Machine Learning Workshop @ RGIT, Dec 2019

Instructor: Santosh Chapaneri

In [0]:
```
import numpy as np
import pandas as pd
import sklearn
import sklearn.linear_model as lm
import sklearn.model_selection as cv
import matplotlib.pyplot as plt
%matplotlib inline

from sklearn.decomposition import PCA
from sklearn.decomposition import KernelPCA
```

In [0]:
```
from google.colab import files
uploaded = files.upload()
```

Choose Files No file chosen

Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

```
Saving wine.data to wine (1).data
```

In [0]:
```
import io
wine = pd.read_csv(io.BytesIO(uploaded['wine.data']), header=None)
```

In [0]:
```
wine.columns = ['Class label', 'Alcohol', 'Malic acid', 'Ash',
                   'Alcalinity of ash', 'Magnesium', 'Total phenols',
                   'Flavanoids', 'Nonflavanoid phenols', 'Proanthocyanins',
                   'Color intensity', 'Hue', 'OD280/OD315 of diluted wines',
                   'Proline']

wine.head()

# In this dataset, there are 3 possible class labels
```

Out[0]:

	Class label	Alcohol	Malic acid	Ash	Alcalinity of ash	Magnesium	Total phenols	Flavanoids	Nonflavanoid phenols	Proanthocyanins	C inter
0	1	14.23	1.71	2.43	15.6	127	2.80	3.06	0.28	2.29	5.64
1	1	13.20	1.78	2.14	11.2	100	2.65	2.76	0.26	1.28	4.38
2	1	13.16	2.36	2.67	18.6	101	2.80	3.24	0.30	2.81	5.68
3	1	14.37	1.95	2.50	16.8	113	3.85	3.49	0.24	2.18	7.80
4	1	13.24	2.59	2.87	21.0	118	2.80	2.69	0.39	1.82	4.32

In [0]:
```
wine.shape
```

Out[0]:
```
(178, 14)
```

In [0]:
```
wine_val = wine.values
wine_val.shape
```

Out[0]:
```
(178, 14)
```

In [0]:
```
np.unique(wine_val[:,0])
```

Out[0]:
```
array([1., 2., 3.])
```

In [0]:
```
X, y = wine_val[:,1:], wine_val[:,0]
```

```
In [0]: # Split the data into 70% training and 30% test subsets
        import sklearn.model_selection as cv
        X_train, X_test, y_train, y_test = cv.train_test_split(X, y,
                                                               test_size=0.3,
                                                               stratify=y,
                                                               random_state=0)
```

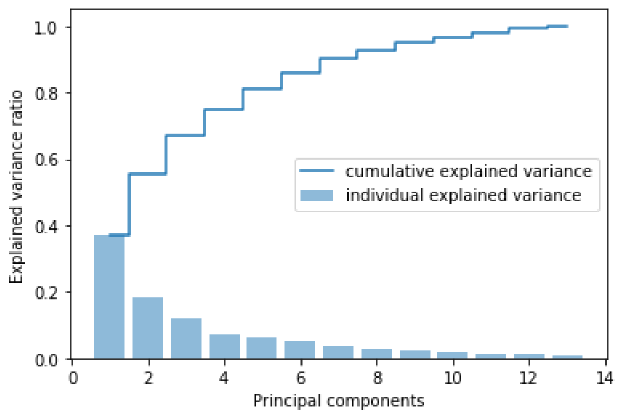
```
In [0]: # Standardize the data using StandardScaler
        # obtain X_train_std and X_test_std

        from sklearn.preprocessing import StandardScaler

        sc = StandardScaler()
        X_train_std = sc.fit_transform(X_train)

        X_test_std = sc.transform(X_test)
```

```
In [0]: print(X_train[:5,:5])
        print(X_train_std[:5,:5])
```

```
[[ 13.62   4.95   2.35  20.    92.  ]
 [ 13.76   1.53   2.7   19.5  132.  ]
 [ 13.73   1.5    2.7   22.5  101.  ]
 [ 13.51   1.8    2.65  19.   110.  ]
 [ 12.6    2.46   2.2   18.5   94.  ]]
[[ 0.71225893  2.22048673 -0.13025864  0.05962872 -0.50432733]
 [ 0.88229214 -0.70457155  1.17533605 -0.09065504  2.34147876]
 [ 0.84585645 -0.73022996  1.17533605  0.81104754  0.13597904]
 [ 0.57866141 -0.4736459   0.98882252 -0.24093881  0.77628541]
 [-0.52655446  0.09083903 -0.68979922 -0.39122257 -0.36203702]]
```

```
In [0]: from sklearn.decomposition import PCA
        import matplotlib.pyplot as plt
        import numpy as np

        pca = PCA()
        X_train_pca = pca.fit_transform(X_train_std)
        print(pca.explained_variance_ratio_)

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

        plt.show()
```

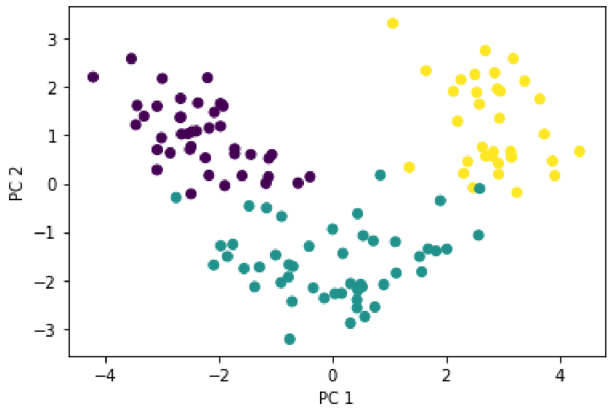
```
[0.36951469 0.18434927 0.11815159 0.07334252 0.06422108 0.05051724
 0.03954654 0.02643918 0.02389319 0.01629614 0.01380021 0.01172226
 0.00820609]
```



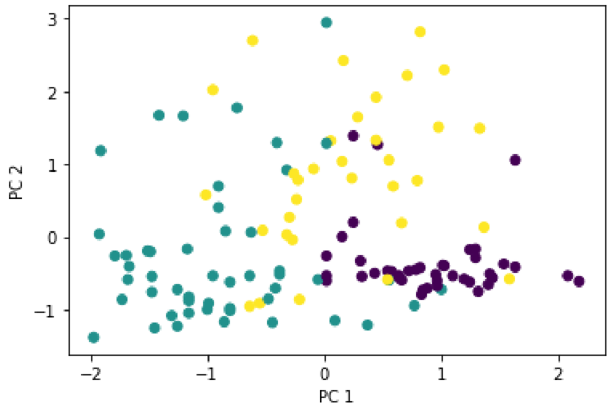
```
In [0]: # Apply PCA (2 components) on X_train_std
        pca = PCA(n_components=2)

        X_train_pca = pca.fit_transform(X_train_std)
```

```
In [0]: # Transform X_test_std using PCA
        X_test_pca = pca.transform(X_test_std)
```

```
In [0]: # Do scatter plot of the transformed data
        plt.scatter(X_train_pca[:, 0], X_train_pca[:, 1], c=y_train)
        plt.xlabel('PC 1')
        plt.ylabel('PC 2')
        plt.show()
```



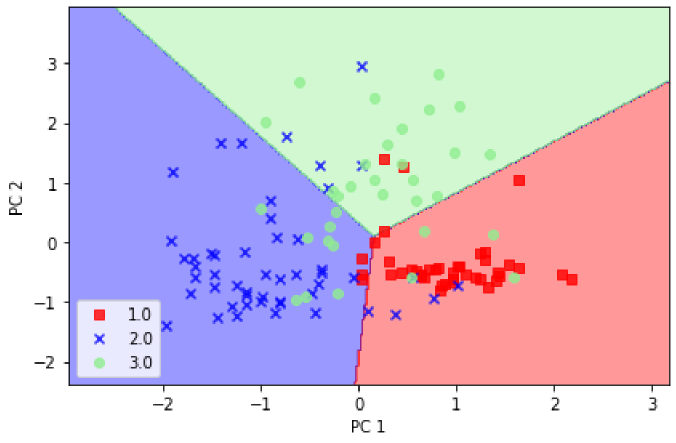
```
In [0]: # Do scatter plot of the transformed data
        plt.scatter(X_train_std[:, 0], X_train_std[:, 1], c=y_train)
        plt.xlabel('PC 1')
        plt.ylabel('PC 2')
        plt.show()
```



In [0]:
```
from matplotlib.colors import ListedColormap
def plot_decision_regions(X, y, classifier, test_idx=None, resolution=0.02):

    # 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=0.8, c=cmap(idx),
                    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,
                    linewidths=1,
                    marker='o',
                    s=55, label='test set')
```

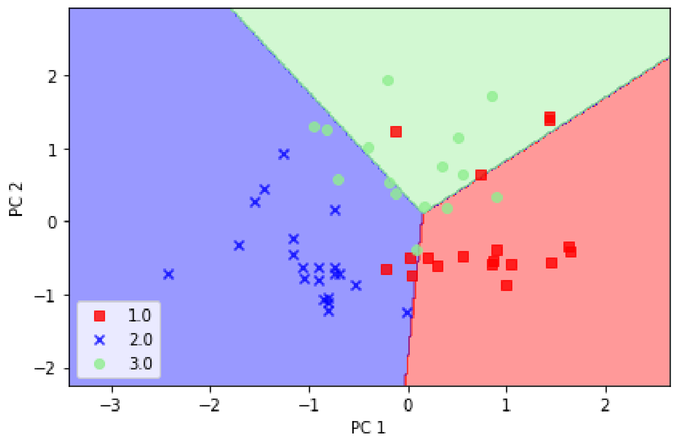
In [0]:
```
# Train Logistic Regression classifier without using PCA

from sklearn.linear_model import LogisticRegression

lr_no_pca = LogisticRegression(solver='lbfgs', multi_class='ovr')
lr_no_pca = lr_no_pca.fit(X_train_std[:,:2], y_train)

plot_decision_regions(X_train_std[:,:2], y_train, classifier=lr_no_pca)
plt.xlabel('PC 1')
plt.ylabel('PC 2')
plt.legend(loc='lower left')
plt.tight_layout()
plt.show()
```

```
'c' argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-ma
pping will have precedence in case its length matches with 'x' & 'y'.  Please use a 2-D array with
a single row if you really want to specify the same RGB or RGBA value for all points.
'c' argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-ma
pping will have precedence in case its length matches with 'x' & 'y'.  Please use a 2-D array with
a single row if you really want to specify the same RGB or RGBA value for all points.
'c' argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-ma
pping will have precedence in case its length matches with 'x' & 'y'.  Please use a 2-D array with
a single row if you really want to specify the same RGB or RGBA value for all points.
```



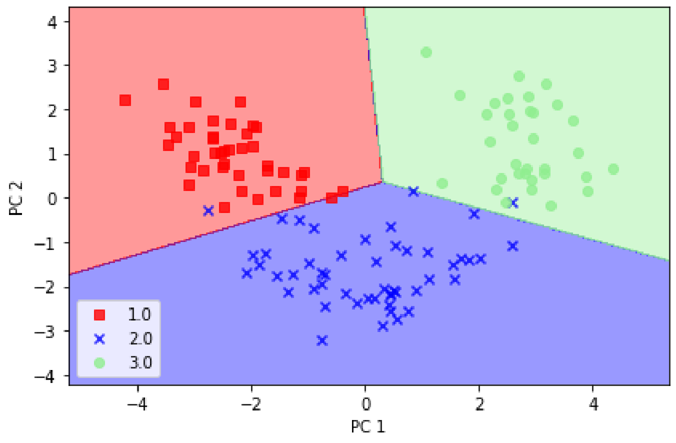
In [0]:
```python
# Test the model
plot_decision_regions(X_test_std[:,:2], y_test, classifier=lr_no_pca)

plt.xlabel('PC 1')
plt.ylabel('PC 2')
plt.legend(loc='lower left')
plt.tight_layout()
plt.show()
```

```
'c' argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-ma
pping will have precedence in case its length matches with 'x' & 'y'.  Please use a 2-D array with
a single row if you really want to specify the same RGB or RGBA value for all points.
'c' argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-ma
pping will have precedence in case its length matches with 'x' & 'y'.  Please use a 2-D array with
a single row if you really want to specify the same RGB or RGBA value for all points.
'c' argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-ma
pping will have precedence in case its length matches with 'x' & 'y'.  Please use a 2-D array with
a single row if you really want to specify the same RGB or RGBA value for all points.
```



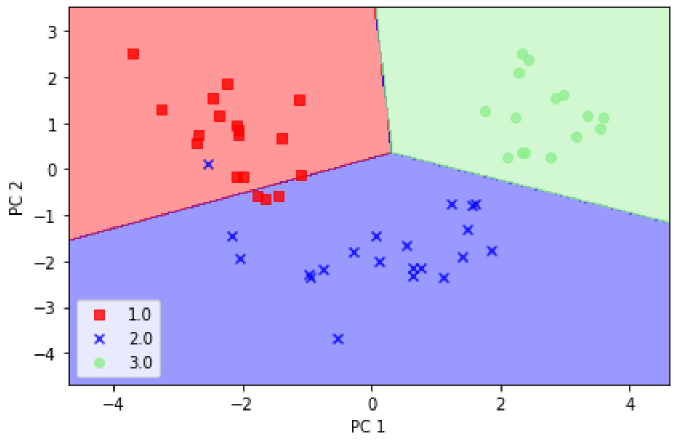
In [0]:
```python
# Train Logistic Regression classifier using the first 2 principal components

from sklearn.linear_model import LogisticRegression

lr_pca = LogisticRegression(solver='lbfgs', multi_class='ovr')
lr_pca = lr_pca.fit(X_train_pca, y_train)

plot_decision_regions(X_train_pca, y_train, classifier=lr_pca)
plt.xlabel('PC 1')
plt.ylabel('PC 2')
plt.legend(loc='lower left')
plt.tight_layout()
plt.show()
```

```
'c' argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-ma
pping will have precedence in case its length matches with 'x' & 'y'.  Please use a 2-D array with
a single row if you really want to specify the same RGB or RGBA value for all points.
'c' argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-ma
pping will have precedence in case its length matches with 'x' & 'y'.  Please use a 2-D array with
a single row if you really want to specify the same RGB or RGBA value for all points.
'c' argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-ma
pping will have precedence in case its length matches with 'x' & 'y'.  Please use a 2-D array with
a single row if you really want to specify the same RGB or RGBA value for all points.
```



In [0]:
```
# Test the model
plot_decision_regions(X_test_pca, y_test, classifier=lr_pca)

plt.xlabel('PC 1')
plt.ylabel('PC 2')
plt.legend(loc='lower left')
plt.tight_layout()
plt.show()
```

```
'c' argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-ma
pping will have precedence in case its length matches with 'x' & 'y'.  Please use a 2-D array with
a single row if you really want to specify the same RGB or RGBA value for all points.
'c' argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-ma
pping will have precedence in case its length matches with 'x' & 'y'.  Please use a 2-D array with
a single row if you really want to specify the same RGB or RGBA value for all points.
'c' argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-ma
pping will have precedence in case its length matches with 'x' & 'y'.  Please use a 2-D array with
a single row if you really want to specify the same RGB or RGBA value for all points.
```


PC 2
PC 1