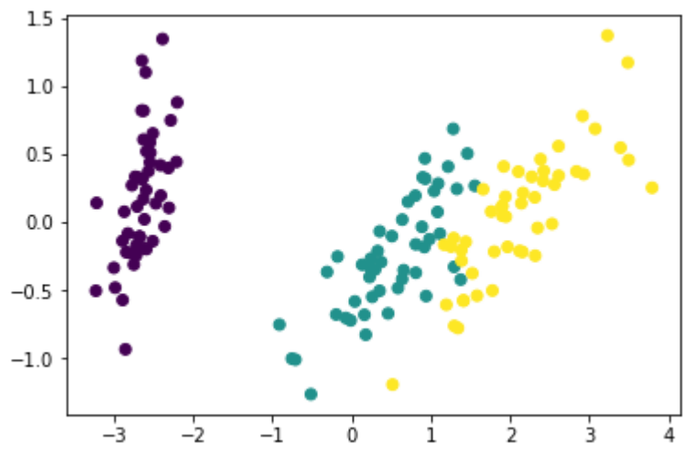
Dimension reduction

In [25]:
```
dim_r = PCA(n_components=2)
dim_r.fit(iris.data)
transformed = dim_r.transform(iris.data)
```

In [26]:
```
print(transformed.shape)
```

```
(150, 2)
```

In [27]:
```
plt.scatter(transformed[:,0], transformed[:,1], c=iris.target)
plt.show()
```



Word Frequency Array

کلمات / متون	یادگیری	ماشین	طبقه بندی	خوشه بندی	...
متن اول	۰	۰.۱۵	۰.۰۵	۰	
متن دوم	۰	۰	۰	۰	
متن سوم	۰.۲۱	۰.۱۷	۰.۱۱	۰.۱	
...					

http://www.tfidf.com (http://www.tfidf.com)

In [28]:
```
import pandas as pd
from scipy.sparse import csr_matrix
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.decomposition import TruncatedSVD
from sklearn.cluster import KMeans
from sklearn.pipeline import make_pipeline
```

In [29]:
```
doc = ['you are watching machine learning course',
       'word frequency array is a part of unsupervised learning from machine learning c
ourse',
       'Faradars is an online educational website']

titles = ['first doc', 'second doc', 'third doc']
```

In [30]:
```
tfidf = TfidfVectorizer()
csr_mat = tfidf.fit_transform(doc)
words = tfidf.get_feature_names()
```

In [31]:
```
tfidf.vocabulary_
```

Out[31]:
```
{'an': 0,
 'are': 1,
 'array': 2,
 'course': 3,
 'educational': 4,
 'faradars': 5,
 'frequency': 6,
 'from': 7,
 'is': 8,
 'learning': 9,
 'machine': 10,
 'of': 11,
 'online': 12,
 'part': 13,
 'unsupervised': 14,
 'watching': 15,
 'website': 16,
 'word': 17,
 'you': 18}
```

In [32]:
```
csr_mat.toarray()
```

Out[32]:
```
array([[ 0.        ,  0.45954803,  0.        ,  0.34949812,  0.        ,
         0.        ,  0.        ,  0.        ,  0.        ,  0.34949812,
         0.34949812,  0.        ,  0.        ,  0.        ,  0.        ,
         0.45954803,  0.        ,  0.        ,  0.45954803],
       [ 0.        ,  0.        ,  0.30084481,  0.22880023,  0.        ,
         0.        ,  0.30084481,  0.30084481,  0.22880023,  0.45760046,
         0.22880023,  0.30084481,  0.        ,  0.30084481,  0.30084481,
         0.        ,  0.        ,  0.30084481,  0.        ],
       [ 0.42339448,  0.        ,  0.        ,  0.        ,  0.42339448,
         0.42339448,  0.        ,  0.        ,  0.32200242,  0.        ,
         0.        ,  0.        ,  0.42339448,  0.        ,  0.        ,
         0.        ,  0.42339448,  0.        ,  0.        ]])
```

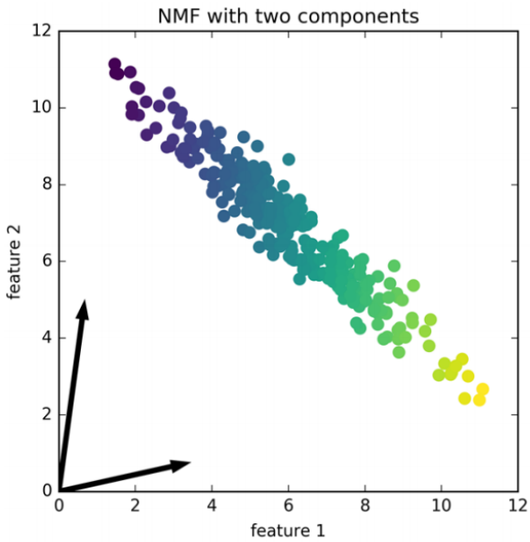
In [33]:
```
svd = TruncatedSVD(n_components=2)
kmeans = KMeans(n_clusters=2)
pipeline = make_pipeline(svd, kmeans)


pipeline.fit(csr_mat)
labels = pipeline.predict(csr_mat)
```

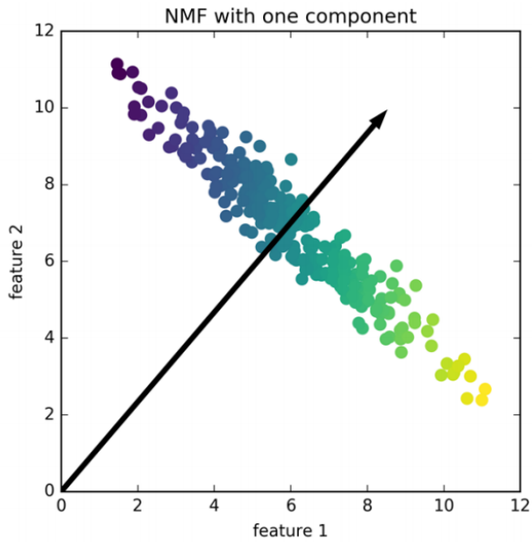
In [34]:
```
df = pd.DataFrame({'label': labels, 'doc': titles})
print(df.sort_values('label'))
```

```
          doc  label
0   first doc      0
1  second doc      0
2   third doc      1
```

NMF

> Non-negative matrix factorization





In [35]:
```
from sklearn.decomposition import NMF
```

In [36]:
```
nmf = NMF(n_components=2)
nmf.fit(csr_mat)
nmf_trf = nmf.transform(csr_mat)
```

In [37]:
```
pd.DataFrame(nmf.components_, columns=sorted(tfidf.vocabulary_))
```

Out[37]:

	an	are	array	course	educational	faradars	frequency	from	
0	0.000000	0.308696	0.199136	0.3864	0.000000	0.000000	0.199136	0.199136	0.14
1	0.412331	0.000000	0.007223	0.0000	0.412331	0.412331	0.007223	0.007223	0.31

In [38]:
```
print(nmf_trf)
```

```
[[ 0.75440592  0.        ]
 [ 0.74426702  0.04978169]
 [ 0.          1.02441145]]
```

In [39]:
```
nmf_df = pd.DataFrame(nmf_trf, index=titles)
nmf_df
```

Out[39]:

	0	1
first doc	0.754406	0.000000
second doc	0.744267	0.049782
third doc	0.000000	1.024411

Example 2

In [40]:
```
mat = [[1, 2, 3],
       [4, 5, 6],
       [7, 8, 9]]
```

In [41]:
```
ex2_nmf = NMF(n_components=2)
ex2_nmf.fit(mat)
nmf_feature = ex2_nmf.transform(mat)
nmf_feature
```

Out[41]:
```
array([[ 0.49770353,  2.01555509],
       [ 1.9886994 ,  1.00775116],
       [ 3.47968482,  0.        ]])
```

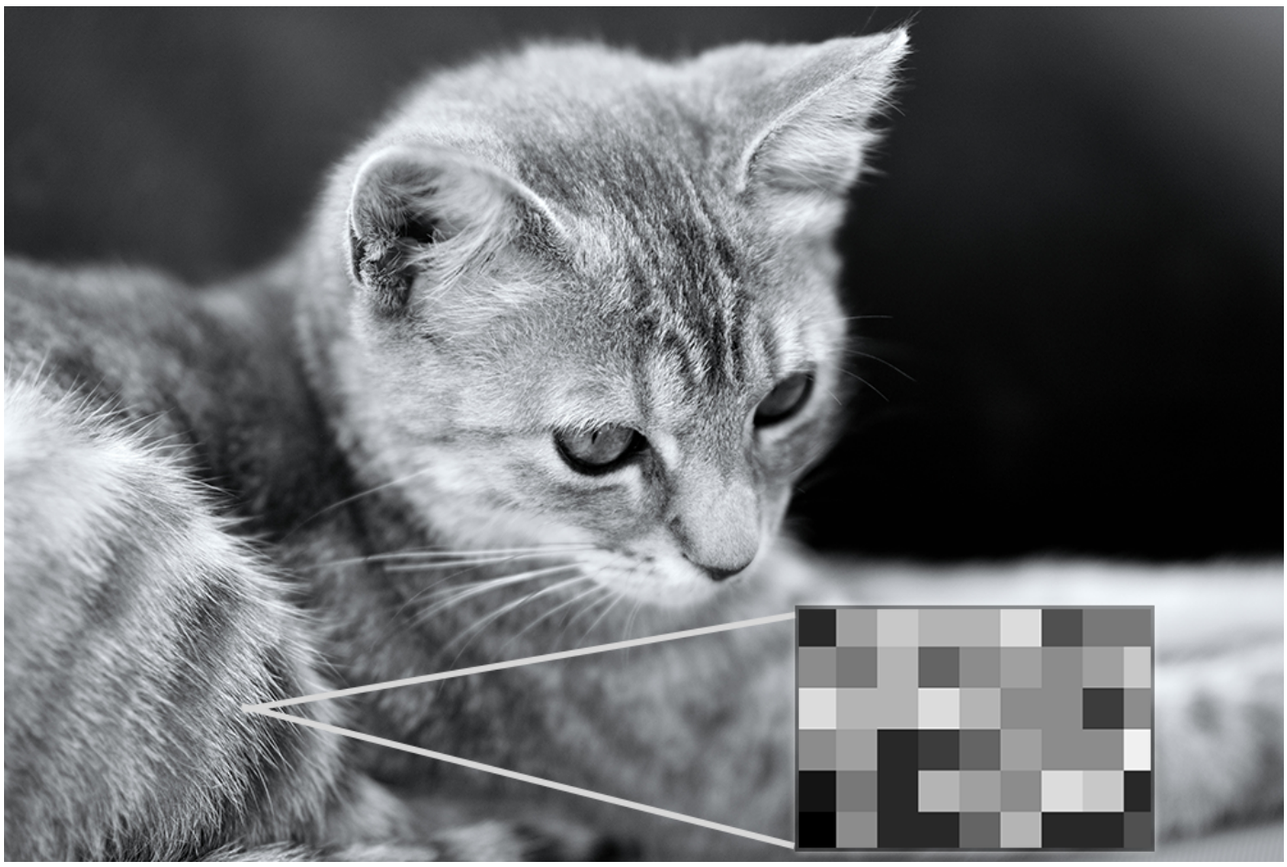
In [42]:
```
ex2_nmf.components_
```

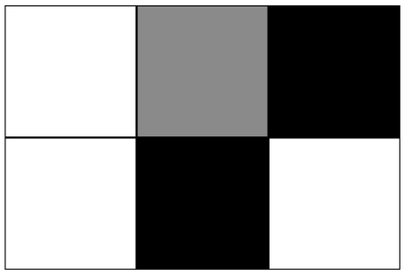
Out[42]:
```
array([[ 2.01166962,  2.29905988,  2.58644549],
       [ 0.        ,  0.42480675,  0.84963252]])
```

In [43]:
```
import numpy as np
np.dot(nmf_feature, ex2_nmf.components_)
```

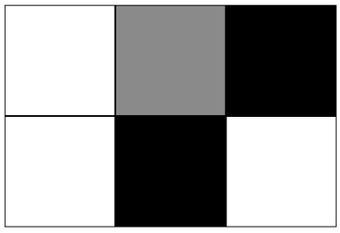
Out[43]:
```
array([[ 1.00121507,  2.00047162,  2.99976419],
       [ 4.00060618,  5.0002385 ,  5.99988075],
       [ 6.99997626,  8.00000376,  9.00001512]])
```

Grayscale

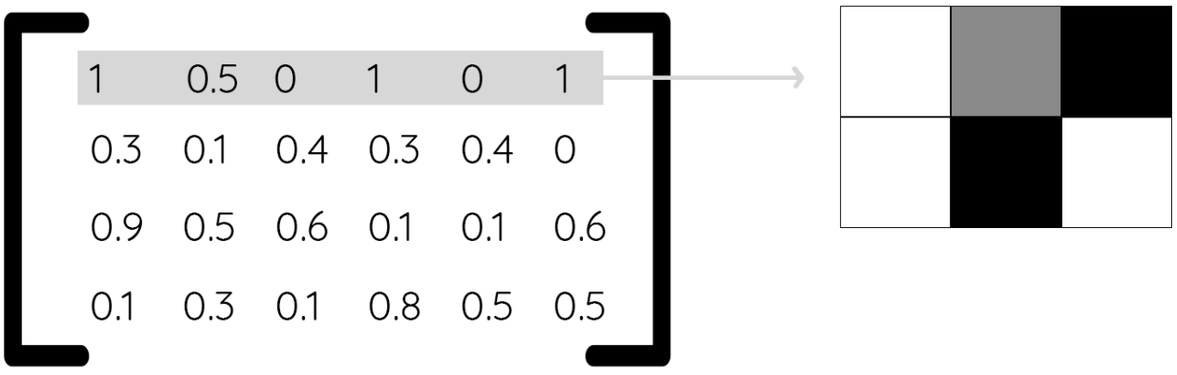




$$\begin{bmatrix} 1 & 0.5 & 0 \\ 1 & 0 & 1 \end{bmatrix}$$



$$\begin{bmatrix} 1 & 0.5 & 0 & 1 & 0 & 1 \end{bmatrix}$$

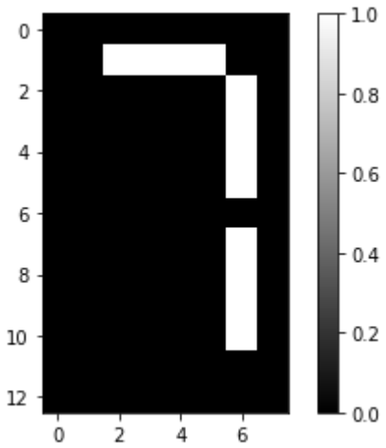


In [44]:
```
import numpy as np
from matplotlib import pyplot as plt
import csv
```

In [45]:
```
data = (open('c://led-digits.csv'))
samples = [[float(x) for x in rec] for rec in csv.reader(data,delimiter=',')]
digit = np.array(samples[0])
bitmap = digit.reshape((13, 8))
bitmap
```

Out[45]:
```
array([[ 0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.],
       [ 0.,  0.,  1.,  1.,  1.,  1.,  0.,  0.],
       [ 0.,  0.,  0.,  0.,  0.,  0.,  1.,  0.],
       [ 0.,  0.,  0.,  0.,  0.,  0.,  1.,  0.],
       [ 0.,  0.,  0.,  0.,  0.,  0.,  1.,  0.],
       [ 0.,  0.,  0.,  0.,  0.,  0.,  1.,  0.],
       [ 0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.],
       [ 0.,  0.,  0.,  0.,  0.,  0.,  1.,  0.],
       [ 0.,  0.,  0.,  0.,  0.,  0.,  1.,  0.],
       [ 0.,  0.,  0.,  0.,  0.,  0.,  1.,  0.],
       [ 0.,  0.,  0.,  0.,  0.,  0.,  1.,  0.],
       [ 0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.],
       [ 0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.]])
```

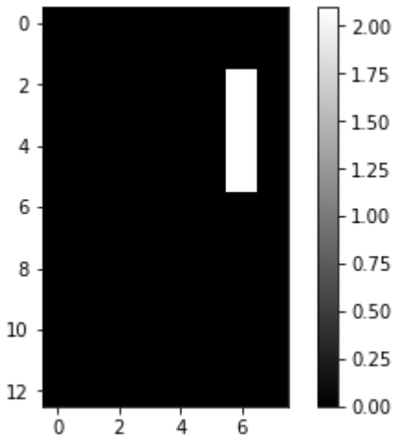
In [46]:
```
plt.imshow(bitmap, cmap='gray')
plt.colorbar()
plt.show()
```

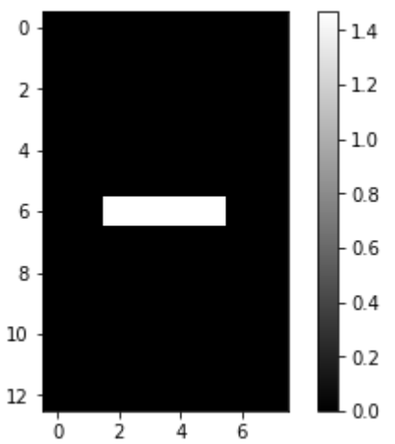


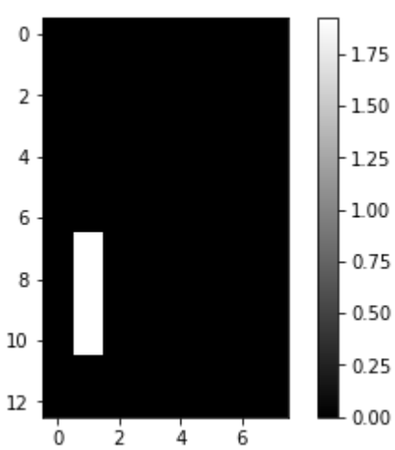
In [47]:
```
model = NMF(n_components=7)
features = model.fit_transform(samples)
```

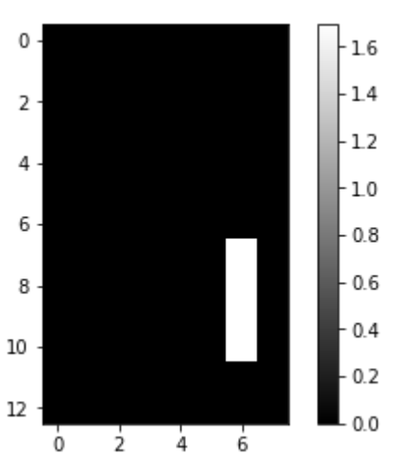
In [48]:
```
def show_as_image(sam):
    bitmap = sam.reshape((13, 8))
    plt.figure()
    plt.imshow(bitmap, cmap='gray')
    plt.colorbar()
    plt.show()
```

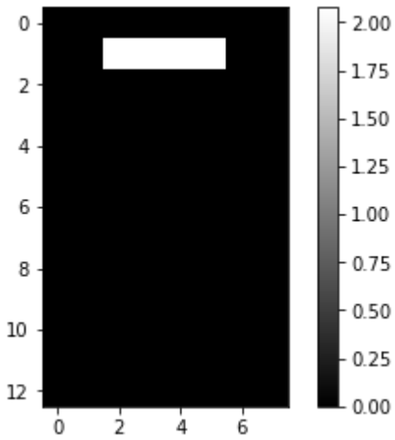
In [49]:
```
[show_as_image(component) for component in model.components_]
```

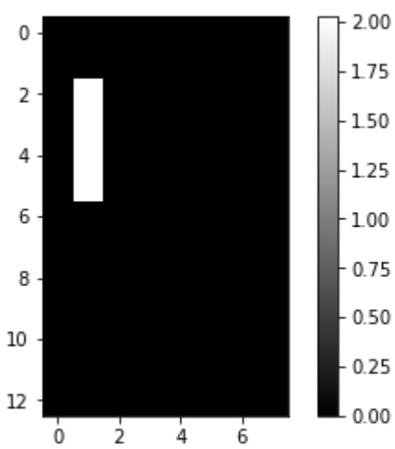

0
2
4
6
8
10
12
0
2
4
6
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00

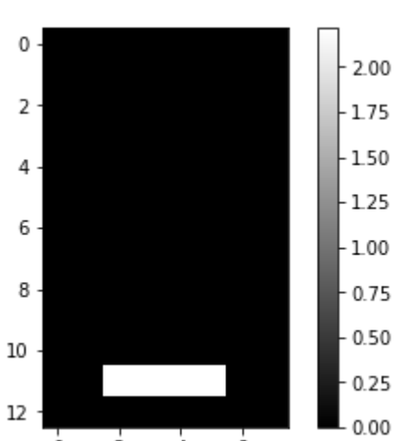

0
2
4
6
8
10
12
0
2
4
6
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0


0
2
4
6
8
10
12
0
2
4
6
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00


0
2
4
6
8
10
12
0
2
4
6
1.6
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0





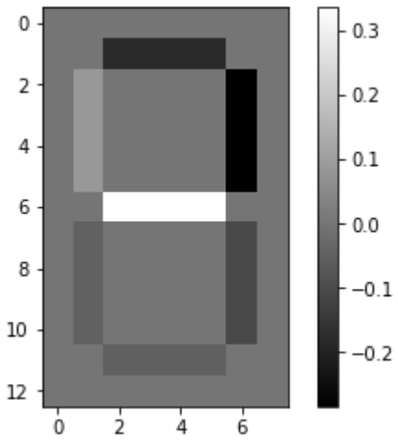


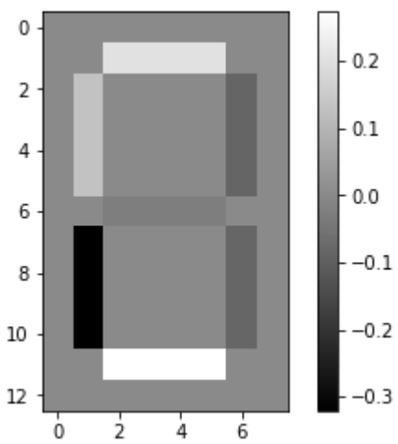
Out[49]: [None, None, None, None, None, None, None]

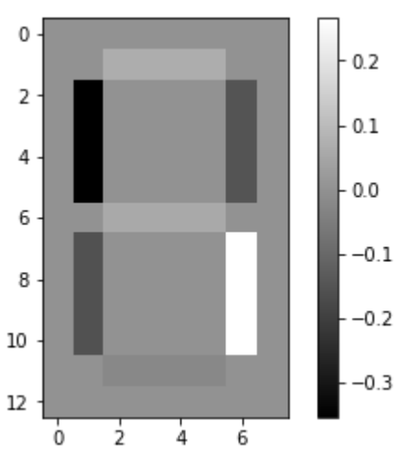
In [50]:
```
digit_features = features[0,:]
digit_features
```

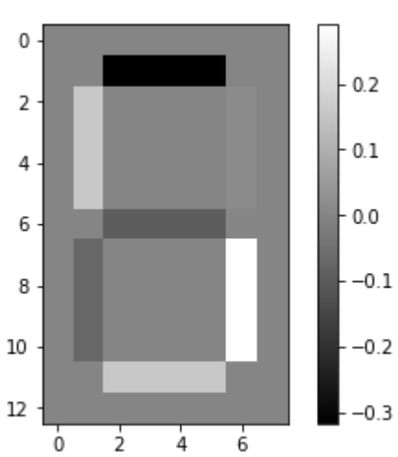
Out[50]:
```
array([  4.76823559e-01,   0.00000000e+00,   0.00000000e+00,
         5.90605054e-01,   4.81559442e-01,   0.00000000e+00,
         7.37551667e-16])
```

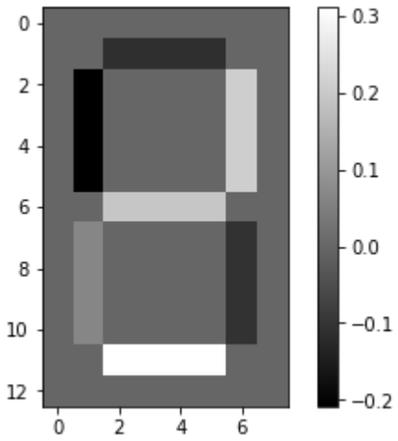
In [51]:
```
model_pca = PCA(n_components=7)
pca_features = model_pca.fit_transform(samples)

[show_as_image(component) for component in model_pca.components_]
```

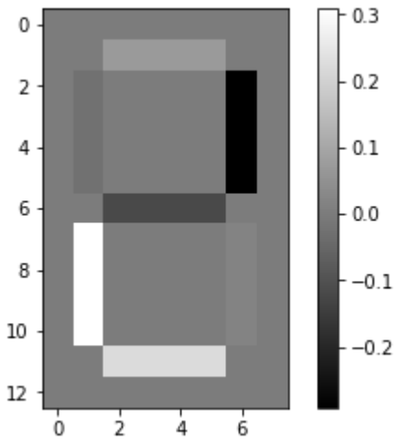

0
2
4
6
8
10
12
0
2
4
6
0.3
0.2
0.1
0.0
−0.1
−0.2

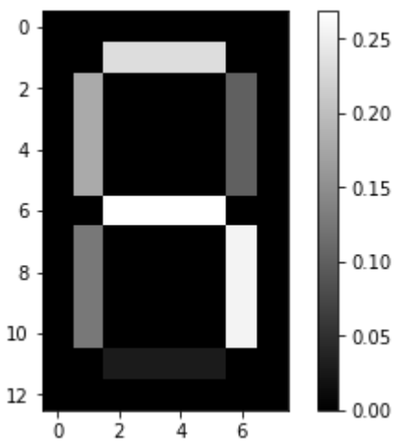

0
2
4
6
8
10
12
0
2
4
6
0.2
0.1
0.0
−0.1
−0.2
−0.3


0
2
4
6
8
10
12
0
2
4
6
0.2
0.1
0.0
−0.1
−0.2
−0.3


0
2
4
6
8
10
12
0
2
4
6
0.2
0.1
0.0
−0.1
−0.2
−0.3







Out[51]: [None, None, None, None, None, None, None]

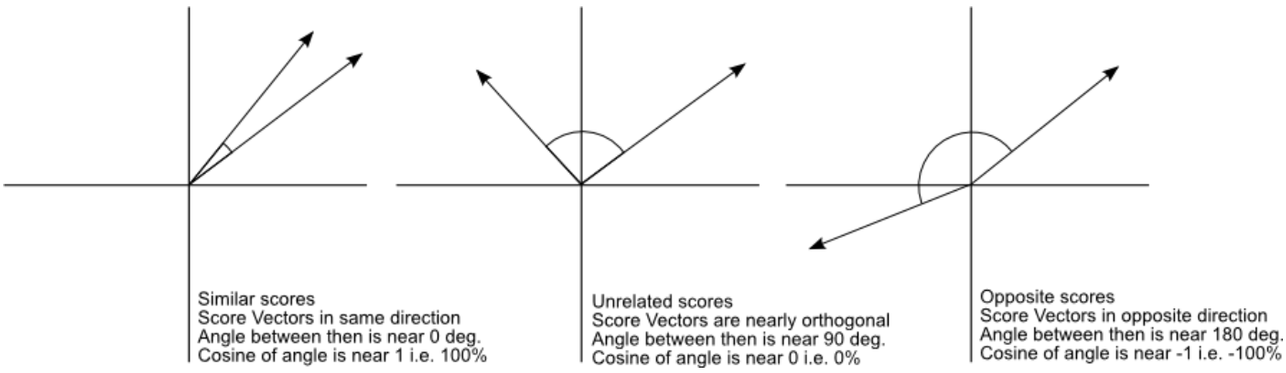
Recommender system

In [52]:
```
docs = ['this video is about machine learning',
       'faradars is an educational website',
       'kmeans is a way to clustering that is a unsupervised learning method']

titles = ['first doc', 'second doc', 'third doc']
```

In [53]:
```
tfidf = TfidfVectorizer()
csr_mat = tfidf.fit_transform(doc)
words = tfidf.vocabulary_
```

In [54]:
```
model = NMF(n_components=2)
model.fit(csr_mat)
features = model.transform(csr_mat)
```

Cosine Similarity



$$\vec{a} \cdot \vec{b} = ||\vec{a}|| \cdot ||\vec{b}|| \cos\theta$$

$$\cos\theta = \frac{\vec{a} \cdot \vec{b}}{||\vec{a}|| \cdot ||\vec{b}||}$$

In [55]:
```
from sklearn.preprocessing import normalize

norm_features = normalize(features)
df = pd.DataFrame(norm_features, index=titles)
df
```

Out[55]:

	0	1
first doc	1.000000	0.000000
second doc	0.997771	0.066738
third doc	0.000000	1.000000