CS4487 - Machine Learning

Lecture 8 - Non-Linear Dimensionality Reduction, Manifold Embedding

Dr. Antoni B. Chan

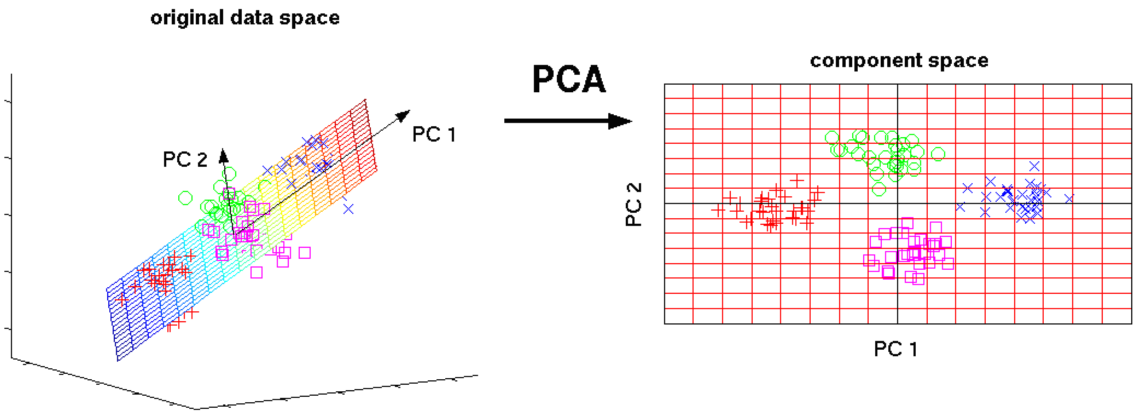
Dept. of Computer Science, City University of Hong Kong

Outline

1. Non-Linear Dimensionality Reduction
    A. Kernel Principal Component Analysis (KPCA)
2. Manifold Embedding
    A. Locally-linear embedding (LLE)
    B. Multi-dimensional Scaling (MDS)
    C. Isometric Mapping (Isomap)
    D. Spectral Embedding (Laplacian Eigenmaps)
    E. t-distributed Stochastic Neighbor Embedding (t-SNE)

Linear Dimensionality Reduction

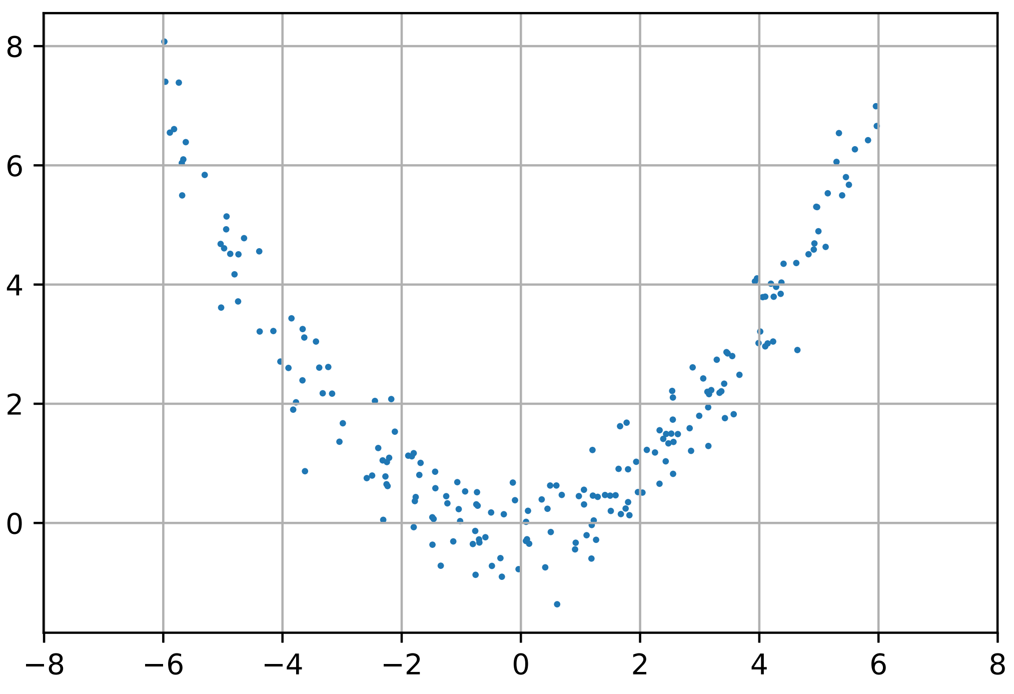
- PCA, NMF, LSA are all linear dimensionality reduction methods
    - model the data as "living" on a linear manifold (line, plane, etc).



Non-linear surface

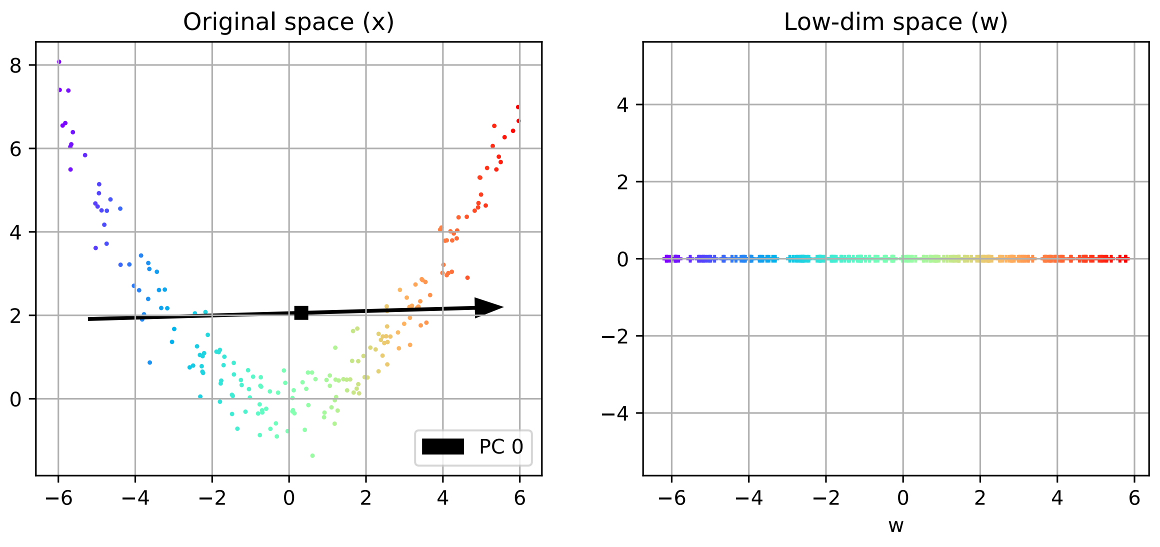
- What if the data "lives" on a non-flat surface?

In [4]:
```
pfig
```
Out[4]:



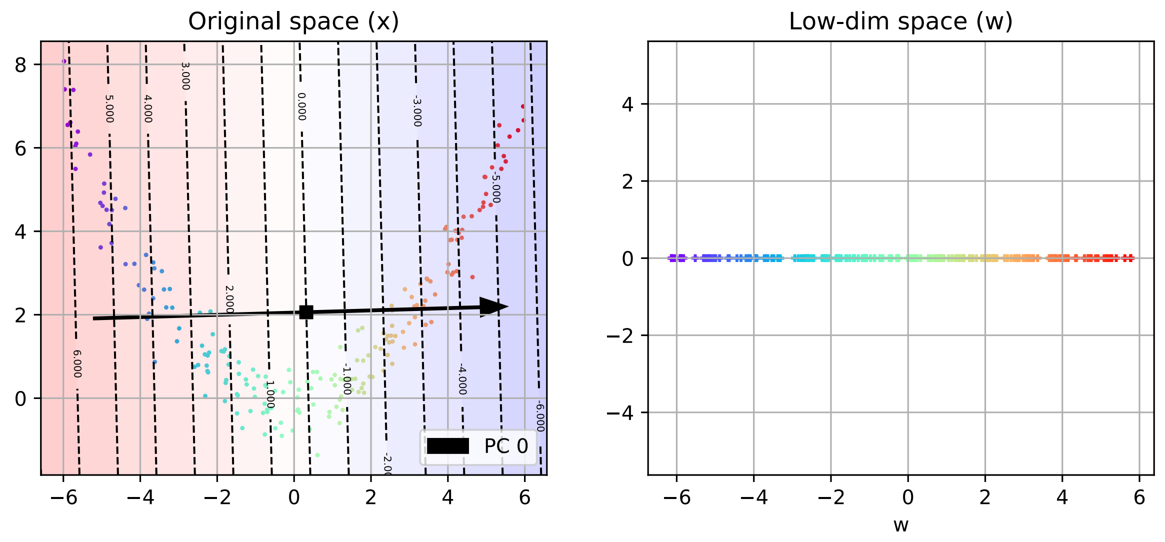
- PCA can't capture the curvature of the data
  - purple points are close together
  - red points are close together

In [5]:
```
pca = decomposition.PCA(n_components=1)
W   = pca.fit_transform(X)

plt.figure(figsize=(10,4))
plot_basis(X, pca.components_, Y=Y, showlowarrow=False)
```



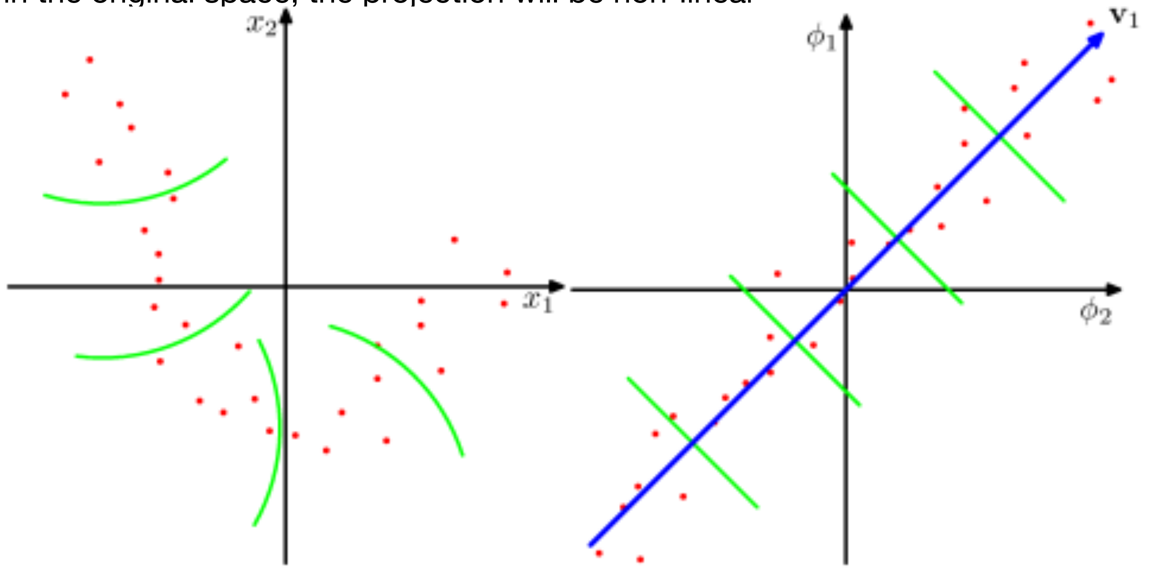
- iso-contours of PCA projection
  - points on the same dashed line are projected to the same PCA coefficient.

In [6]:
```
plt.figure(figsize=(10,4))
plot_basis(X, pca.components_, Y=Y, showcontours=True, pca=pca, showlowarrow=False)
```



Kernel PCA

- *How to project to a non-linear surface?*
  - apply a high-dimensional feature transformation to the data
    - $\mathbf{x}_i \Rightarrow \phi(\mathbf{x}_i)$
  - project high-dim data to a linear surface
    - i.e. run PCA on $\phi(\mathbf{x}_i)$
  - in the original space, the projection will be non-linear



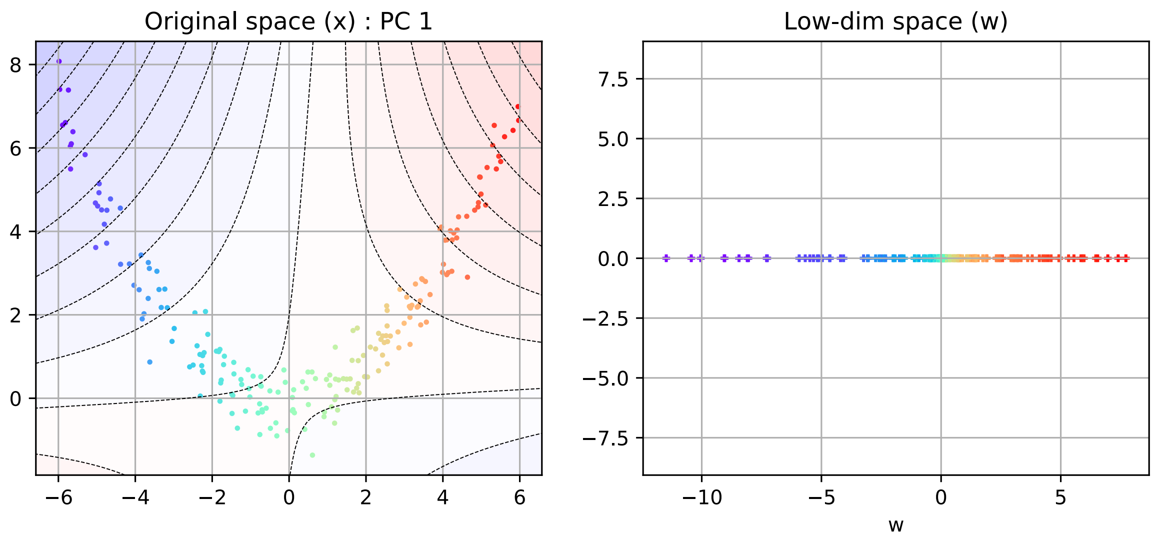
Kernel principal components

- kernel principal component $\mathbf{v}$ is a linear combination of high-dim vectors
  - $\mathbf{v} = \sum_{i=1}^{n} \alpha_i \phi(\mathbf{x}_i)$
  - where α_i are learned weights.
- For a new point $\mathbf{x}_*$, the KPCA coefficient for $\mathbf{v}$ is
  - $w = \phi(\mathbf{x}_*)^T \mathbf{v} = \sum_{i=1}^{n} \alpha_i \phi(\mathbf{x}_*)^T \phi(\mathbf{x}_i) = \sum_{i=1}^{n} \alpha_i k(\mathbf{x}_*, \mathbf{x}_i)$
  - coefficient is based on similarity to data points belonging to $\mathbf{v}$.
  - using the kernel trick saves computation.

- Example using polynomial kernel
  - purple points are further apart.
  - PC coefficient corresponds to location along the data curve.

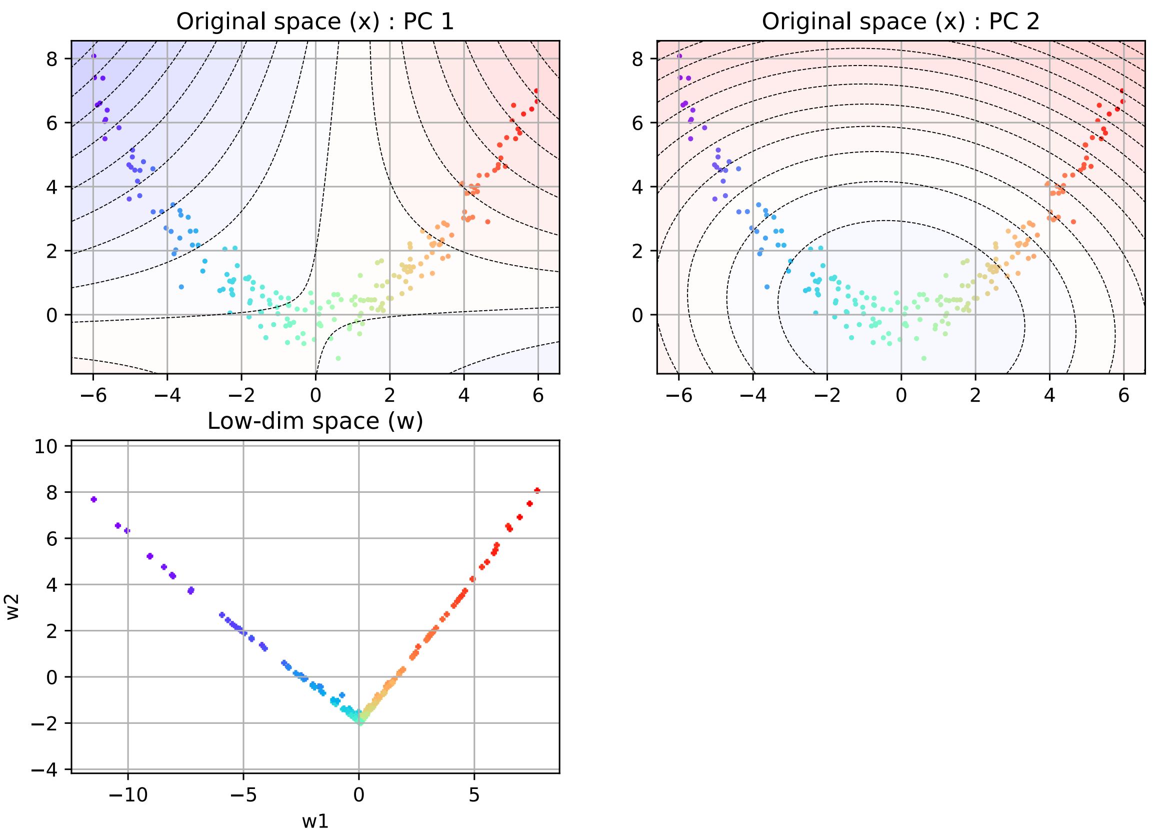
In [8]:

```
# run KPCA
kpca = decomposition.KernelPCA(n_components=1, kernel='poly', gamma=0.15, degree
=2, coef0=0)
W = kpca.fit_transform(X)

plt.figure(figsize=(10,4))
plot_kpca(X, W, kpca, showcontours=True, Y=Y)
```



- Example: 2 PCs
  - 2nd PC corresponds to the distance from the center

In [9]:
```python
# run KPCA
kpca = decomposition.KernelPCA(n_components=2, kernel='poly', gamma=0.15, degree
=2, coef0=0)
W = kpca.fit_transform(X)

plt.figure(figsize=(10,7))
plot_kpca(X, W, kpca, showcontours=True, Y=Y)
```



RBF kernel

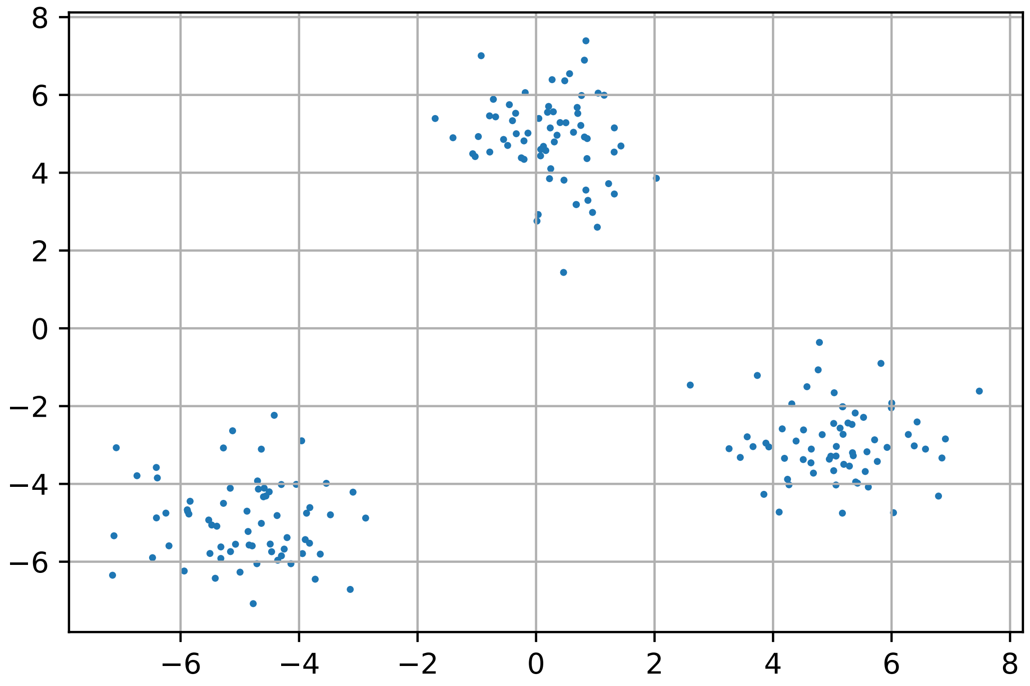
- principal components separate the data into clusters
- coefficient is distance to clusters

Example

- data with 3 clusters

In [11]:

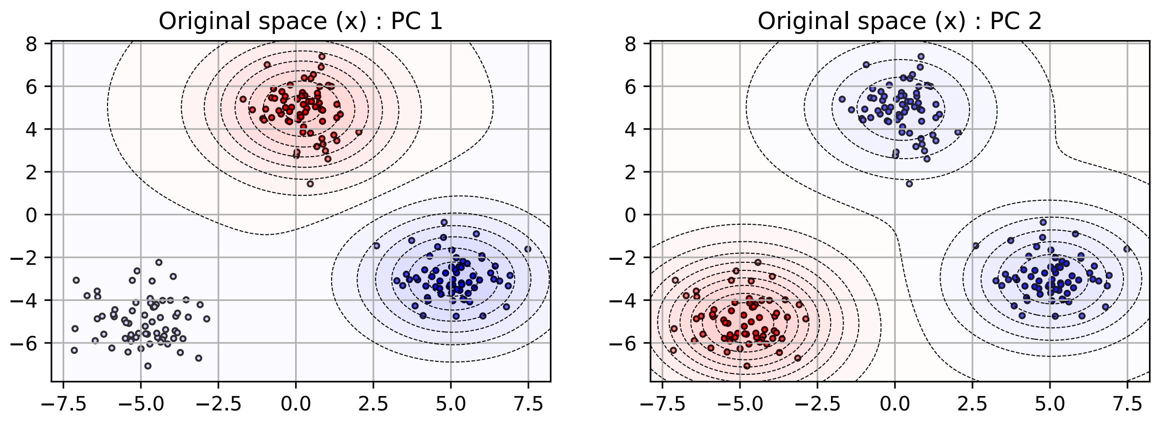
```
plt.scatter(X[:,0], X[:,1], s=6, edgecolor="")
plt.grid(True);
```



- The first 2 PCs can split the data into 3 clusters
  - the color of the datapoint corresponds to the coefficient value.

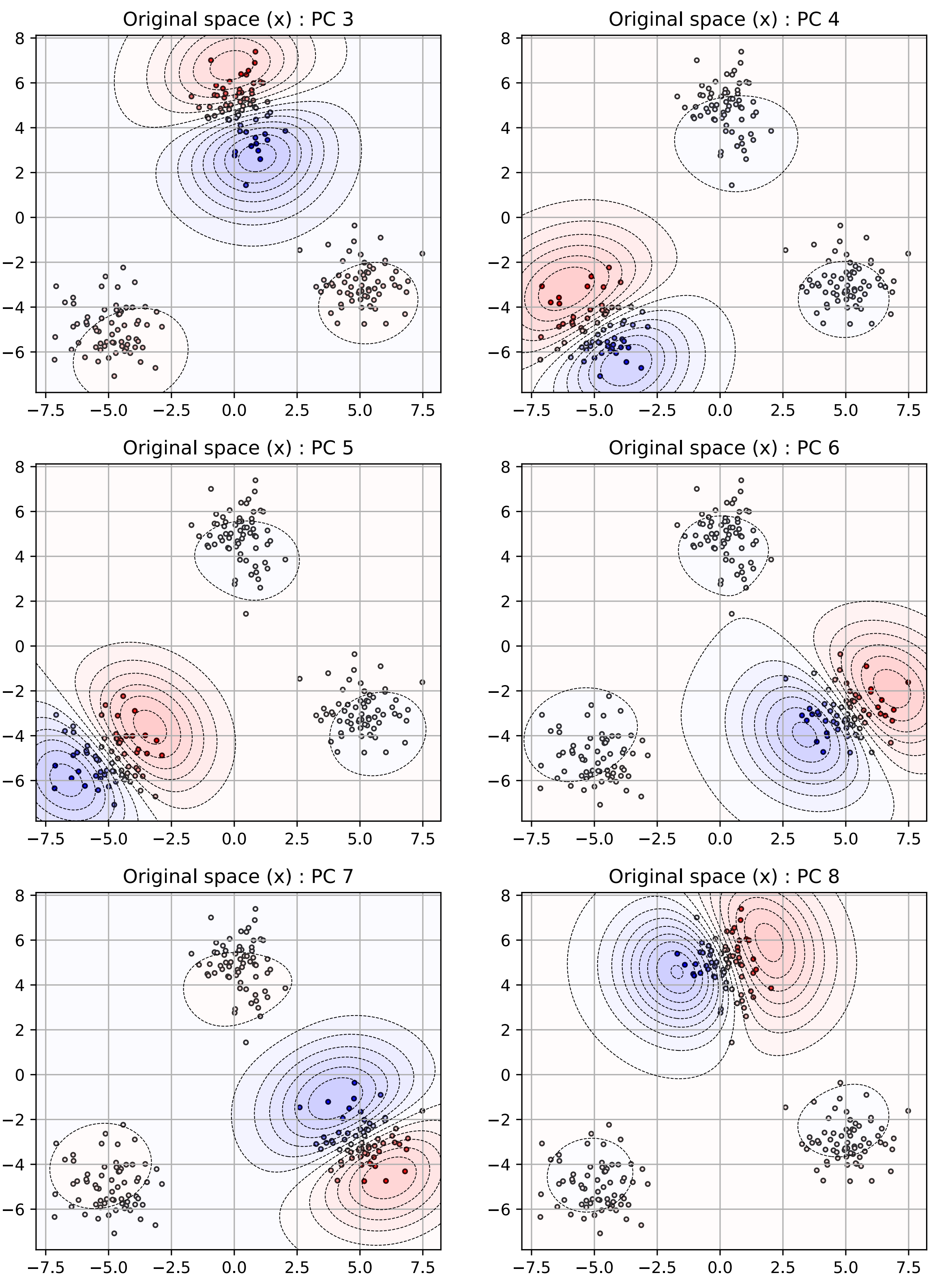
In [12]:

```
# run KPCA
kpca = decomposition.KernelPCA(n_components=8, kernel='rbf', gamma=0.15)
W = kpca.fit_transform(X)

plt.figure(figsize=(10,7))
plot_kpca(X, W, kpca, showcontours=True, showpcs=[0,1], colorcoefs=True)
```



- The remaining 6 PCs split each cluster into halves
  - multiple splits in orthogonal directions

In [13]:
```
plt.figure(figsize=(10,14))
plot_kpca(X, W, kpca, showcontours=True, showpcs=range(2,8), colorcoefs=True)
```



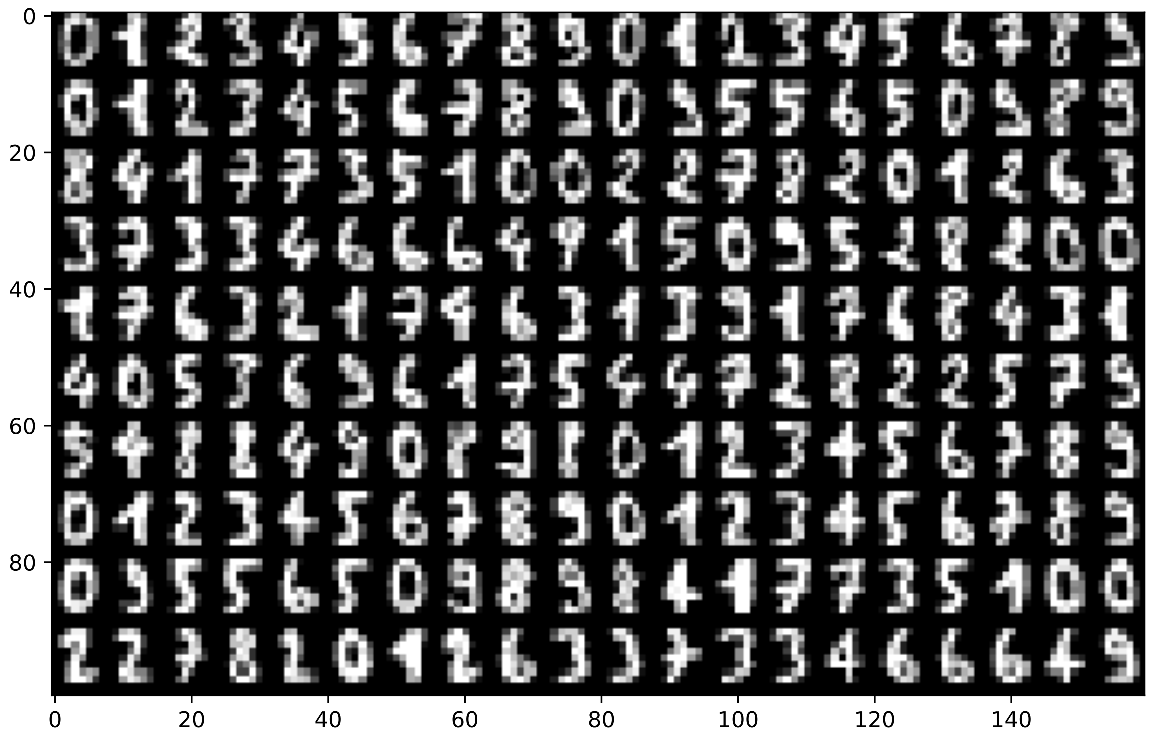
Example on digit images

- 8 x 8 images -> 64D vector

In [14]:
```
digits = datasets.load_digits(n_class=10)
X = digits.data
Y = digits.target

# randomly split data into training and testing
trainX, testX, trainY, testY = \
  model_selection.train_test_split(X, Y,
  train_size=0.8, test_size=0.2, random_state=4487)
```

In [16]:
```
digitfig
```

Out[16]:

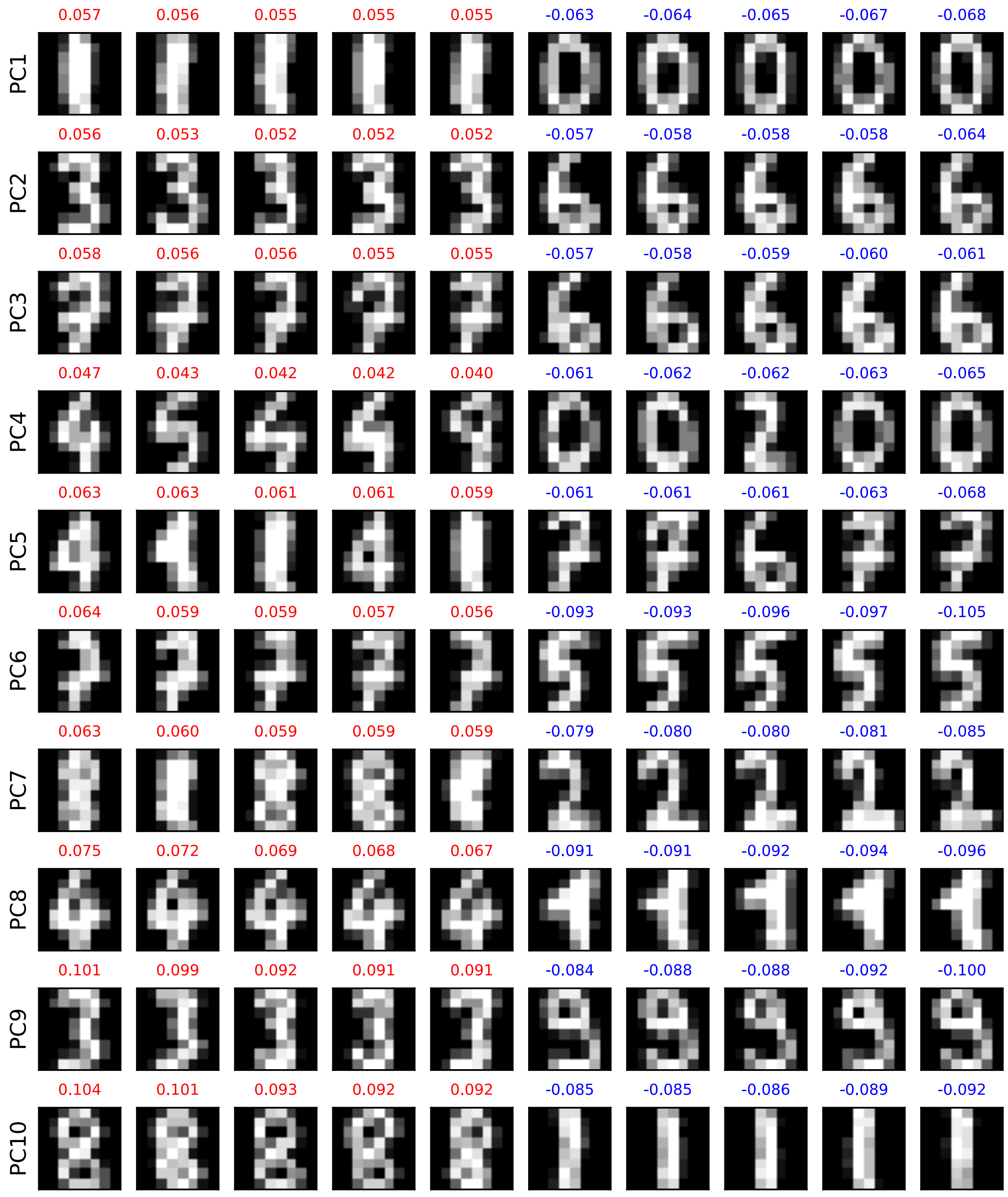


- Apply KPCA with RBF kernel
  - (parallelize with `n_jobs`)

In [17]:
```
kpca = decomposition.KernelPCA(n_components=10, kernel='rbf', gamma=0.001, n_job
s=-1)
trainW = kpca.fit_transform(trainX)
```

- Top-5 positive and negative prototypes for each PC
  - the number is the α value for that image.
  - from the prototypes, the PCs are modeling the differences in appearance between digits

In [19]:
```
plt.figure(figsize=(8,10))
plot_kbasis(kpca, (8,8), trainX)
```



Classification experiment

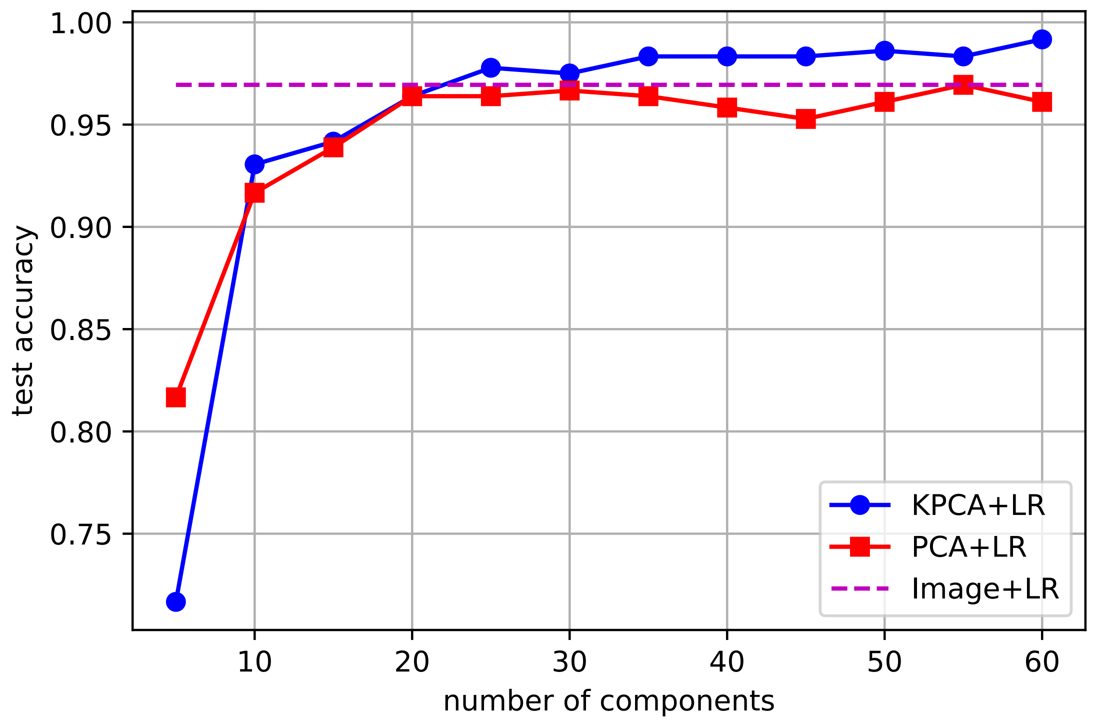
- use KPCA coefficients as the new representation
  - train a logistic regression classifier
  - try different numbers of components
    - Note: can do this efficiently by selecting a subset of KPCA components.

In [20]:
```
# apply kernel PCA
kpca = decomposition.KernelPCA(n_components=60, kernel='rbf', gamma=0.001, n_job
s=-1)
trainW = kpca.fit_transform(trainX)
testW = kpca.transform(testX)
```

In [21]:
```
ncs = [5, 10, 15, 20, 25, 30, 35, 40, 45, 50, 55, 60]
accs = []
for nc in ncs:
    # extract the first nc PCs
    trainWnew = trainW[:,0:nc]
    testWnew  = testW[:,0:nc]

    # train classifier
    logreg = linear_model.LogisticRegressionCV(Cs=logspace(-4,4,10), cv=5, n_job
s=-1)
    logreg.fit(trainWnew, trainY)

    # test classifier
    predYtest  = logreg.predict(testWnew)
    acc        = metrics.accuracy_score(testY, predYtest)
    accs.append(acc)
```

- Classification results on test set
    - KPCA can improve the performance, compared with PCA and raw image.

In [24]:
```python
# make a plot
plt.plot(ncs, accs, 'bo-', label='KPCA+LR')
plt.plot(ncs, accs_pca, 'rs-', label='PCA+LR')
plt.plot([min(ncs), max(ncs)], [acc_raw, acc_raw], 'm--', label='Image+LR')
plt.legend(loc="best")
plt.xlabel('number of components')
plt.ylabel('test accuracy')
plt.grid(True)
```



KPCA Summary

- Use kernel trick to perform PCA in high-dimensional space.
  - Coefficients are based on a non-linear projection of the data.
  - The type of projection is based on the kernel function selected.
- Using RBF kernel, KPCA can split the data into clusters.