Compressing Data via Dimensionality Reduction

Principal Component Analysis

Total and Explained Variance

```
[2]  import pandas as pd
```

```
[3]  df_wine = pd.read_csv('https://archive.ics.uci.edu/ml/machine-learning-da
                           header=None)
```

```
[6]  from sklearn.cross_validation import train_test_split
     from sklearn.preprocessing import StandardScaler
```

```
[7]  X, y = df_wine.iloc[:,1:].values, df_wine.iloc[:,0].values
```

```
[8]  X_train, X_test, y_train, y_test = train_test_split(X, y,
                                                         test_size=0.3,
                                                         random_state=0)
```

```
[9]  sc = StandardScaler()
     X_train_std = sc.fit_transform(X_train)
     X_test_std = sc.transform(X_test)
```

```
[10] import numpy as np
```

```
[11] cov_mat = np.cov(X_train_std.T)
```

```
[12] eigen_vals, eigen_vecs = np.linalg.eig(cov_mat)
```

```
[14] print("\nEigenvalues \n {}".format(eigen_vals))
```

```
Eigenvalues
 [ 4.8923083   2.46635032  1.42809973  1.01233462  0.84906459  0.60181514
```

```
 0.52251546  0.08414846  0.33051429  0.29595018  0.16831254  0.21432212
 0.2399553 ]
```

[15]

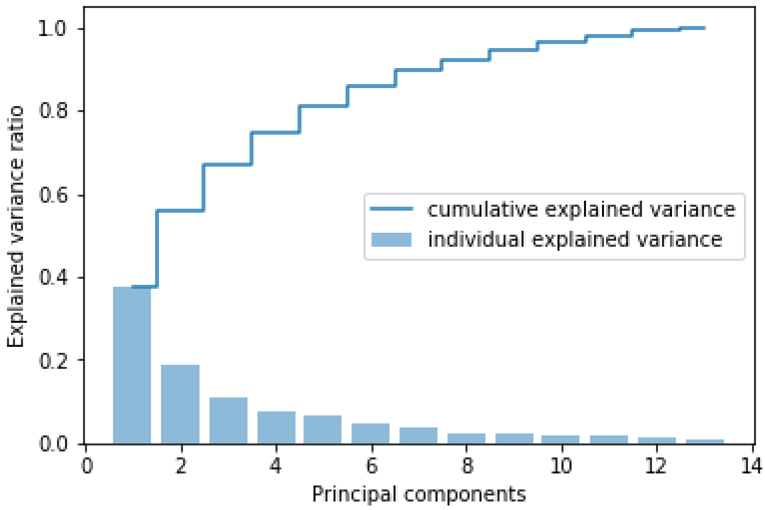
```
tot = sum(eigen_vals)
var_exp = [(i/tot) for i in sorted(eigen_vals, reverse=True)]
cum_var_exp = np.cumsum(var_exp)
```

[16]

```
import matplotlib.pyplot as plt
%matplotlib inline
```

[17]

```
plt.bar(range(1,14), var_exp, alpha=0.5, align='center',
       label='individual explained variance')
plt.step(range(1,14), cum_var_exp, where='mid',
        label='cumulative explained variance')
plt.ylabel('Explained variance ratio')
plt.xlabel('Principal components')
plt.legend(loc='best')
plt.show()
```



Feature Transformation

[18]

```
eigen_pairs = [(np.abs(eigen_vals[i]), eigen_vecs[:,i]) for i in range(le
eigen_pairs.sort(reverse=True)
```

[19]

```
w = np.hstack((eigen_pairs[0][1][:,np.newaxis],eigen_pairs[1][1][:, np.ne
```

[20]

```
print('Matrix W: {}'.format(w))
```

```
Matrix W: [[ 0.14669811  0.50417079]
 [-0.24224554  0.24216889]
```

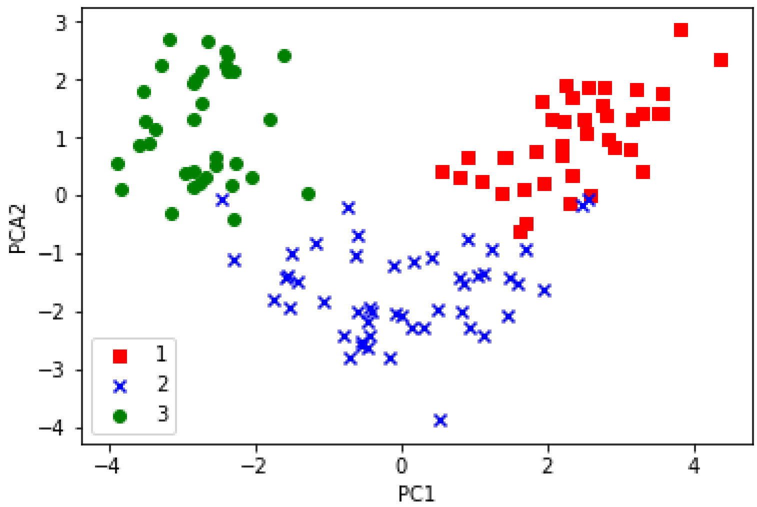
```
 [-0.02993442  0.28698484]
 [-0.25519002 -0.06468718]
 [ 0.12079772  0.22995385]
 [ 0.38934455  0.09363991]
 [ 0.42326486  0.01088622]
 [-0.30634956  0.01870216]
 [ 0.30572219  0.03040352]
 [-0.09869191  0.54527081]
 [ 0.30032535 -0.27924322]
 [ 0.36821154 -0.174365  ]
 [ 0.29259713  0.36315461]]
```

[21]
```
X_train_std[0].dot(w)
```

```
array([ 2.59891628,  0.00484089])
```

[22]
```
X_train_pca = X_train_std @ w
```

[25]
```
colors=['r', 'b', 'g']
markers = ['s', 'x', 'o']
for l,c,m in zip(np.unique(y_train), colors, markers):
    plt.scatter(X_train_pca[y_train==l,0],
                X_train_pca[y_train==l,1],
                c=c, label=l, marker=m)
plt.xlabel('PC1')
plt.ylabel('PCA2')
plt.legend(loc='lower left')
plt.show()
```



PCA in sklearn

[ ]
```
from matplotlib.colors import ListedColormap
```

```python
[37] def plot_decision_regions(X, y, classifier, test_idx=None, resolution=0.0

    # setup marker generator and color map
    markers = ('s', 'x', 'o', '^', 'v')
    colors = ('red', 'blue', 'lightgreen', 'gray', 'cyan')
    cmap = ListedColormap(colors[:len(np.unique(y))])

    # plot the decision surface
    x1_min, x1_max = X[:, 0].min() - 1, X[:, 0].max() + 1
    x2_min, x2_max = X[:, 1].min() - 1, X[:, 1].max() + 1
    xx1, xx2 = np.meshgrid(np.arange(x1_min, x1_max, resolution),
                           np.arange(x2_min, x2_max, resolution))
    Z = classifier.predict(np.array([xx1.ravel(), xx2.ravel()]).T)
    Z = Z.reshape(xx1.shape)
    plt.contourf(xx1, xx2, Z, alpha=0.4, cmap=cmap)
    plt.xlim(xx1.min(), xx1.max())
    plt.ylim(xx2.min(), xx2.max())

    for idx, cl in enumerate(np.unique(y)):
        plt.scatter(x=X[y == cl, 0],
                    y=X[y == cl, 1],
                    alpha=0.6,
                    c=cmap(idx),
                    edgecolor='black',
                    marker=markers[idx],
                    label=cl)

    # highlight test samples
    if test_idx:
        # plot all samples
        if not versiontuple(np.__version__) >= versiontuple('1.9.0'):
            X_test, y_test = X[list(test_idx), :], y[list(test_idx)]
            warnings.warn('Please update to NumPy 1.9.0 or newer')
        else:
            X_test, y_test = X[test_idx, :], y[test_idx]

        plt.scatter(X_test[:, 0],
                    X_test[:, 1],
                    c='',
                    alpha=1.0,
                    edgecolor='black',
                    linewidths=1,
                    marker='o',
                    s=55, label='test set')
```

```python
[36] from sklearn.linear_model import LogisticRegression
from sklearn.decomposition import PCA
```

```python
pca = PCA(n_components=2)
```

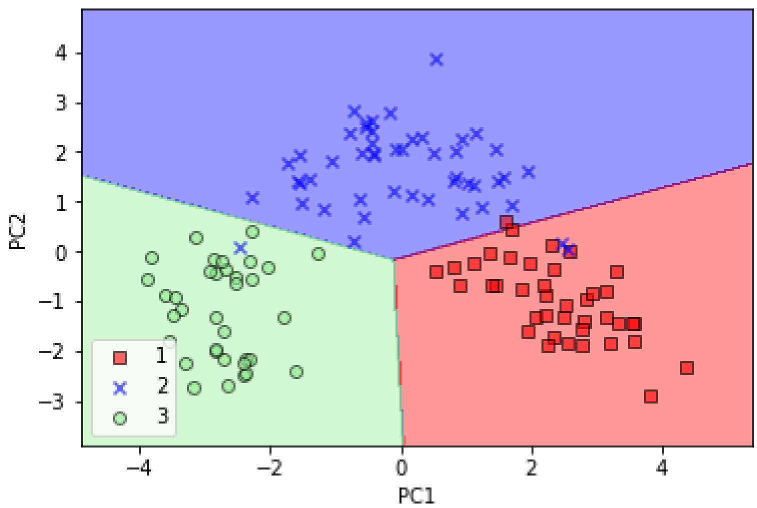
```
[30]    lr = LogisticRegression()
```

```
[31]    X_train_pca = pca.fit_transform(X_train_std)
        X_test_pca = pca.transform(X_test_std)
```

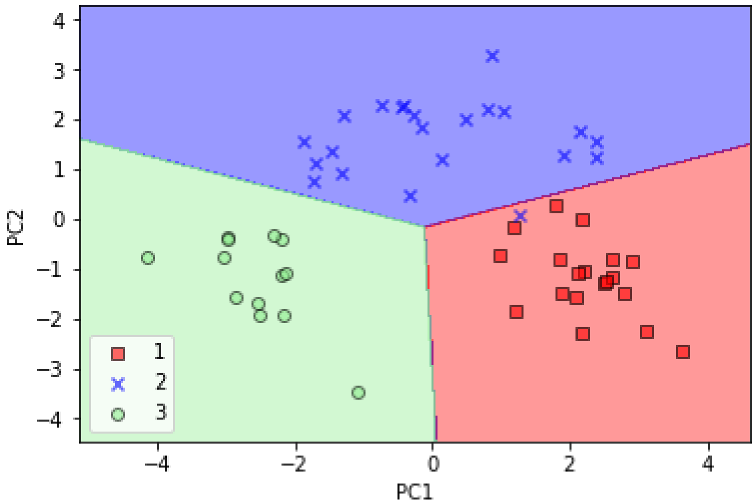
```
[32]    lr.fit(X_train_pca, y_train)
```

```
LogisticRegression(C=1.0, class_weight=None, dual=False,
fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=1,
          penalty='l2', random_state=None, solver='liblinear', tol=0.0001,
          verbose=0, warm_start=False)
```

```
[33]    plot_decision_regions(X_train_pca, y_train, classifier=lr)
        plt.xlabel('PC1')
        plt.ylabel('PC2')
        plt.legend(loc='lower left')
        plt.show()
```




```
[35]    plot_decision_regions(X_test_pca, y_test, classifier=lr)
        plt.xlabel('PC1')
        plt.ylabel('PC2')
        plt.legend(loc='lower left')
        plt.show()
```



4
3
2
1
0
−1
−2
−3
−4
PC2
−4
−2
0
2
4
PC1
1
2
3

[ ]