DimensionalityReduction in Python

October 28, 2018

```
In [5]: import numpy as np
        import matplotlib.pyplot as plt
        from sklearn import datasets
```

1 PCA

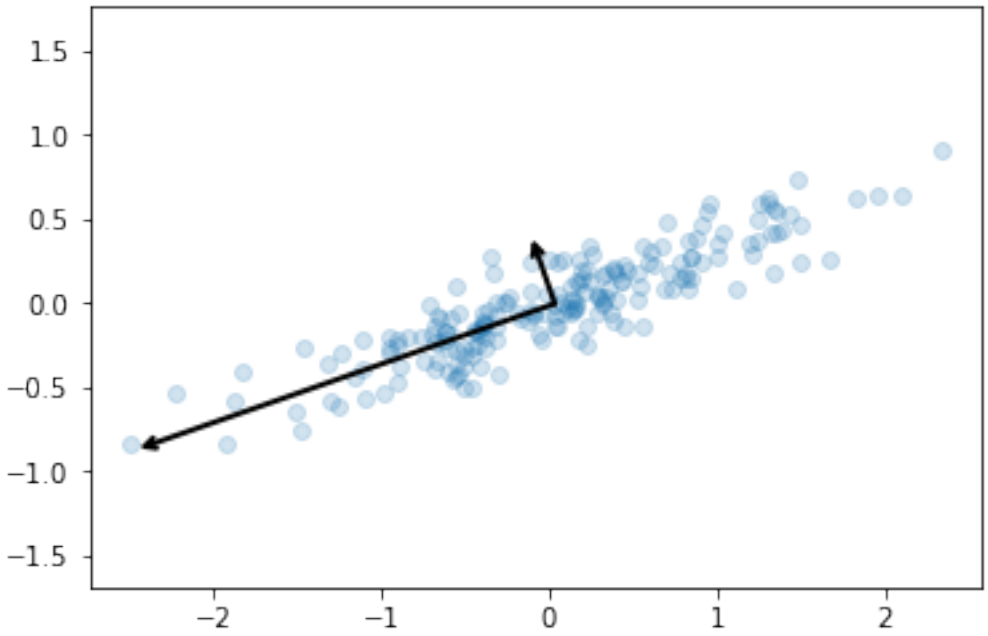
```
In [36]: from sklearn.decomposition import PCA

         rng = np.random.RandomState(1)
         X = np.dot(rng.rand(2, 2), rng.randn(2, 200)).T

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

         print("Components = ",pca.components_)
         print("Explained variance = ",pca.explained_variance_)
```

```
Components =  [[-0.94446029 -0.32862557]
 [-0.32862557  0.94446029]]
Explained variance =  [0.7625315 0.0184779]
```

```
In [37]: def draw_vector(v0, v1, ax=None):
             ax = ax or plt.gca()
             arrowprops=dict(arrowstyle='->',
                             linewidth=2,
                             shrinkA=0, shrinkB=0)
             ax.annotate('', v1, v0, arrowprops=arrowprops)

         # plot data
         plt.scatter(X[:, 0], X[:, 1], alpha=0.2)
         for length, vector in zip(pca.explained_variance_, pca.components_):
             v = vector * 3 * np.sqrt(length)
             draw_vector(pca.mean_, pca.mean_ + v)
         plt.axis('equal')
```

```
Out[37]: (-2.7391278364515688,
          2.5801310701596343,
```

```
-0.9477947579593763,
1.0195904306706842)
```



1.1 Visualizing PCA

```
In [36]: from sklearn.decomposition import PCA
         iris = datasets.load_iris()
         df = pd.DataFrame(data=np.c_[iris['data']], columns=iris['feature_names'])

         pca = PCA(n_components=2, svd_solver='full')
         pca.fit(df)
         T = pca.transform(df)
         print(df.shape,T.shape)
```

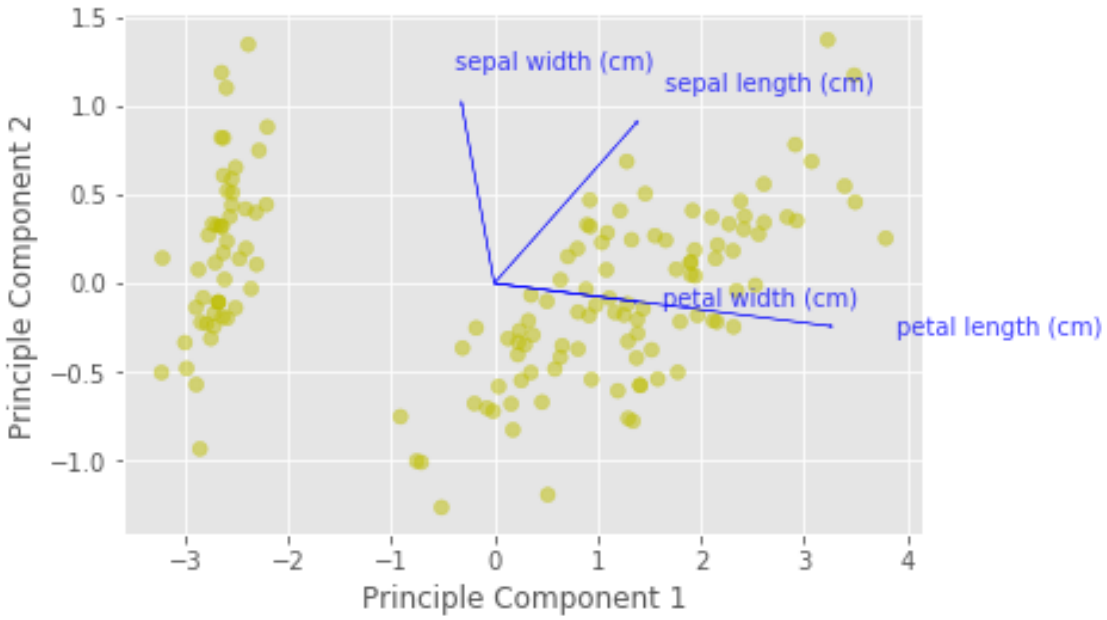
```
(150, 4) (150, 2)
```

```
In [33]: import math
         def get_important_features(transformed_features, components_, columns):
             num_columns = len(columns)
             # Scale the principal components by the max value in the transformed set
             xvector = components_[0] * max(transformed_features[:,0])
             yvector = components_[1] * max(transformed_features[:,1])
             # Sort each original column by it's length
             impt_features = { columns[i] : math.sqrt(xvector[i]**2 + yvector[i]**2)
                                 for i in range(num_columns) }
```

```
            impt_features = sorted(zip(impt_features.values(),impt_features.keys()),reverse=T
            print("Features by importance:\n", impt_features)

        get_important_features(T, pca.components_, df.columns.values)
```

```
Features by importance:
 [(3.2593371676186176, 'petal length (cm)'), (1.640840904115524, 'sepal length (cm)'), (1.3655
```

```
In [34]: plt.style.use('ggplot')
         def draw_vectors(transformed_features, components_, columns):
             num_columns = len(columns)
             xvector = components_[0] * max(transformed_features[:,0])
             yvector = components_[1] * max(transformed_features[:,1])
             ax = plt.axes()
             for i in range(num_columns):
                 # Use an arrow to project each original feature as a labeled vector
                 plt.arrow(0, 0, xvector[i], yvector[i], color='b',
                           width=0.0005, head_width=0.02, alpha=0.75)
                 plt.text(xvector[i]*1.2,yvector[i]*1.2,list(columns)[i],color='b',alpha=0.75)

             return ax

         ax = draw_vectors(T, pca.components_, df.columns.values)
         T_df = pd.DataFrame(T)
         T_df.columns = ['component1', 'component2']
         T_df['color'] = 'y'
         T_df.loc[T_df['component1'] > 125, 'color'] = 'g'
         T_df.loc[T_df['component2'] > 125, 'color'] = 'r'
         plt.xlabel('Principle Component 1')
         plt.ylabel('Principle Component 2')
         plt.scatter(T_df['component1'], T_df['component2'], color=T_df['color'], alpha=0.5)
         plt.show()
```

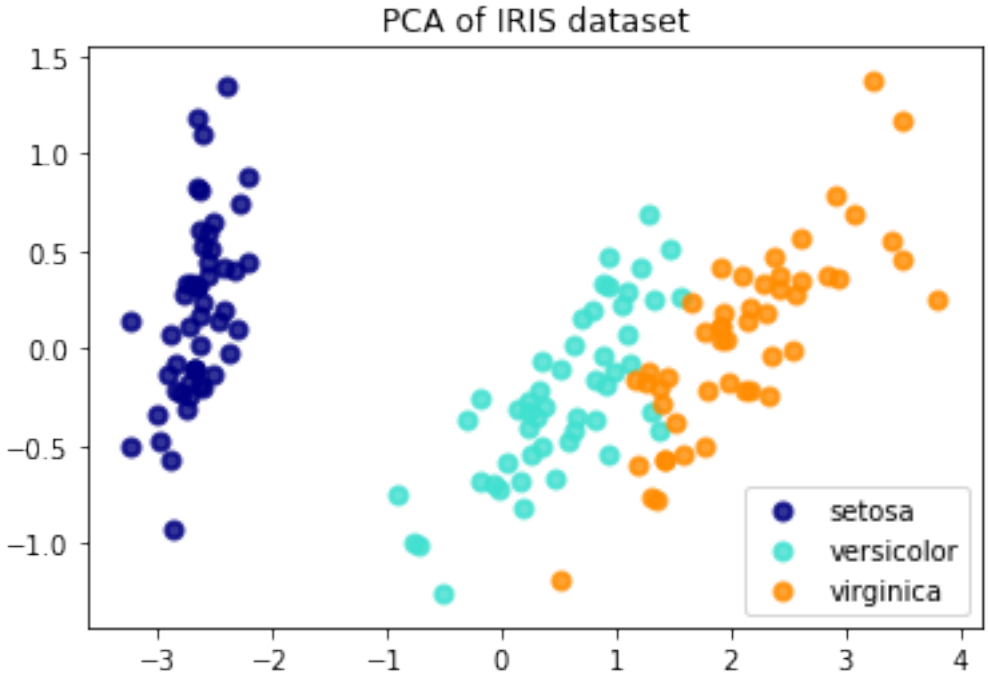


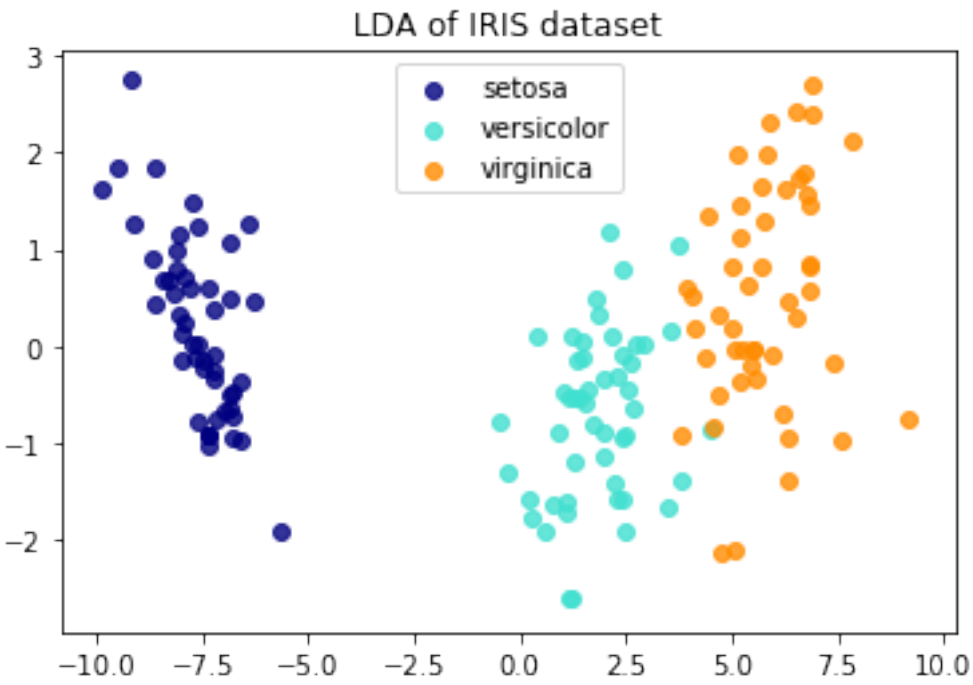
1.2 Comparing PCA with LDA

```
In [11]: from sklearn.decomposition import PCA
         from sklearn.discriminant_analysis import LinearDiscriminantAnalysis

         iris = datasets.load_iris()
         X, y = iris.data, iris.target
         target_names = iris.target_names

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

         lda = LinearDiscriminantAnalysis(n_components=2)
         X_r2 = lda.fit(X, y).transform(X)

         # Percentage of variance explained for each components
         print('explained variance ratio (first two components): %s'
               % str(pca.explained_variance_ratio_))

         plt.figure()
         colors = ['navy', 'turquoise', 'darkorange']
         for color, i, target_name in zip(colors, [0, 1, 2], target_names):
             plt.scatter(X_r[y==i,0],X_r[y==i,1],color=color,alpha=.8,lw=2,label=target_name)
         plt.legend(loc='best', shadow=False, scatterpoints=1)
         plt.title('PCA of IRIS dataset')
         plt.figure()
```

```
        for color, i, target_name in zip(colors, [0, 1, 2], target_names):
            plt.scatter(X_r2[y==i,0],X_r2[y==i,1],alpha=.8,color=color,label=target_name)
        plt.legend(loc='best', shadow=False, scatterpoints=1)
        plt.title('LDA of IRIS dataset')
        plt.show()
```

```
explained variance ratio (first two components): [0.92461621 0.05301557]
```





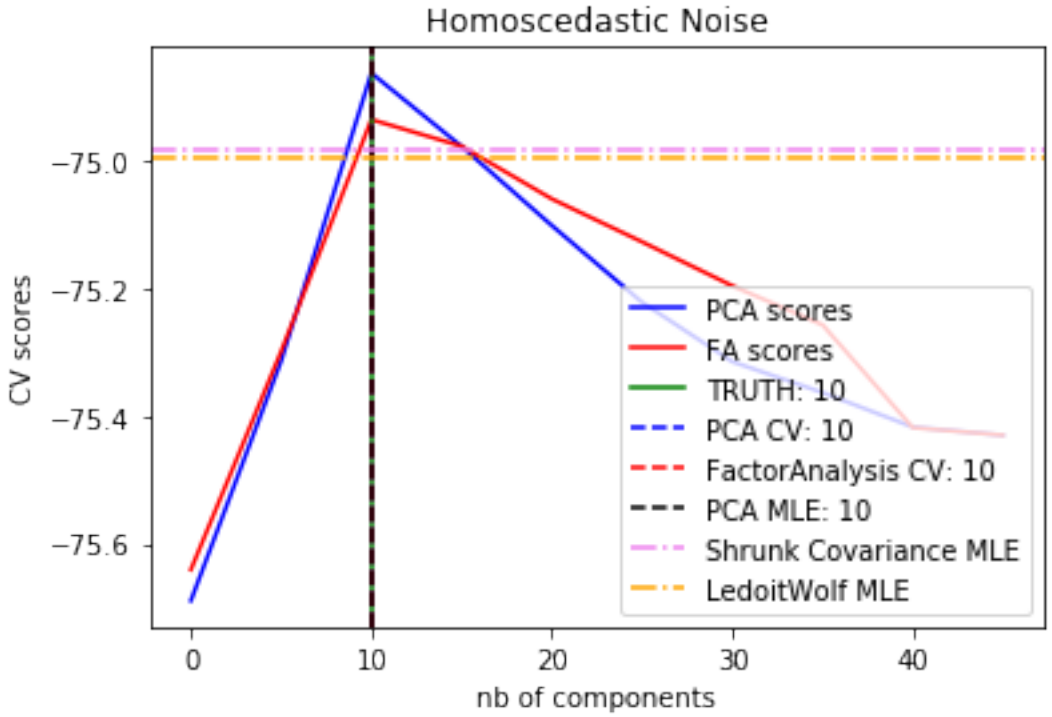
1.3 PCA noise variation with #components

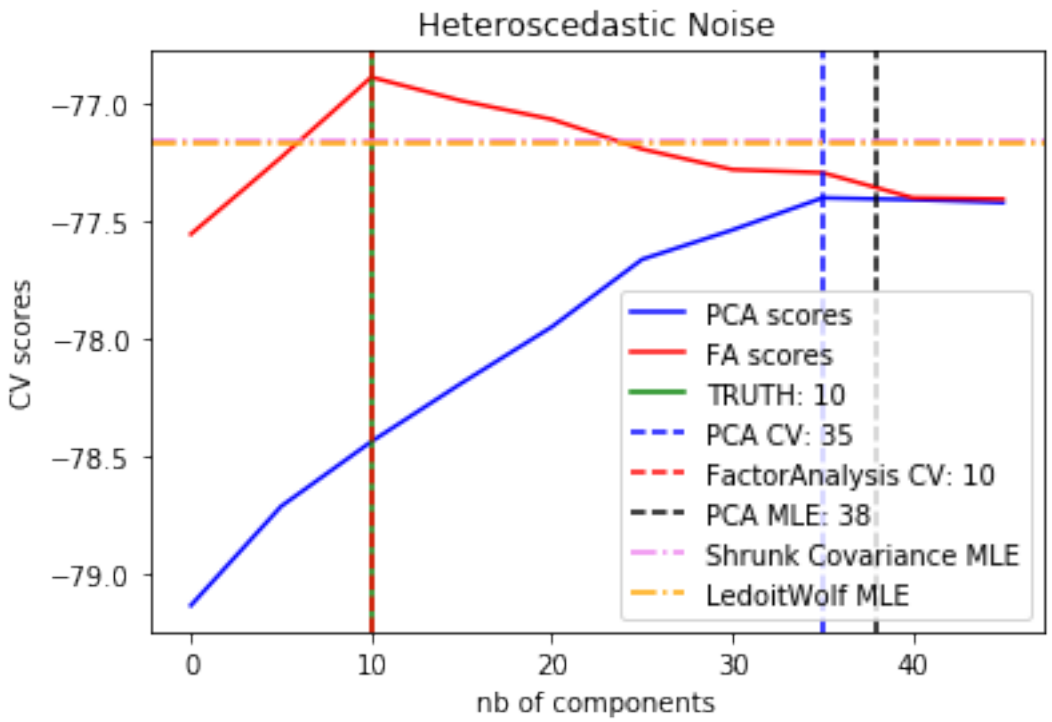
```
In [12]: from scipy import linalg
         from sklearn.decomposition import PCA, FactorAnalysis
         from sklearn.covariance import ShrunkCovariance, LedoitWolf
         from sklearn.model_selection import cross_val_score
         from sklearn.model_selection import GridSearchCV

         # Create the data
         n_samples, n_features, rank = 1000, 50, 10
         sigma = 1.
         rng = np.random.RandomState(42)
         U, _, _ = linalg.svd(rng.randn(n_features, n_features))
         X = np.dot(rng.randn(n_samples, rank), U[:, :rank].T)
         X_homo = X + sigma * rng.randn(n_samples, n_features) #add homoscedastic noise
         sigmas = sigma * rng.rand(n_features) + sigma / 2.
         X_hetero = X + rng.randn(n_samples, n_features) * sigmas #add heteroscedastic noise

         # Fit the models
         n_components = np.arange(0, n_features, 5)  # options for n_components

         def compute_scores(X):
             pca = PCA(svd_solver='full')
             fa = FactorAnalysis()
```

```
    pca_scores, fa_scores = [], []
    for n in n_components:
        pca.n_components = n
        fa.n_components = n
        pca_scores.append(np.mean(cross_val_score(pca, X, cv=5)))
        fa_scores.append(np.mean(cross_val_score(fa, X, cv=5)))
    return pca_scores, fa_scores

def shrunk_cov_score(X):
    shrinkages = np.logspace(-2, 0, 30)
    cv = GridSearchCV(ShrunkCovariance(), {'shrinkage': shrinkages}, cv=5)
    return np.mean(cross_val_score(cv.fit(X).best_estimator_, X, cv=5))

def lw_score(X):
    return np.mean(cross_val_score(LedoitWolf(), X, cv=5))


for X, title in [(X_homo,'Homoscedastic Noise'),(X_hetero,'Heteroscedastic Noise')]:
    pca_scores, fa_scores = compute_scores(X)
    n_components_pca = n_components[np.argmax(pca_scores)]
    n_components_fa = n_components[np.argmax(fa_scores)]

    pca = PCA(svd_solver='full', n_components='mle')
    pca.fit(X)
    n_components_pca_mle = pca.n_components_

    print("best n_components by PCA CV = %d" % n_components_pca)
    print("best n_components by FactorAnalysis CV = %d" % n_components_fa)
    print("best n_components by PCA MLE = %d" % n_components_pca_mle)
    plt.figure()
    plt.plot(n_components, pca_scores, 'b', label='PCA scores')
    plt.plot(n_components, fa_scores, 'r', label='FA scores')
    plt.axvline(rank, color='g', label='TRUTH: %d' % rank, linestyle='-')
    plt.axvline(n_components_pca, color='b',
                label='PCA CV: %d' % n_components_pca, linestyle='--')
    plt.axvline(n_components_fa, color='r',
                label='FactorAnalysis CV: %d'%n_components_fa,linestyle='--')
    plt.axvline(n_components_pca_mle, color='k',
                label='PCA MLE: %d' % n_components_pca_mle, linestyle='--')

    # compare with other covariance estimators
    plt.axhline(shrunk_cov_score(X), color='violet',
                label='Shrunk Covariance MLE', linestyle='-.')
    plt.axhline(lw_score(X), color='orange',
                label='LedoitWolf MLE' % n_components_pca_mle, linestyle='-.')
    plt.xlabel('nb of components')
    plt.ylabel('CV scores')
    plt.legend(loc='lower right')
```

```
            plt.title(title)
        plt.show()
```

```
best n_components by PCA CV = 10
best n_components by FactorAnalysis CV = 10
best n_components by PCA MLE = 10
best n_components by PCA CV = 35
best n_components by FactorAnalysis CV = 10
best n_components by PCA MLE = 38
```

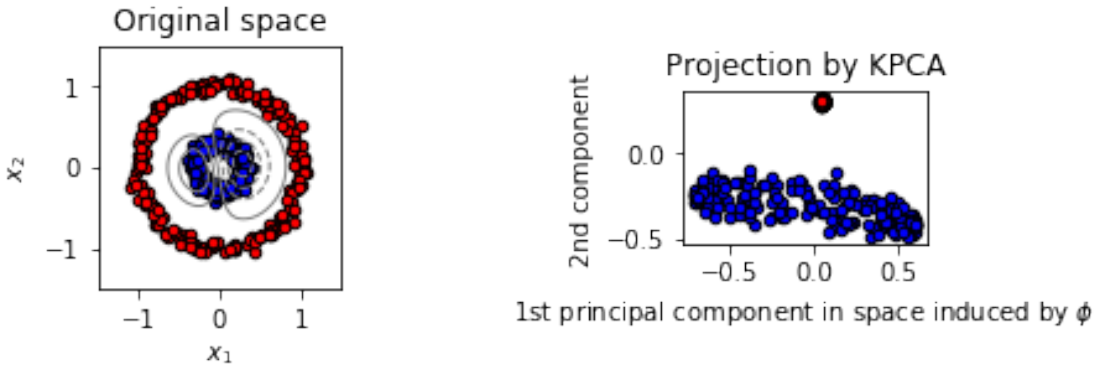




1.4 Kernel PCA

```
In [35]: from sklearn.decomposition import PCA, KernelPCA
         from sklearn.datasets import make_circles

         np.random.seed(0)
         X, y = make_circles(n_samples=400, factor=.3, noise=.05)

         kpca = KernelPCA(kernel="rbf", fit_inverse_transform=True, gamma=10)
         X_kpca = kpca.fit_transform(X)
         X_back = kpca.inverse_transform(X_kpca)
         pca = PCA()
         X_pca = pca.fit_transform(X)

         # Plot results
         plt.figure()
         plt.subplot(1, 3, 1, aspect='equal')
         plt.title("Original space")
         reds = y == 0
         blues = y == 1
         plt.scatter(X[reds, 0], X[reds, 1], c="red", s=20, edgecolor='k')
         plt.scatter(X[blues, 0], X[blues, 1], c="blue", s=20, edgecolor='k')
         plt.xlabel("$x_1$")
```

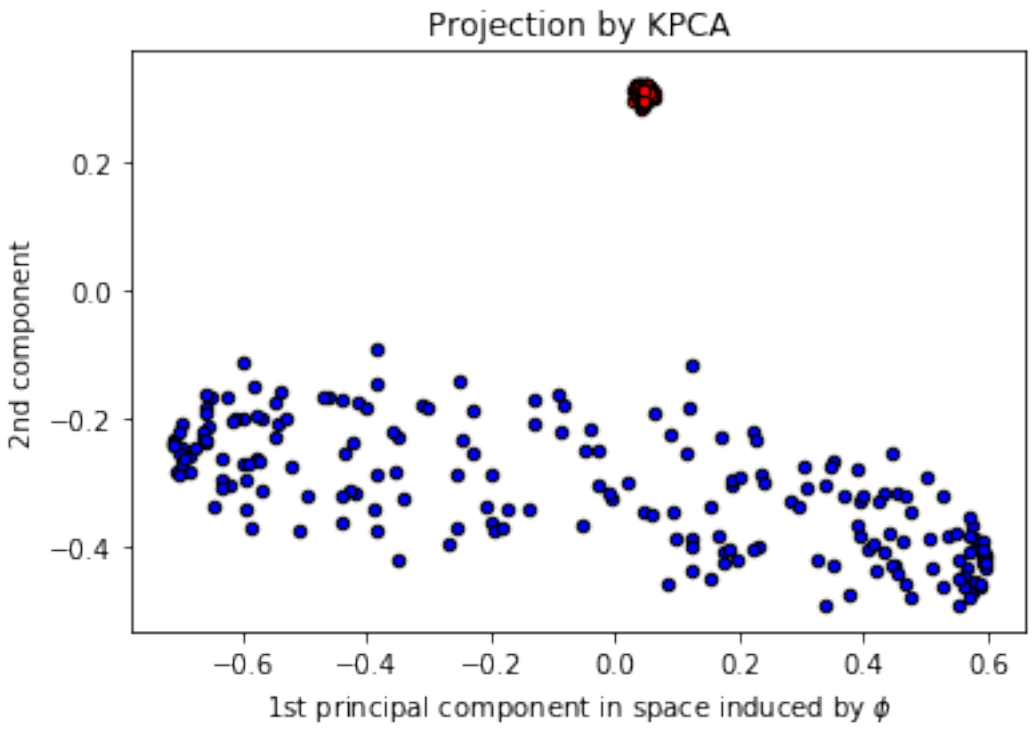
```
plt.ylabel("$x_2$")

X1, X2 = np.meshgrid(np.linspace(-1.5, 1.5, 50), np.linspace(-1.5, 1.5, 50))
X_grid = np.array([np.ravel(X1), np.ravel(X2)]).T

# projection on the first principal component (in the phi space)
Z_grid = kpca.transform(X_grid)[:, 0].reshape(X1.shape)
plt.contour(X1, X2, Z_grid, colors='grey', linewidths=1, origin='lower')

plt.subplot(1, 3, 3, aspect='equal')
plt.scatter(X_kpca[reds, 0], X_kpca[reds, 1], c="red", s=20, edgecolor='k')
plt.scatter(X_kpca[blues,0], X_kpca[blues,1], c="blue",s=20, edgecolor='k')
plt.title("Projection by KPCA")
plt.xlabel("1st principal component in space induced by $\phi$")
plt.ylabel("2nd component")
plt.show()
```



1.5 Ploting projected axes

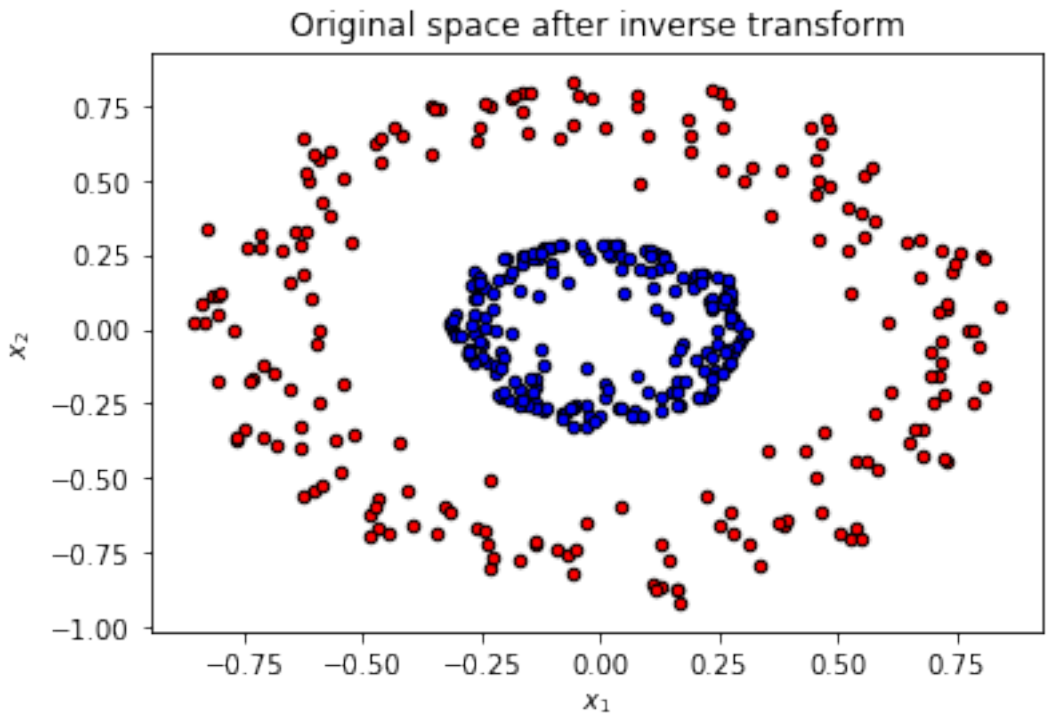
In [27]:
```
#Recovering previous (Kernel)PCA results
plt.scatter(X_kpca[reds, 0], X_kpca[reds, 1], c="red", s=20, edgecolor='k')
plt.scatter(X_kpca[blues,0], X_kpca[blues,1], c="blue",s=20, edgecolor='k')
plt.title("Projection by KPCA")
plt.xlabel("1st principal component in space induced by $\phi$")
plt.ylabel("2nd component")
```

Out[27]:
```
Text(0,0.5,'2nd component')
```



1.6 Reversing PCA (recovering original data space)

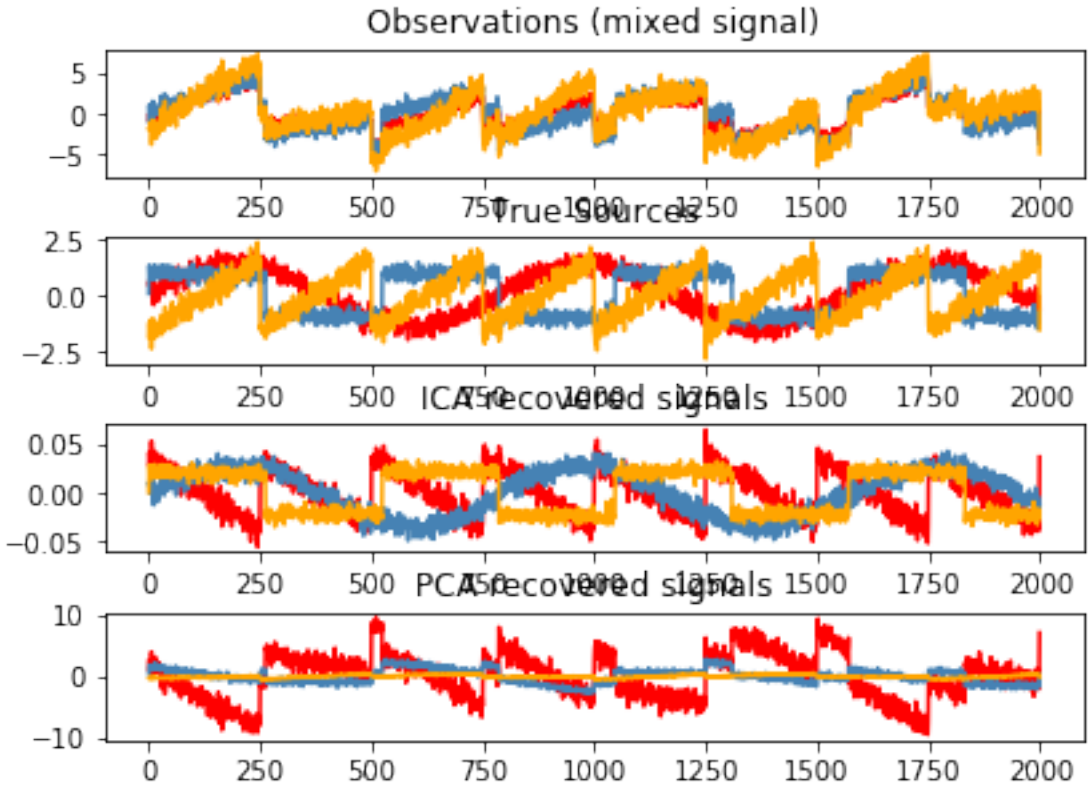
```
In [28]: #Recovering previous (Kernel)PCA results
         plt.scatter(X_back[reds, 0], X_back[reds, 1], c="red", s=20, edgecolor='k')
         plt.scatter(X_back[blues,0], X_back[blues,1], c="blue",s=20, edgecolor='k')
         plt.title("Original space after inverse transform")
         plt.xlabel("$x_1$")
         plt.ylabel("$x_2$")

Out[28]: Text(0,0.5,'$x_2$')
```



1.7 Comparing PCA with ICA

In [22]:
```
from scipy import signal
from sklearn.decomposition import FastICA, PCA

# Generate sample data
np.random.seed(0)
n_samples = 2000
time = np.linspace(0, 8, n_samples)
s1 = np.sin(2 * time)  # Signal 1 : sinusoidal signal
s2 = np.sign(np.sin(3 * time))  # Signal 2 : square signal
s3 = signal.sawtooth(2 * np.pi * time)  # Signal 3: saw tooth signal
S = np.c_[s1, s2, s3]
S += 0.2 * np.random.normal(size=S.shape)  # Add noise
S /= S.std(axis=0)  # Standardize data

# Mix data
A = np.array([[1, 1, 1], [0.5, 2, 1.0], [1.5, 1.0, 2.0]])  # Mixing matrix
X = np.dot(S, A.T)  # Generate observations

# Compute ICA
ica = FastICA(n_components=3)
S_ = ica.fit_transform(X)  # Reconstruct signals
```

```
A_ = ica.mixing_  # Get estimated mixing matrix

# Show that the ICA model applies by reverting the unmixing.
assert np.allclose(X, np.dot(S_, A_.T) + ica.mean_)

# For comparison, compute PCA
pca = PCA(n_components=3)
H = pca.fit_transform(X)  # Reconstruct signals based on orthogonal components

# Plot results
plt.figure()
models = [X, S, S_, H]
names = ['Observations (mixed signal)',
         'True Sources',
         'ICA recovered signals',
         'PCA recovered signals']
colors = ['red', 'steelblue', 'orange']
for ii, (model, name) in enumerate(zip(models, names), 1):
    plt.subplot(4, 1, ii)
    plt.title(name)
    for sig, color in zip(model.T, colors): plt.plot(sig, color=color)
plt.subplots_adjust(0.09, 0.04, 0.94, 0.94, 0.26, 0.46)
plt.show()
```



2 Random projections

Types of unstructured random matrix: Gaussian random matrix and sparse random matrix.

```
In [25]: from sklearn import random_projection
         X = np.random.rand(100, 10000)

         transformer = random_projection.GaussianRandomProjection()
         X_new = transformer.fit_transform(X)
         X_new.shape
```

```
Out[25]: (100, 3947)
```

```
In [26]: from sklearn import random_projection
         X = np.random.rand(100,10000)

         transformer = random_projection.SparseRandomProjection()
         X_new = transformer.fit_transform(X)
         X_new.shape
```

```
Out[26]: (100, 3947)
```