CS4487 - Machine Learning

Lecture 7 - Linear Dimensionality Reduction

Dr. Antoni B. Chan

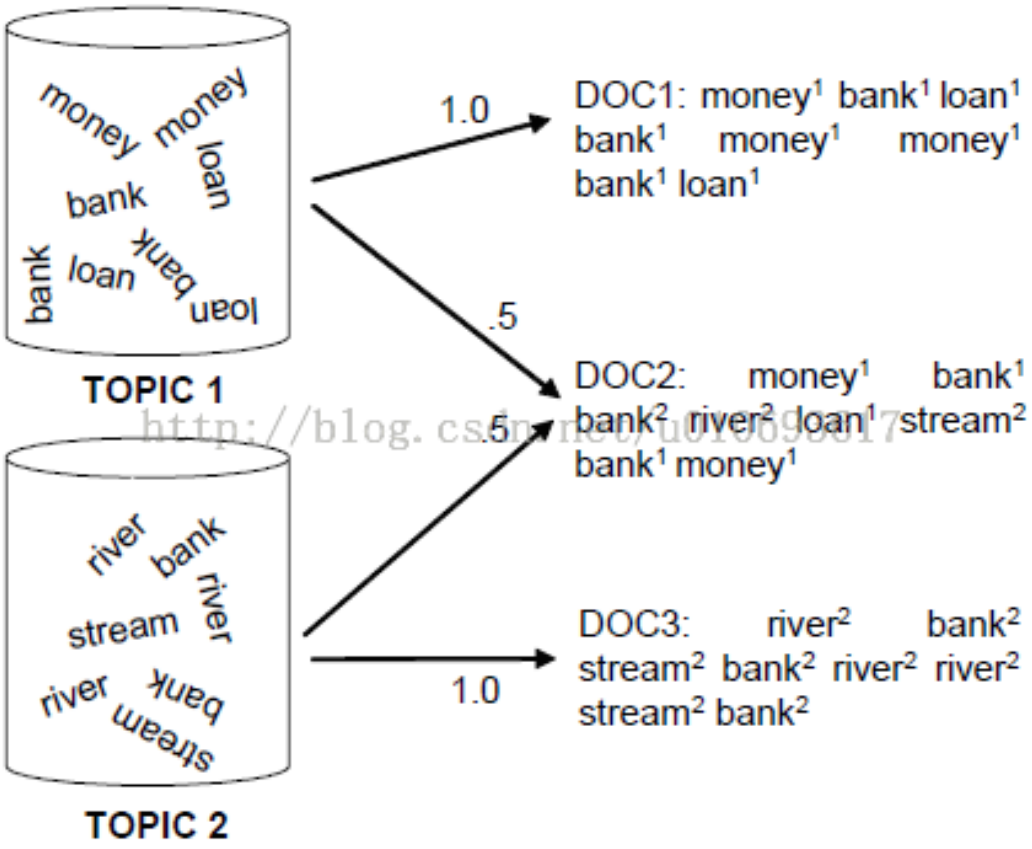
Dept. of Computer Science, City University of Hong Kong

Outline

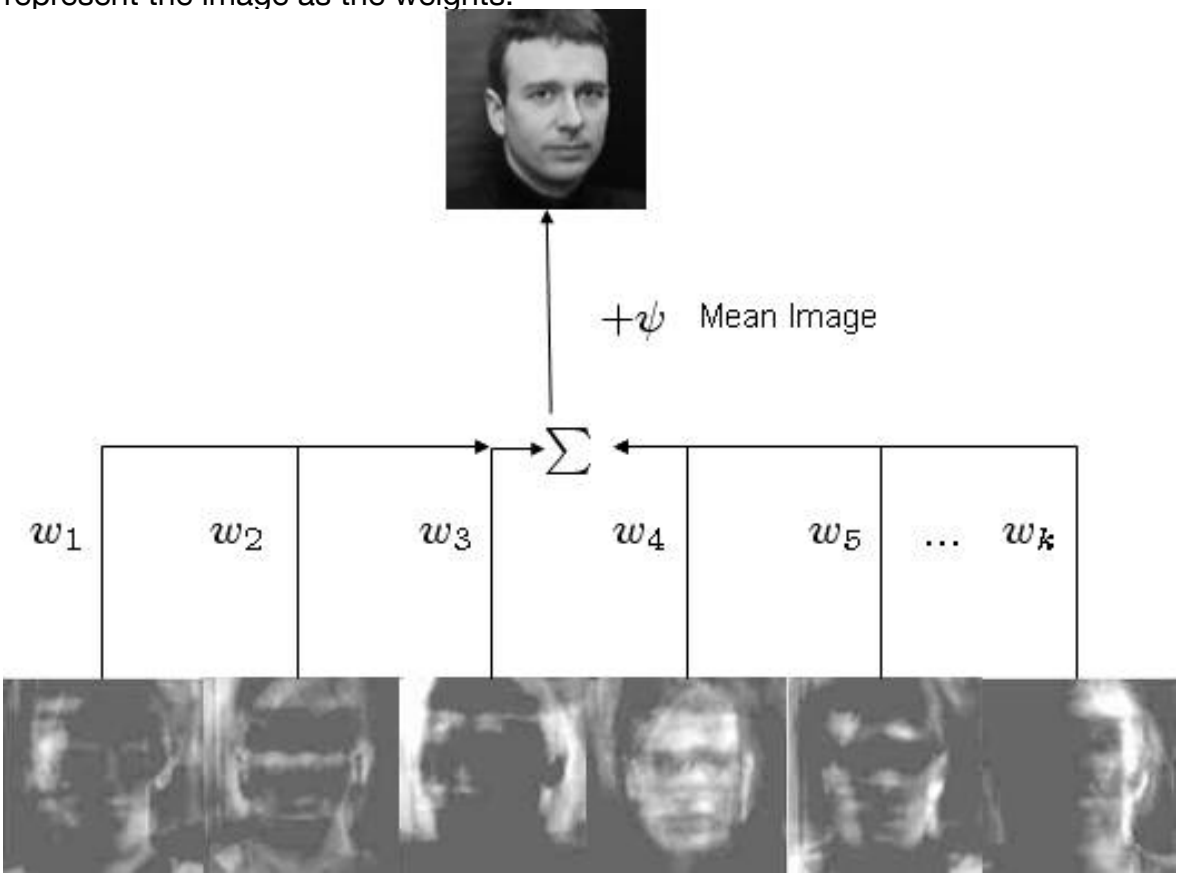
1. Linear Dimensionality Reduction for Vectors
   A. Principal Component Analysis (PCA)
   B. Random Projections
   C. Fisher's Linear Discriminant (FLD)
2. Linear Dimensionality Reduction for Text
   A. Latent Semantic Analysis (LSA)
   B. Non-negative Matrix Factorization (NMF)
   C. Latent Dirichlet Allocation (LDA)

Dimensionality Reduction

- **Goal:** Transform high-dimensional vectors into low-dimensional vectors.
  - Dimensions in the low-dim data represent co-occuring features in high-dim data.
  - Dimensions in the low-dim data may have semantic meaning.
- **For example:** document analysis
  - high-dim: bag-of-word vectors of documents
  - low-dim: each dimension represents similarity to a topic.



- **Example:** image analysis
  - approximate an image as a weighted combination of several basis images
  - represent the image as the weights.

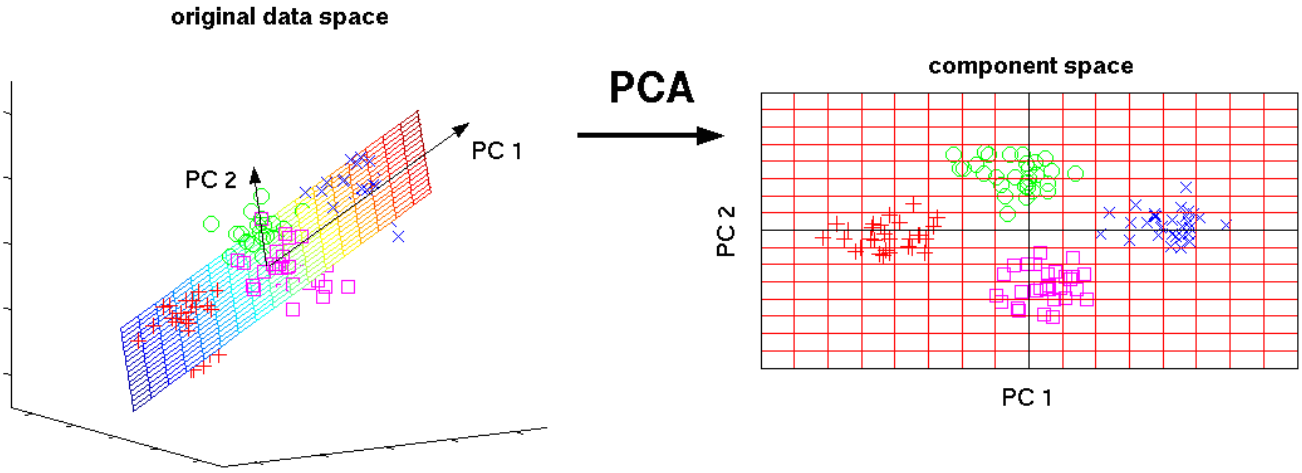


Reasons for Dimensionality Reduction

- Preprocessing - make the dataset easier to use
- Reduce computational cost of running machine learning algorithms
- Remove noise
- Make the results easier to understand (visualization)

Linear Dimensionality Reduction

- Project the original data onto a lower-dimensional hyperplane (e.g., line, plane).
  - I.e, Move and rotate the coordinate axis of the data
- Represent the data with coordinates in the new component space.



- Equivalently, approximate the data point $\mathbf{x}$ as a linear combination of basis vectors (components) in the original space.
  - original data point $\mathbf{x} \in \mathbb{R}^d$
  - approximation: $\hat{\mathbf{x}} = \sum_{j=1}^{p} w_j \mathbf{v}_j$
    - $\mathbf{v}_j \in \mathbb{R}^d$ is a basis vector and $w_j \in \mathbb{R}$ the corresponding weight.
  - the data point $\mathbf{x}$ is then represented its corresponding weights
    - $\mathbf{w} = [w_1, \cdots, w_P] \in \mathbb{R}^p$

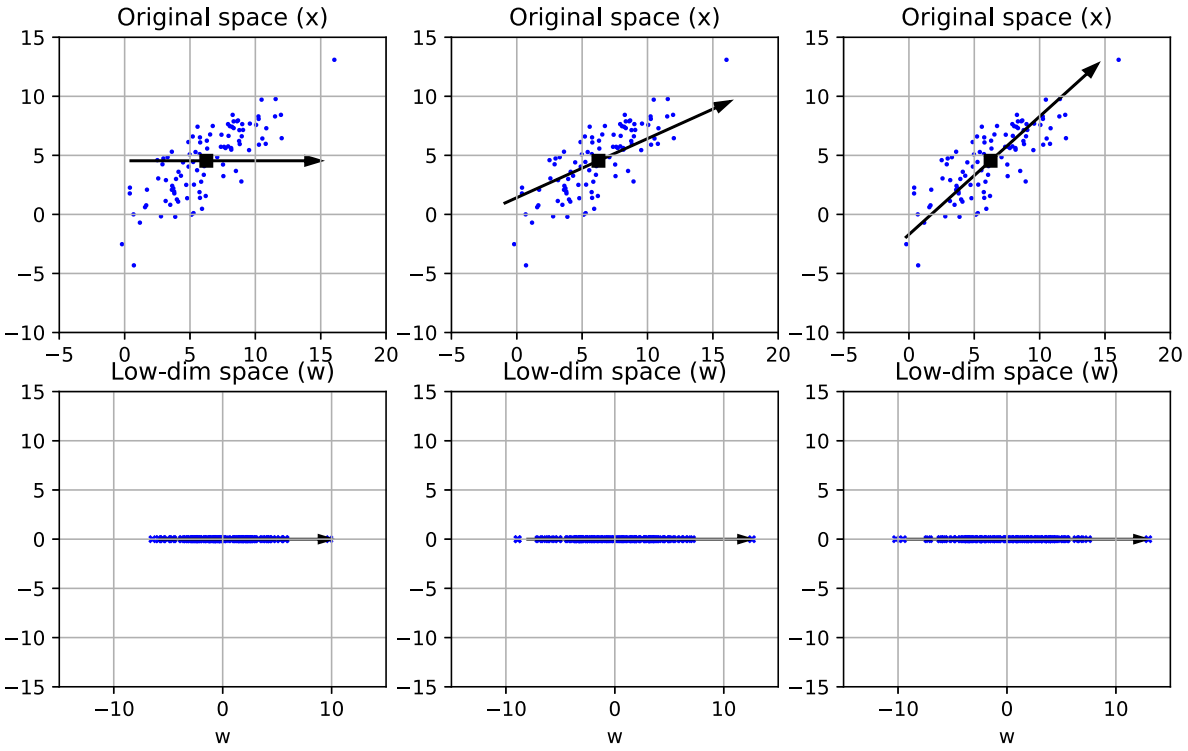
- Several methods for linear dimensionality reduction.
- **Differences:**
  - goal (reconstruction vs classification)
  - unsupervised vs. supervised
  - constraints on the basis vectors and the weights.
  - reconstruction error criteria

Principal Component Analysis (PCA)

- Unsupervised method
- **Goal:** preserve the variance of the data as much as possible
  - choose basis vectors along the maximum variance (longest extent) of the data.
  - the basis vectors are called *principal components* (PC).

In [4]:
```
vfig
```

Out[4]:



- **Goal:** Equivalently, minimize the reconstruction error over all the data points $\{\mathbf{x}_i\}_{i=1}^N$.
  - reconstruction: $\hat{\mathbf{x}_i} = \sum_{j=1}^p w_{i,j}\mathbf{v}_j$

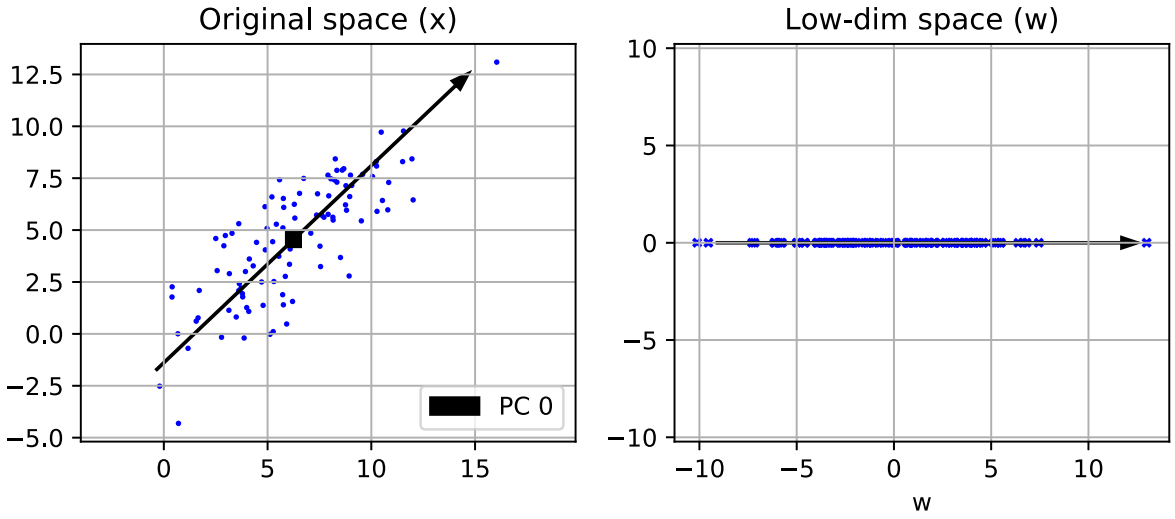
$$\min_{w,\mathbf{v}} \sum_{i=1}^{N} ||\mathbf{x}_i - \hat{\mathbf{x}_i}||^2$$

  - *constraint:* principal components $\mathbf{v}_j$ are orthogonal (perpendicular) to each other.

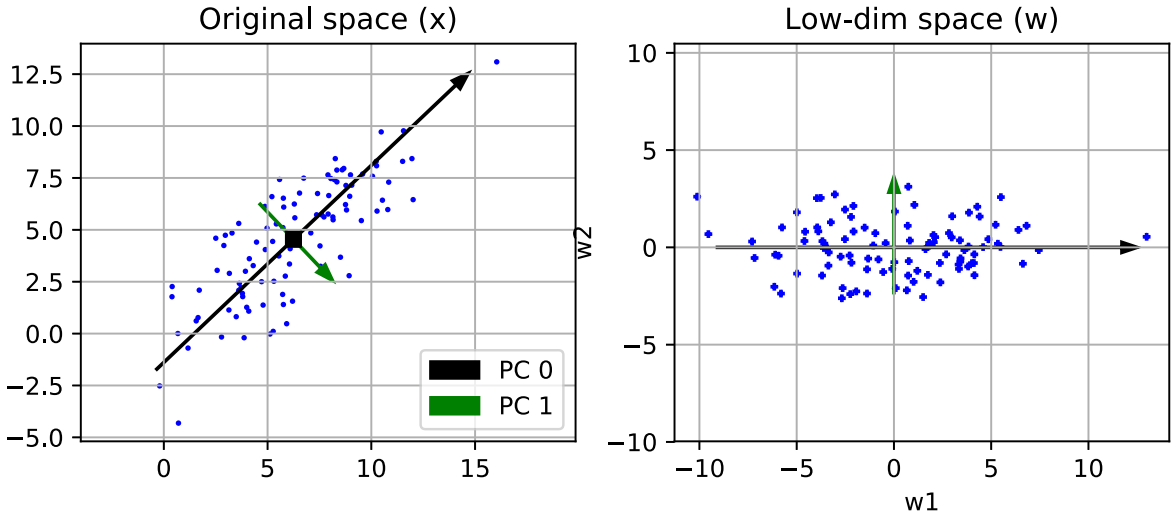
PCA algorithm

- 1) subtract the mean of the data
- 2) the first PC v_1 is the direction that explains the most variance of the data.
- 3) the second PC v_2 is the direction perpendicular to v_1 that explains the most variance.
- 4) the third PC v_3 is the direction perpendicular to $\{v_1, v_2\}$ that explains the most variance.
- 5) ...

In [5]:
```
X = Xblob

# run PCA
pca = decomposition.PCA(n_components=1)
W   = pca.fit_transform(X)  # returns the coefficients

v = pca.components_  # the principal component vector
m = pca.mean_        # the data mean

plt.figure(figsize=(8,3))
plot_basis(X, v);
```



In [6]:
```
# run PCA
pca = decomposition.PCA(n_components=2)
W   = pca.fit_transform(X)

v = pca.components_  # the principal component vector
m = pca.mean_        # the data mean

plt.figure(figsize=(8,3))
plot_basis(X, v);
```

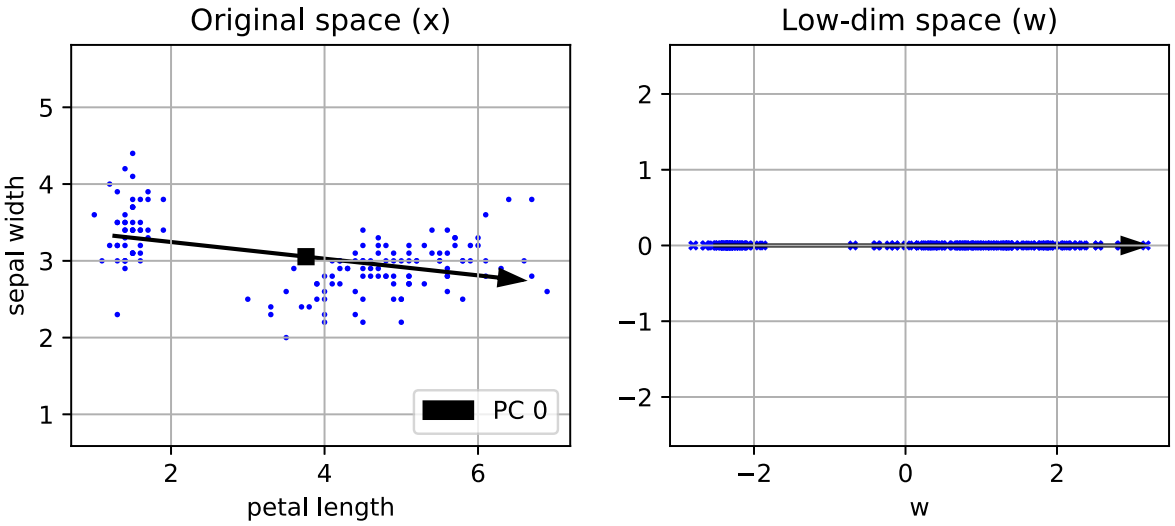


Example on Iris data

- 2D (petal length, sepal width) to 1D

In [8]:
```
ifig
```

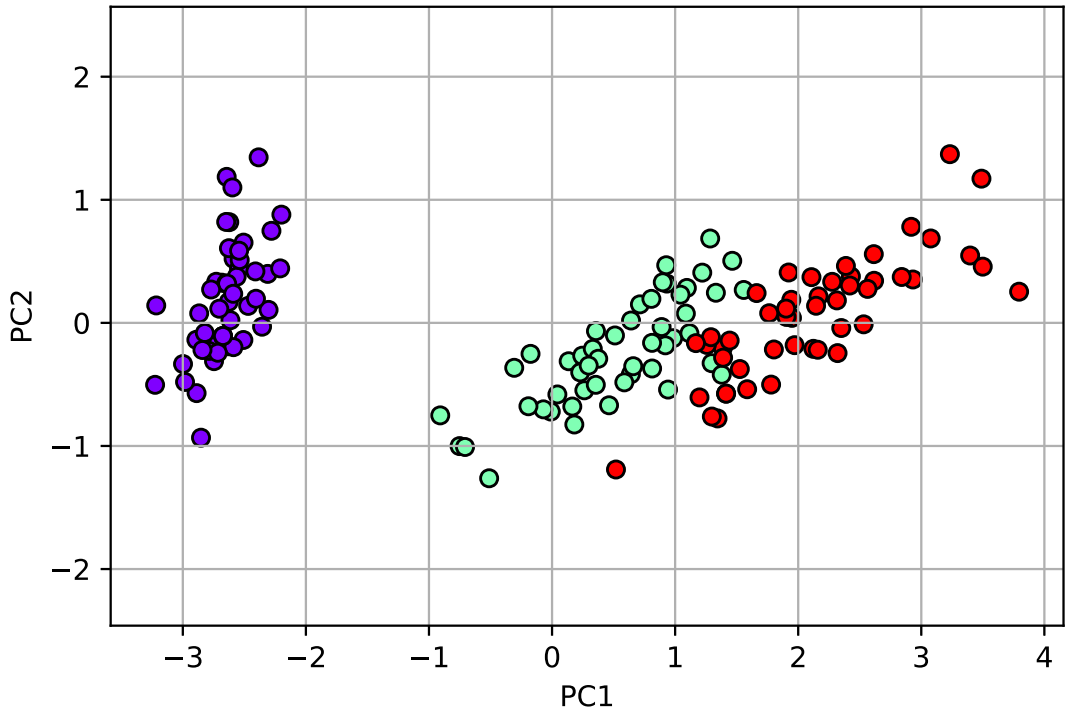
Out[8]:



- 4D to 2D
- mostly preserves the structure of the classes.

In [9]:
```
# get data
iris = datasets.load_iris()
X = iris.data
Y = iris.target

# run PCA
pca = decomposition.PCA(n_components=2)
W = pca.fit_transform(X)

print(iris.feature_names)
print(pca.components_)

plt.figure()
plt.scatter(W[:,0], W[:,1], c=Y, cmap=rbow, edgecolors='k')
plt.axis('equal'); plt.grid(True)
plt.xlabel('PC1'); plt.ylabel('PC2');
```

```
['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal width (c
m)']
[[ 0.36158968 -0.08226889  0.85657211  0.35884393]
 [ 0.65653988  0.72971237 -0.1757674  -0.07470647]]
```



How to choose the number of principal components?

- Two methods to set the number of components p:
  - preserve some percentage of the variance (e.g., 95%).
  - whatever works well for our final task (e.g., classification, regression).

Handwritten digits data

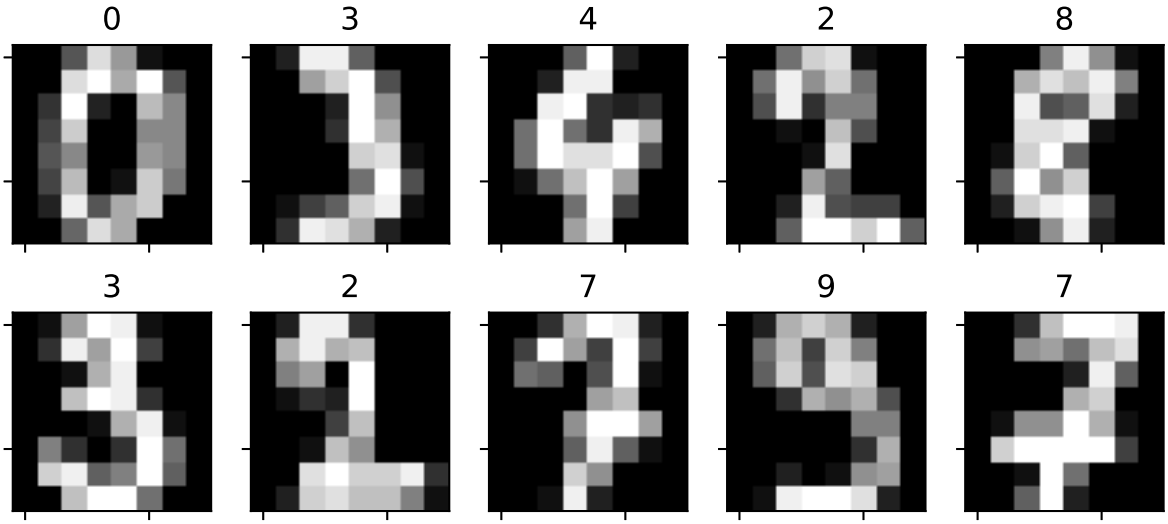
- 1797 images of handwritten digits 0-9
  - each image is 8x8
  - flattened into a 64 dimensional vector

In [10]:
```
# get digit data
digits = datasets.load_digits()
Xdigits = float64(digits.data)
Ydigits = digits.target

print(Xdigits.shape)
```

```
(1797, 64)
```

In [12]:
```
dfig
```

Out[12]:



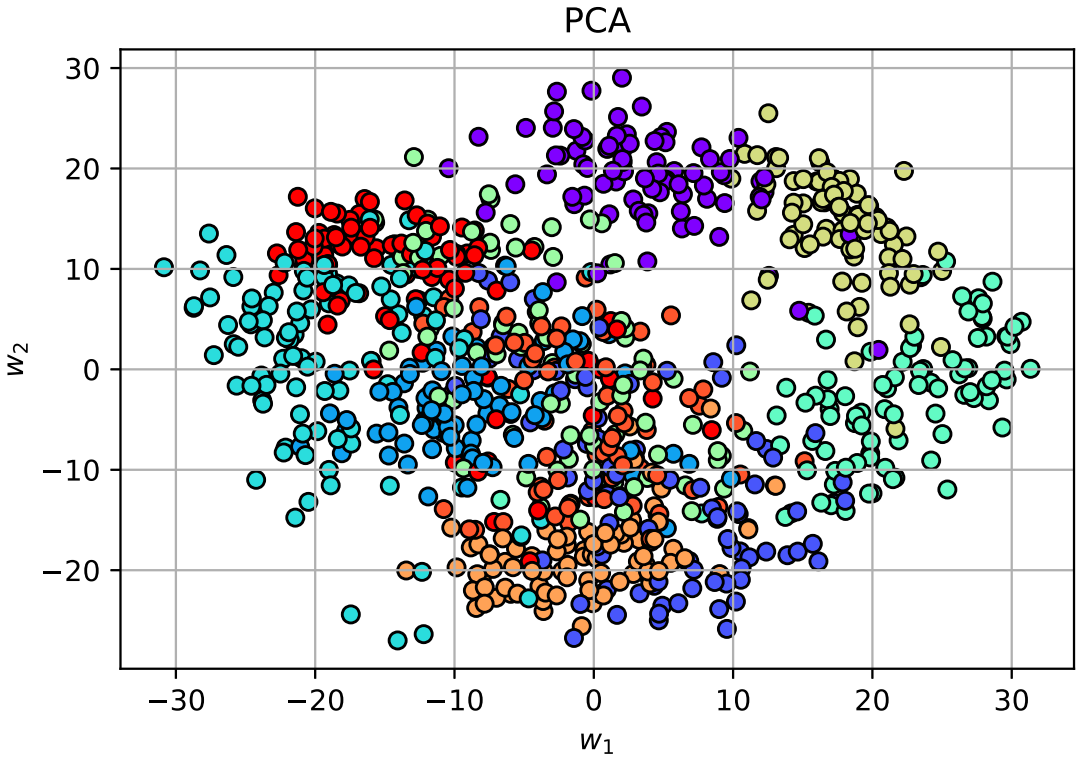
Run PCA on the data

- split data into training and testing sets.
- run PCA on training set, apply to test set
- the top 25 PCs are shown

In [13]:
```
# randomly split data into 80% train and 20% test set
trainX, testX, trainY, testY = \
  model_selection.train_test_split(Xdigits, Ydigits,
  train_size=0.5, test_size=0.5, random_state=4487)
Xdim = Xdigits.shape[1]

# run PCA
pca = decomposition.PCA() # default: n_components=dimension
W  = pca.fit_transform(trainX)  # fit the training set
Wt = pca.transform(testX)  # use the pca model to transform the test set
```

- Visualize the coefficients for the first two PCs.
  - grouping of different digits is sometimes preserved

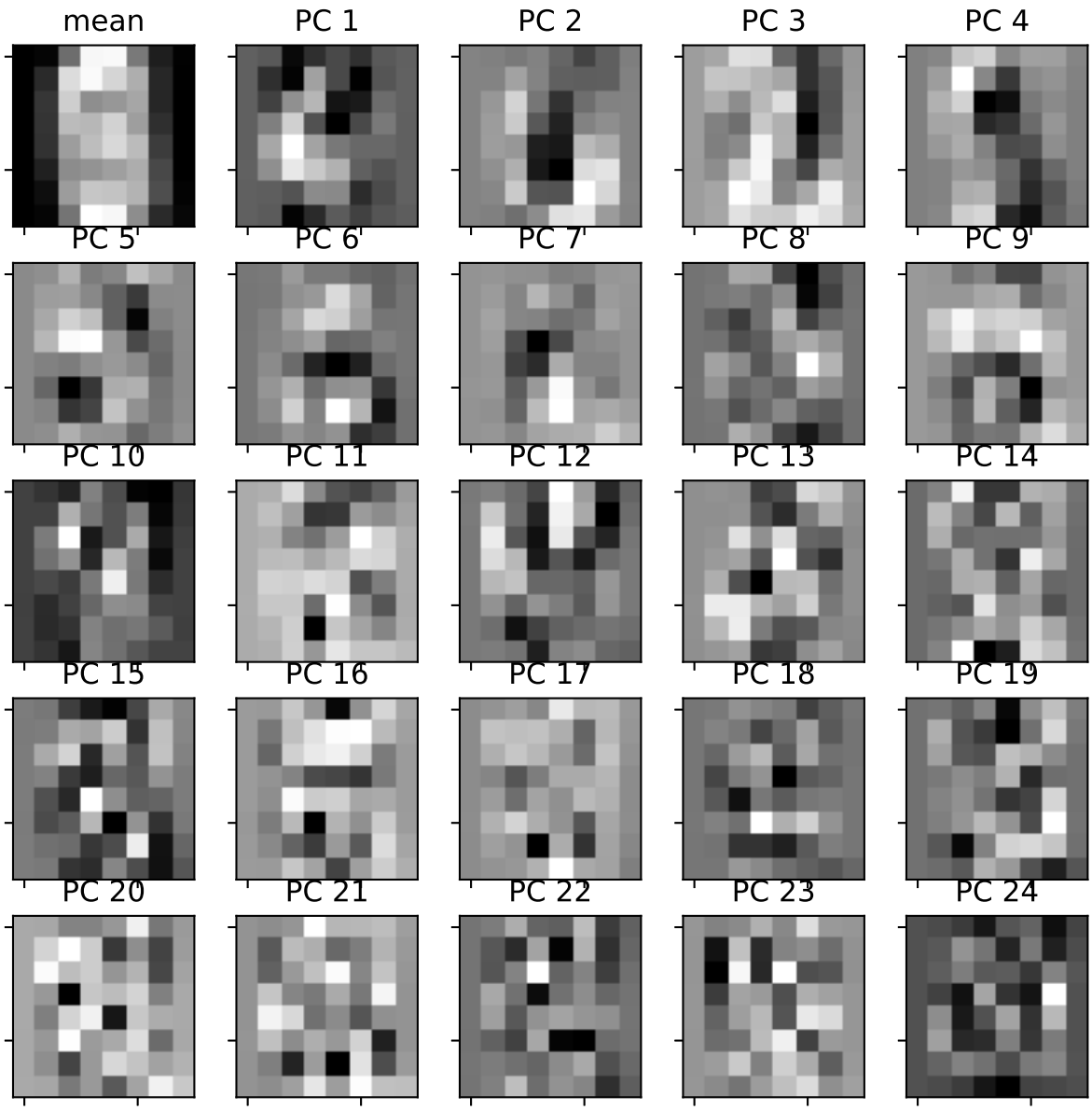
In [14]:
```
plt.figure()
plt.scatter(W[:,0], W[:,1], c=trainY, cmap=rbow, edgecolors='k')
plt.xlabel('$w_1$'); plt.ylabel('$w_2$')
plt.title('PCA'); plt.grid(True);
```



- Look at the mean and principal components

In [16]:
```
pcfig
```

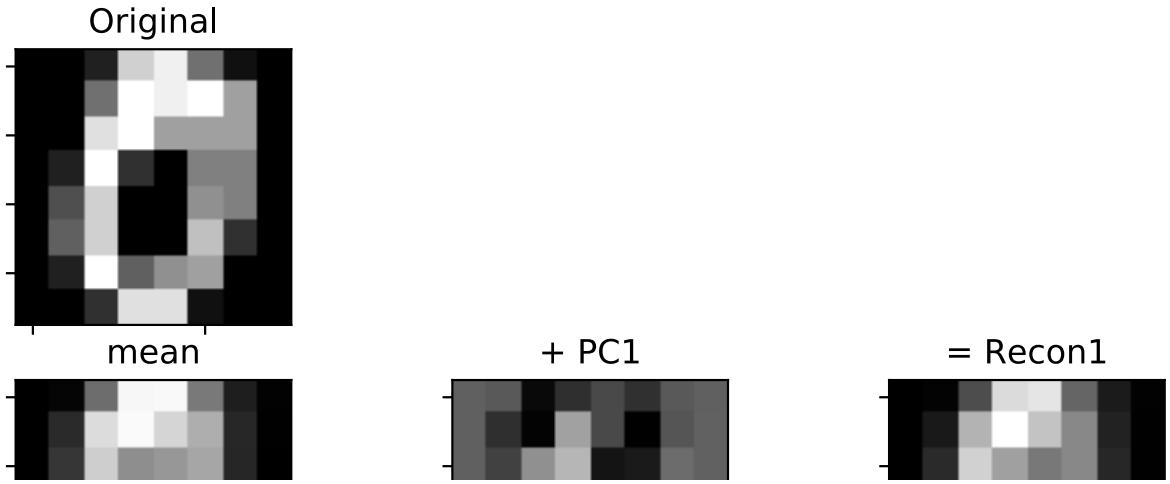
Out[16]:

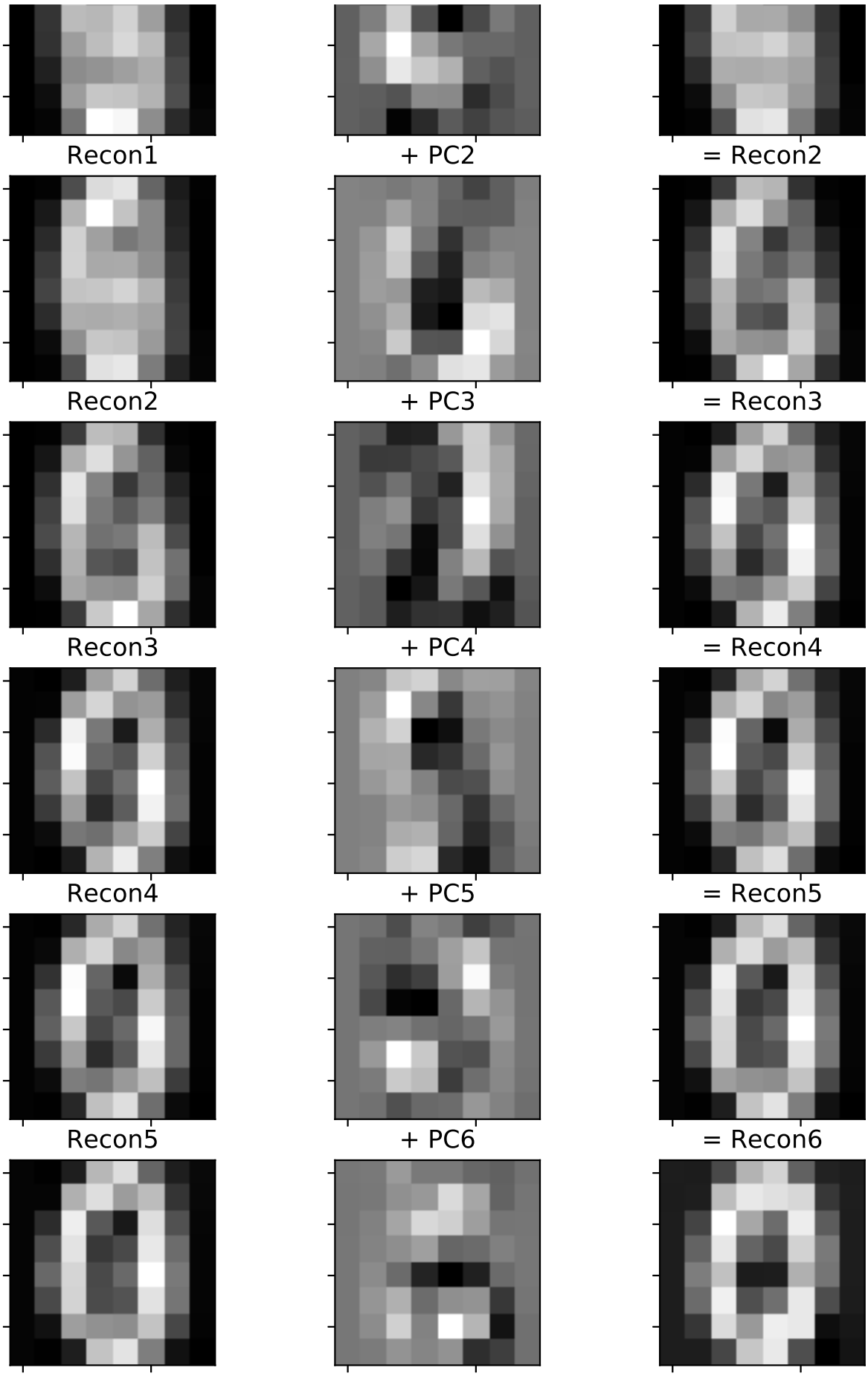


- Reconstruction of a digit image from PC coefficients
  - using more PCs will make the reconstruction better

In [18]:
```
reconfig
```

Out[18]:

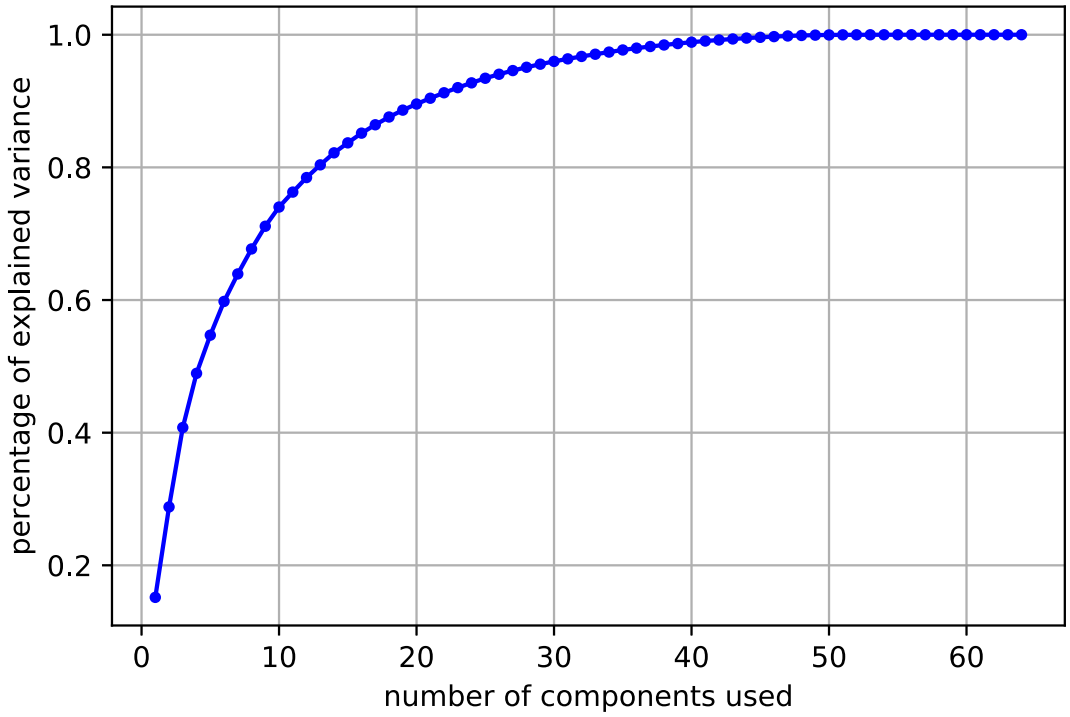



Recon1
+ PC2
= Recon2
Recon2
+ PC3
= Recon3
Recon3
+ PC4
= Recon4
Recon4
+ PC5
= Recon5
Recon5
+ PC6
= Recon6

Explained variance

- each PC explains a percentage of the original data
  - this is called the *explained variance*.
  - PCs are already sorted by explained variance from highest to lowest
- pick the number of PCs to get a certain percentage of explained variance
  - typically 95%

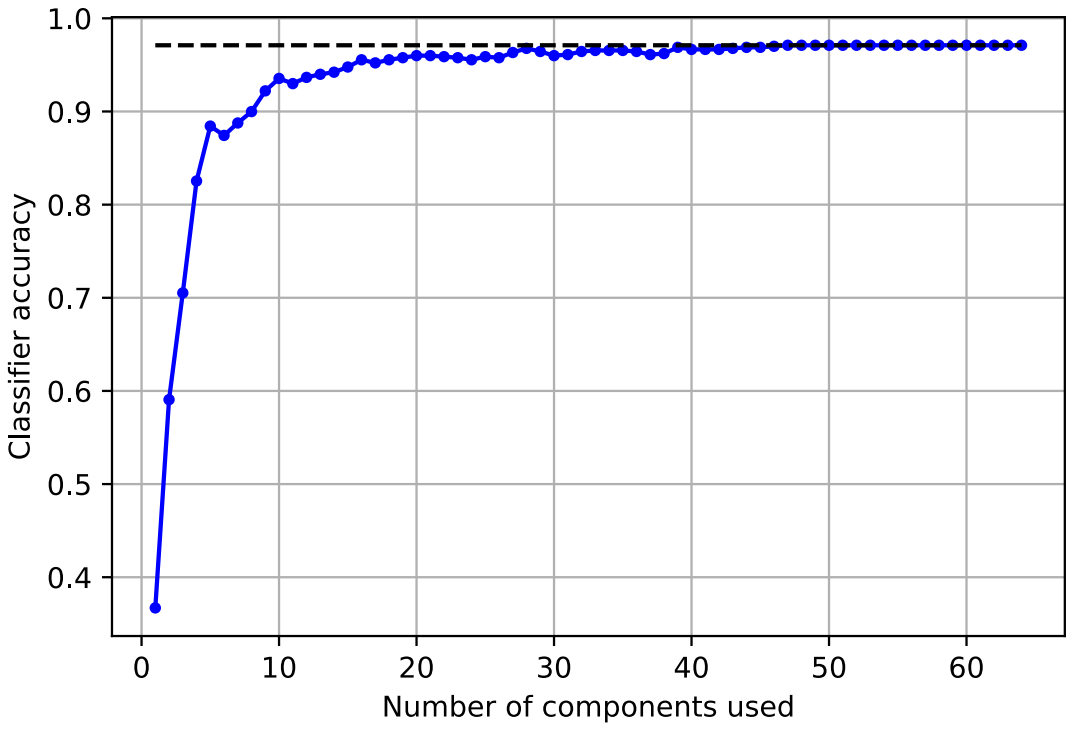
In [19]:
```
ev    = pca.explained_variance_ratio_  # variance explained by each component
cumev = cumsum(ev)                     # cumulative explained variance

plt.plot(range(1,Xdim+1), cumev, 'b.-')
plt.grid(True)
plt.xlabel('number of components used')
plt.ylabel('percentage of explained variance');
```



Task-dependent Selection

- use results on the final task (in this case classification) to select the best number of components
- Note: we don't need to rerun PCA for each number of components
  - just select the subset of PCs based on the number of components desired.

In [20]:
```
acc = zeros(Xdim)
for j in range(Xdim):
    # extract the subset of PC weights [0,j]
    Wnew     = W[:,0:(j+1)]
    Wnewtest = Wt[:,0:(j+1)]

    # train classifier
    clf = svm.SVC(kernel='linear', C=1)
    clf.fit(Wnew, trainY)

    # test classifier
    Ypred = clf.predict(Wnewtest)
    acc[j] = metrics.accuracy_score(testY, Ypred)

# make a plot
plt.plot(range(1,Xdim+1), acc, '.b-')
plt.plot([1,Xdim], [acc.max(), acc.max()], 'k--')
plt.grid(True)
plt.xlabel('Number of components used')
plt.ylabel('Classifier accuracy');
```



- classification accuracy is stable after using 20 PCs.
  - not much loss in performance if using only 20 PCs.

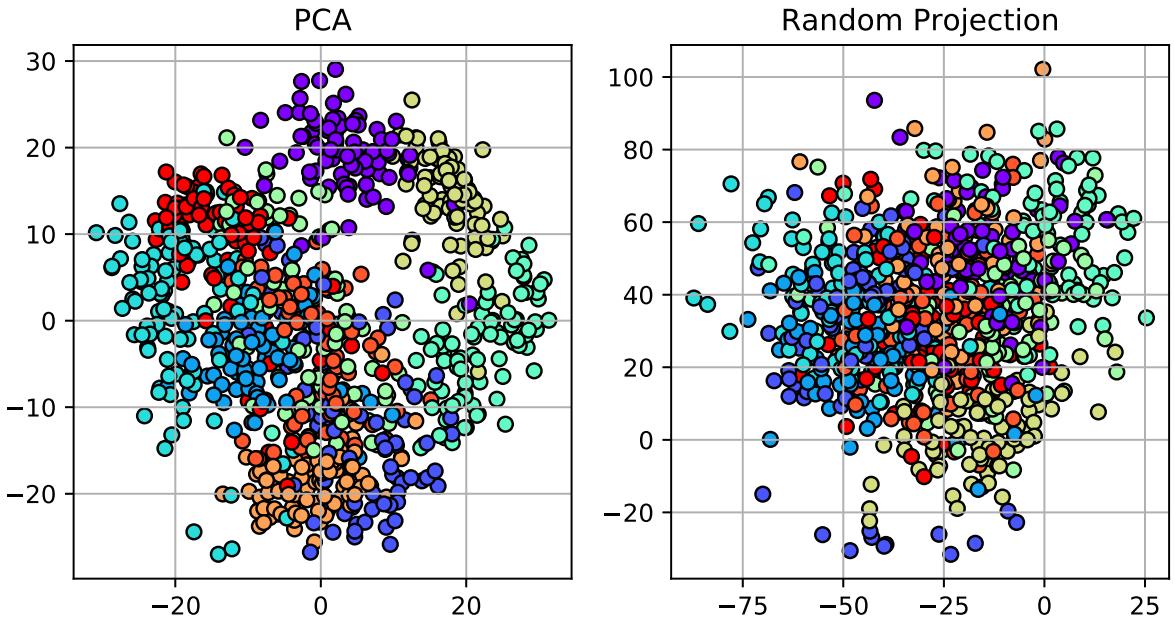
Random Projections

- If the data is very high-dimensional, then it might take too many calculations to do PCA.
  - Complexity: $O(dk^2)$, d is the dimension, k is the number of components
- Do we really need to estimate the principal components to reduce the dimension?

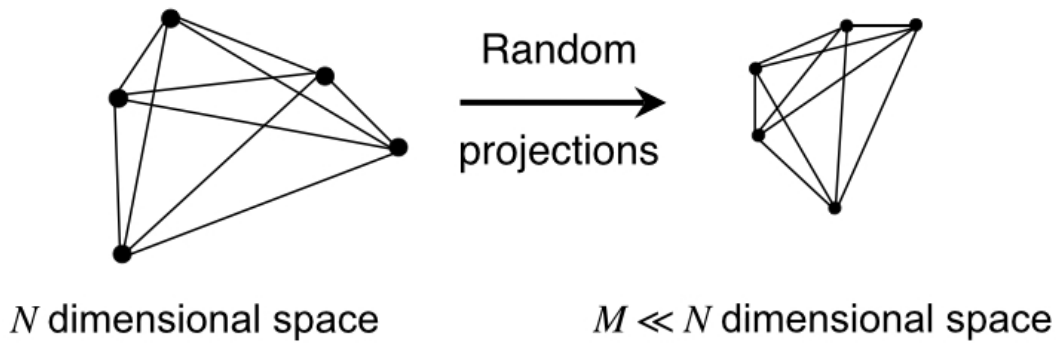
- **Solution:**
  - We can generate random basis vectors and use those.
    - Each entry of $\mathbf{v}_j$ sampled from a Gaussian.
  - This will save a lot of time.
  - Random Projections can reduce computation at the expense of losing some accuracy in the points (adding noise).

In [21]:

```
# project the digits data with Random Projection
rp = random_projection.GaussianRandomProjection(n_components=2, random_state=4487)
Wrp = rp.fit_transform(trainX)

plt.figure(figsize=(8,4))
plt.subplot(1,2,1)
plt.scatter(W[:,0], W[:,1], c=trainY, cmap=rbow, edgecolors='k'); plt.grid(True)
plt.title('PCA')
plt.subplot(1,2,2)
plt.scatter(Wrp[:,0], Wrp[:,1], c=trainY, cmap=rbow, edgecolors='k'); plt.grid(Tru
e)
plt.title('Random Projection');
```



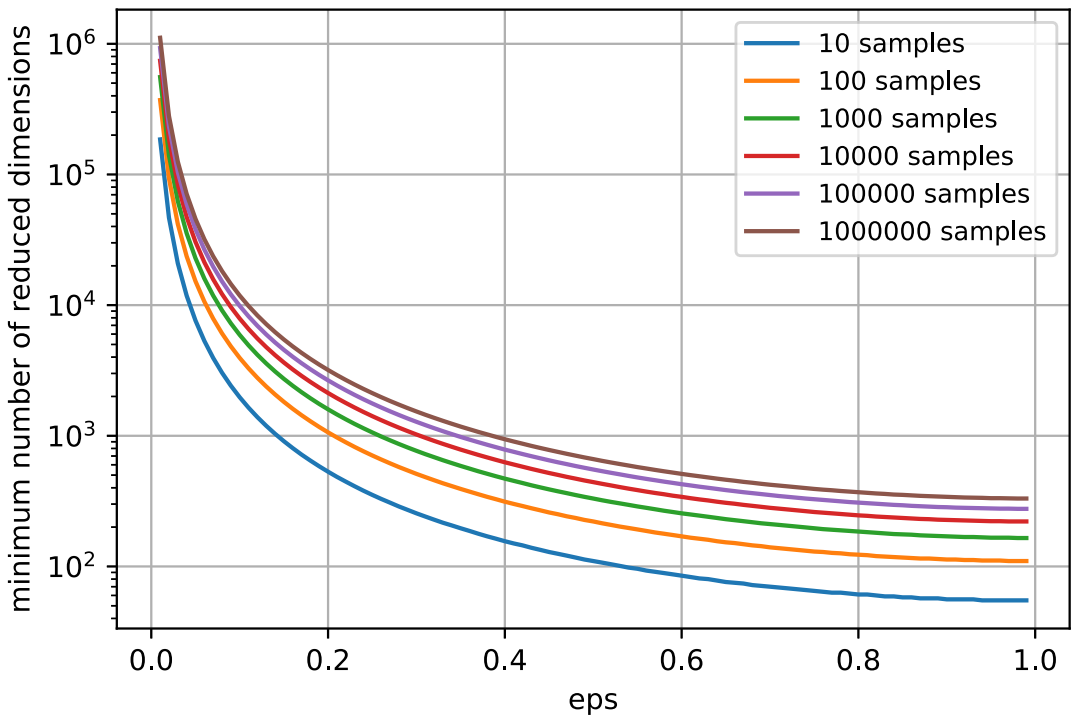
- Okay, but is it good?
  - One way to measure "goodness" is to see if the structure of the data is preserved.
  - In other words, are distances between points preserved in the transformed data?



- **Answer:**
  - Yes!
  - According to the *Johnson-Lindenstrauss lemma*, carefully selecting the distribution of the random projection matrices will preserve the preserve the pairwise distances between any two samples of the dataset, within some error *epsilon*.
    - $(1-\epsilon)\|\mathbf{x}_i - \mathbf{x}_j\|^2 < \|\mathbf{w}_i - \mathbf{w}_j\|^2 < (1+\epsilon)\|\mathbf{x}_i - \mathbf{x}_j\|^2$
    - the minimum reduced dimension p to gaurantee ϵ error depends on the number of samples.
      - (actually, this is fairly conservative)

In [23]:
```
jlfig
```

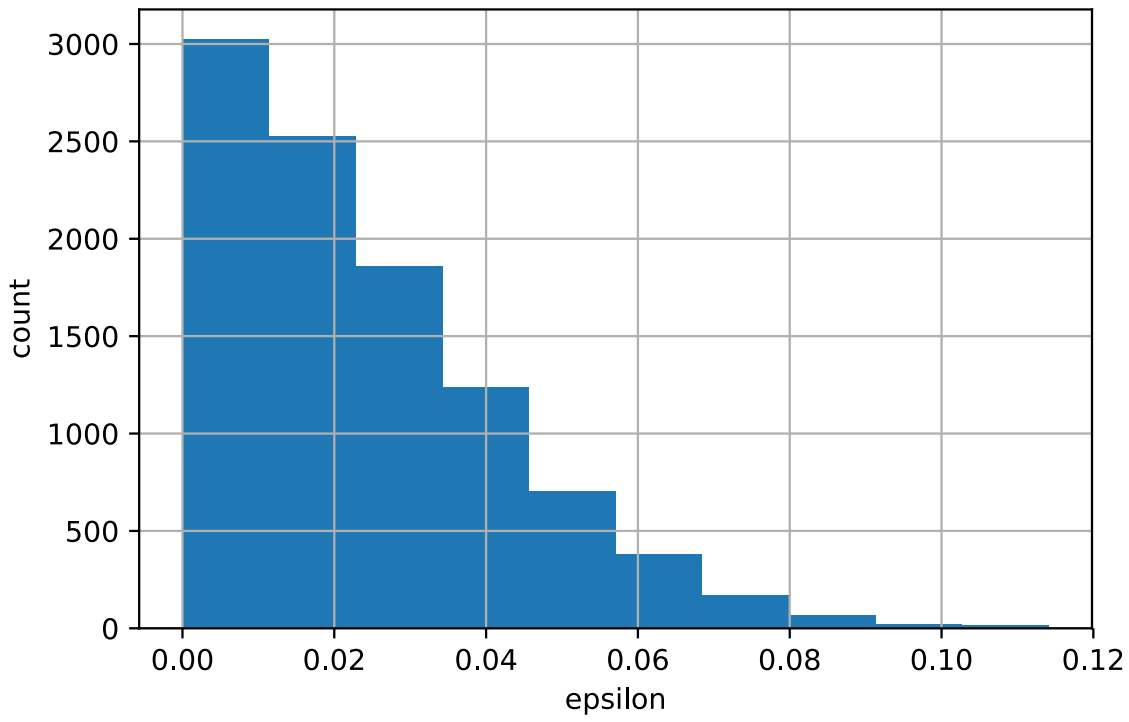
Out[23]:



- Example

In [24]:
```
# generate random data
# (dimension=10000, samples=100)
X = random.rand(100,10000)

# fit to 500 components
rp = random_projection.GaussianRandomProjection(n_components=500, random_state=448
7)
Wrp = rp.fit_transform(X)

# Calculate the pairwise distances
D = metrics.pairwise.euclidean_distances(X, X)
Drp = metrics.pairwise.euclidean_distances(Wrp, Wrp)

# calculate epsilon
epss = abs(Drp/D-1)
epss[isnan(epss)] = 0 # remove 0/0
plt.hist(epss.flatten())
plt.xlabel('epsilon')
plt.ylabel('count')
plt.grid(True)
```

```
/anaconda3/lib/python3.5/site-packages/ipykernel_launcher.py:14: RuntimeWarnin
g: divide by zero encountered in true_divide

/anaconda3/lib/python3.5/site-packages/ipykernel_launcher.py:14: RuntimeWarnin
g: invalid value encountered in true_divide
```

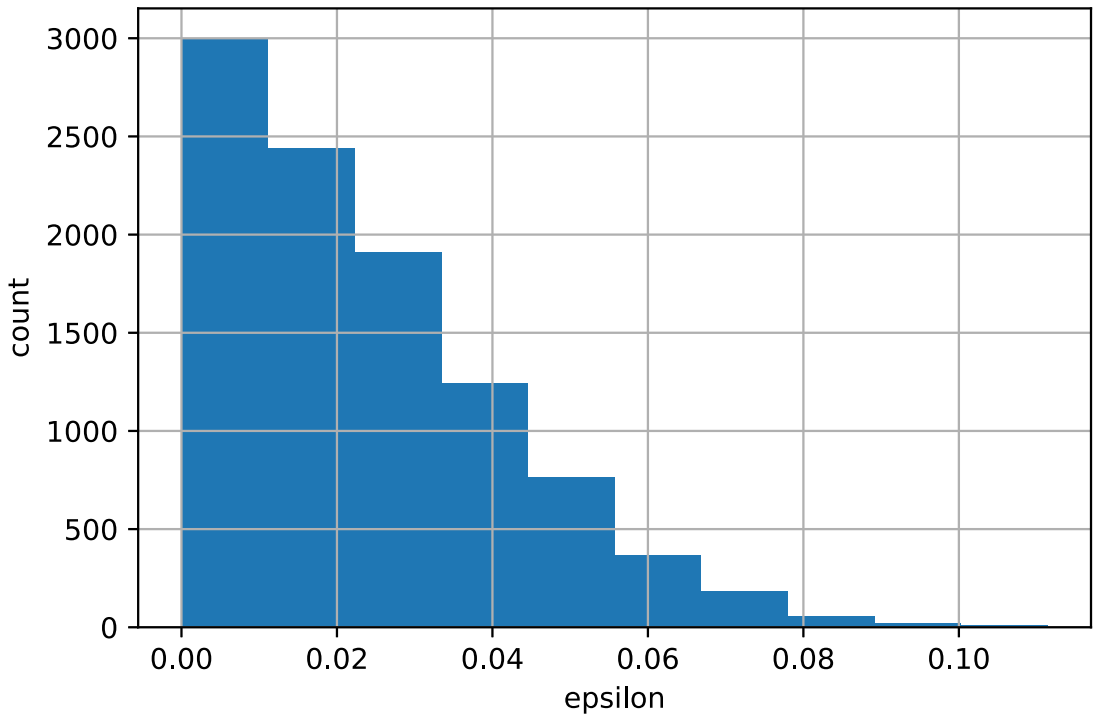


Sparse Random Projection

- More computation can be saved by using a *sparse* random projection matrix
  - "sparse" means that many entries in the basis vector are zero, so we can ignore those entries when multiplying.

In [25]:
```
# project the digits data with Random Projection
srp = random_projection.SparseRandomProjection(n_components=500, random_state=4487
)
Wsrp = srp.fit_transform(X)

# calculate pairwise distances
D = metrics.pairwise.euclidean_distances(X, X)
Dsrp = metrics.pairwise.euclidean_distances(Wsrp, Wsrp)

# calculate epsilon
epss = abs(Dsrp/D-1)
epss[isnan(epss)] = 0 # remove 0/0
plt.hist(epss.flatten())
plt.xlabel('epsilon')
plt.ylabel('count')
plt.grid(True)
```

```
/anaconda3/lib/python3.5/site-packages/ipykernel_launcher.py:10: RuntimeWarnin
g: divide by zero encountered in true_divide
  # Remove the CWD from sys.path while we load stuff.
/anaconda3/lib/python3.5/site-packages/ipykernel_launcher.py:10: RuntimeWarnin
g: invalid value encountered in true_divide
  # Remove the CWD from sys.path while we load stuff.
```

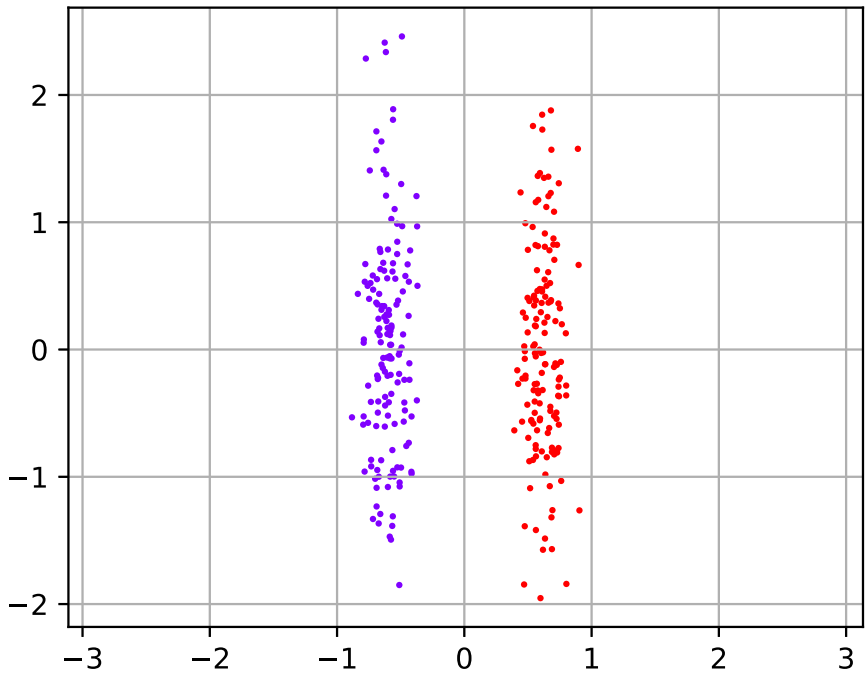


Question

- Suppose we have data for the below classification problem...
- We want to reduce the data to 1 dimension using PCA.
  - What is the first PC?

In [27]:
```
efig
```

Out[27]:

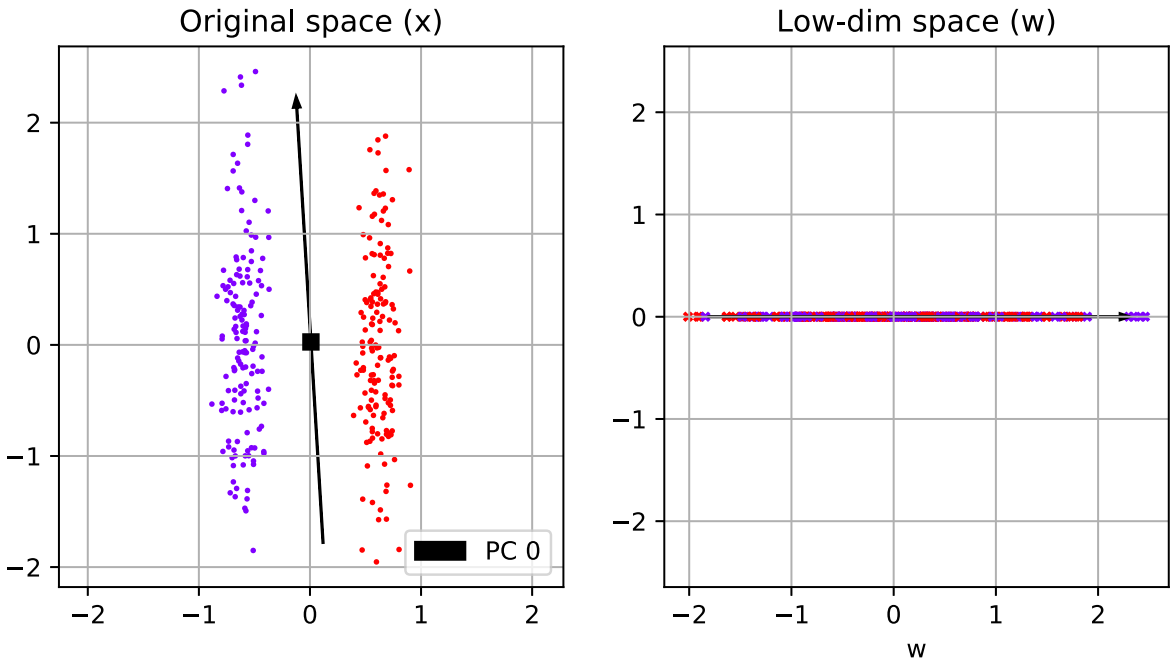


Answer

- first PC is along the direction of most variance.
  - collapses the two classes together!

In [28]:
```
e2fig
```

Out[28]:



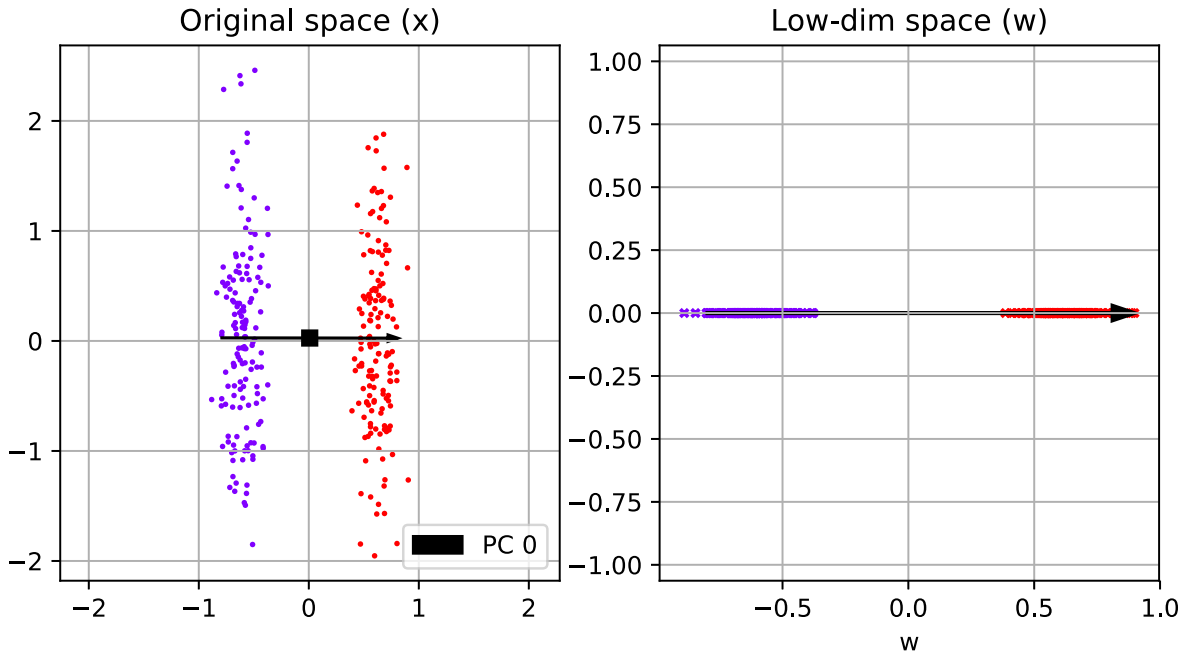
Problem with Unsupervised Methods

- If our end goal is classification, preserving the variance sometimes won't help!
  - PCA doesn't consider which class the data belongs to.
  - When the "classification" signal is less than the "noise", PCA will make classification more difficult.

Fisher's Linear Discriminant (FLD)

- Supervised dimensionality reduction
- Also called "*Linear Discriminant Analysis*" (LDA)
- **Goal:** find a lower-dim space so as to minimize the class overlap (or maximize the class separation).
  - data from each class is modeled as a Gaussian.
  - requires the class labels

In [29]:
```
# example of FLD projection (using LDA name)
fld = discriminant_analysis.LinearDiscriminantAnalysis(n_components=1)
W  = fld.fit_transform(X, Y)

v = fld.coef_   # the basis vectors

plt.figure(figsize=(8,4))
plot_basis(X, v, Y=Y);
```



On Iris data

- 4D vector to 2D vector
- FLD forms more compact classes
- With FLD, classes have less overlap if only using 1st basis vector.

In [31]:
```
ifig
```

Out[31]: