data_reduction

March 9, 2020

1 Samuel Bush

1.1 CS 6470

[1]:
```python
import pandas as pd
df_wine = pd.read_csv('https://archive.ics.uci.edu/ml/'
                      'machine-learning-databases/wine/wine.data',
                      header=None)
print(df_wine)
```

```
     0      1     2     3     4    5     6     7     8     9     10    11  \
0    1  14.23  1.71  2.43  15.6  127  2.80  3.06  0.28  2.29   5.64  1.04
1    1  13.20  1.78  2.14  11.2  100  2.65  2.76  0.26  1.28   4.38  1.05
2    1  13.16  2.36  2.67  18.6  101  2.80  3.24  0.30  2.81   5.68  1.03
3    1  14.37  1.95  2.50  16.8  113  3.85  3.49  0.24  2.18   7.80  0.86
4    1  13.24  2.59  2.87  21.0  118  2.80  2.69  0.39  1.82   4.32  1.04
..  ..    …     …     …    …    …    …     …     …     …     …     …
173  3  13.71  5.65  2.45  20.5   95  1.68  0.61  0.52  1.06   7.70  0.64
174  3  13.40  3.91  2.48  23.0  102  1.80  0.75  0.43  1.41   7.30  0.70
175  3  13.27  4.28  2.26  20.0  120  1.59  0.69  0.43  1.35  10.20  0.59
176  3  13.17  2.59  2.37  20.0  120  1.65  0.68  0.53  1.46   9.30  0.60
177  3  14.13  4.10  2.74  24.5   96  2.05  0.76  0.56  1.35   9.20  0.61

       12    13
0    3.92  1065
1    3.40  1050
2    3.17  1185
3    3.45  1480
4    2.93   735
..    …     …
173  1.74   740
174  1.56   750
175  1.56   835
176  1.62   840
177  1.60   560

[178 rows x 14 columns]
```

[2]:

```python
from sklearn.model_selection import train_test_split
X, y = df_wine.iloc[:, 1:].values, df_wine.iloc[:, 0].values
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.
 ↪3,stratify=y,random_state=0)

from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train_std = sc.fit_transform(X_train)
X_test_std = sc.transform(X_test)
```

[3]:

```python
import numpy as np
cov_mat = np.cov(X_train_std.T)
eigen_vals, eigen_vecs = np.linalg.eig(cov_mat)
print('\nEigenvalues \n%s' % eigen_vals)
```

```
Eigenvalues
[4.84274532 2.41602459 1.54845825 0.96120438 0.84166161 0.6620634
 0.51828472 0.34650377 0.3131368  0.10754642 0.21357215 0.15362835
 0.1808613 ]
```

[4]:

```python
tot = sum(eigen_vals)
var_exp = [(i / tot) for i in sorted(eigen_vals, reverse=True)]
cum_var_exp = np.cumsum(var_exp)
import matplotlib.pyplot as plt
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 component index')
plt.legend(loc='best')
plt.tight_layout()
plt.show()
```

```
<Figure size 640x480 with 1 Axes>
```

2 What this plot is telling us is that the first principal component accounts for close to 40% of the varience. When you combind the first and second component that accounts for almost 60% of the varience and so fourth.

[5]:
```
eigen_pairs = [(np.abs(eigen_vals[i]), eigen_vecs[:, i])for i in␣
 ↪range(len(eigen_vals))]

eigen_pairs.sort(key=lambda k: k[0], reverse=True)
```

[6]:
```
w = np.hstack((eigen_pairs[0][1][:, np.newaxis],eigen_pairs[1][1][:, np.
 ↪newaxis]))
print('Matrix W:\n', w)
```

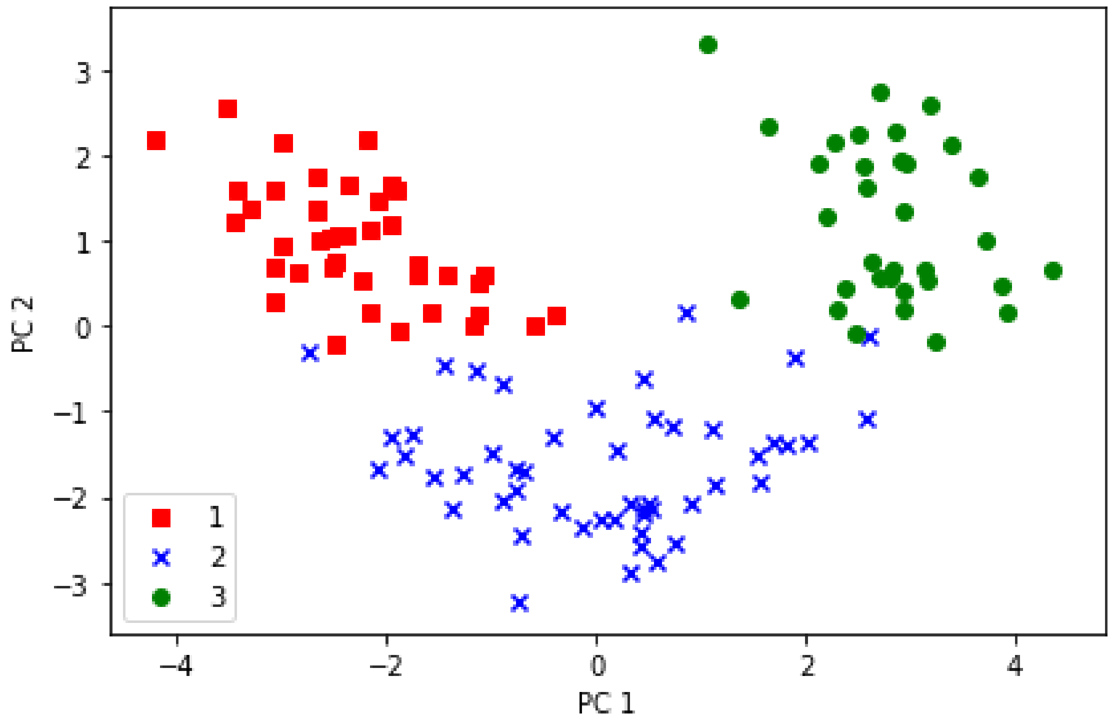
```
Matrix W:
 [[-0.13724218  0.50303478]
 [ 0.24724326  0.16487119]
 [-0.02545159  0.24456476]
 [ 0.20694508 -0.11352904]
 [-0.15436582  0.28974518]
 [-0.39376952  0.05080104]
 [-0.41735106 -0.02287338]
 [ 0.30572896  0.09048885]
 [-0.30668347  0.00835233]
 [ 0.07554066  0.54977581]
 [-0.32613263 -0.20716433]
 [-0.36861022 -0.24902536]
 [-0.29669651  0.38022942]]
```

[7]:
```
X_train_std[0].dot(w)
```

[7]:
```
array([2.38299011, 0.45458499])
```

[8]:
```
X_train_pca = X_train_std.dot(w)
```

[9]:
```
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('PC 1')
plt.ylabel('PC 2')
plt.legend(loc='lower left')
plt.tight_layout()
plt.show()
```

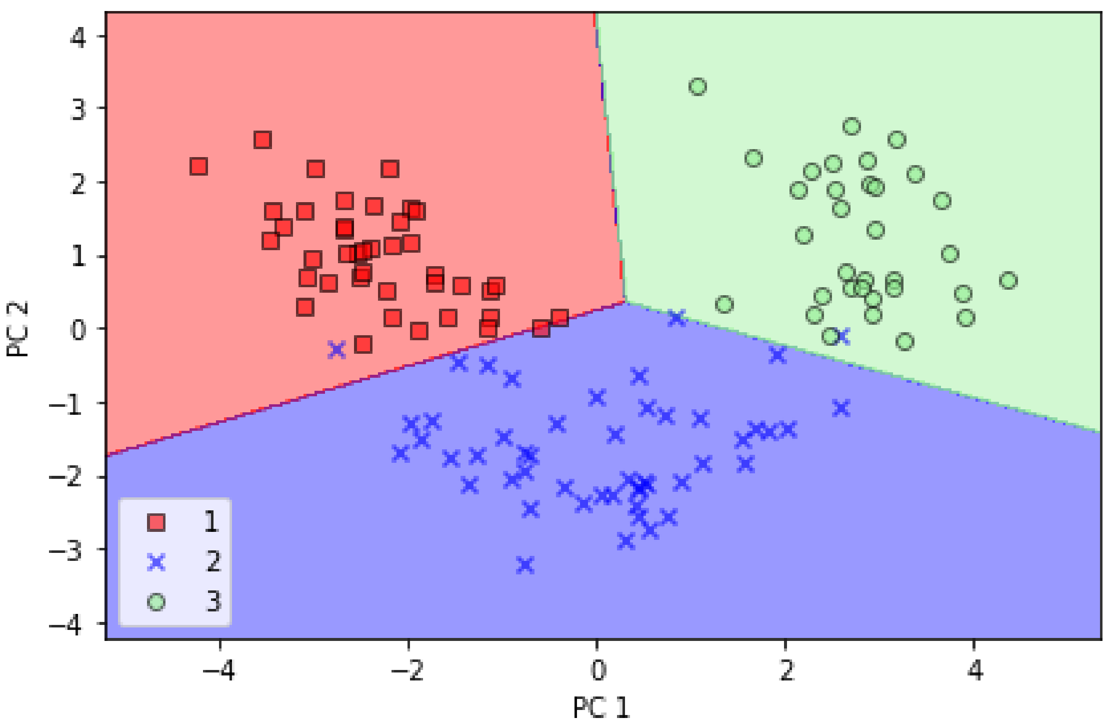


3 What we can infer from this graph is that the data is more spread along the x axis, and that a linear classifier would not work on this data.

[10]:
```
from matplotlib.colors import ListedColormap
def plot_decision_regions(X, y, classifier, resolution=0.02):
    markers = ('s', 'x', 'o', '^', 'v')
    colors = ('red', 'blue', 'lightgreen', 'gray', 'cyan')
    cmap = ListedColormap(colors[:len(np.unique(y))])
    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,
```

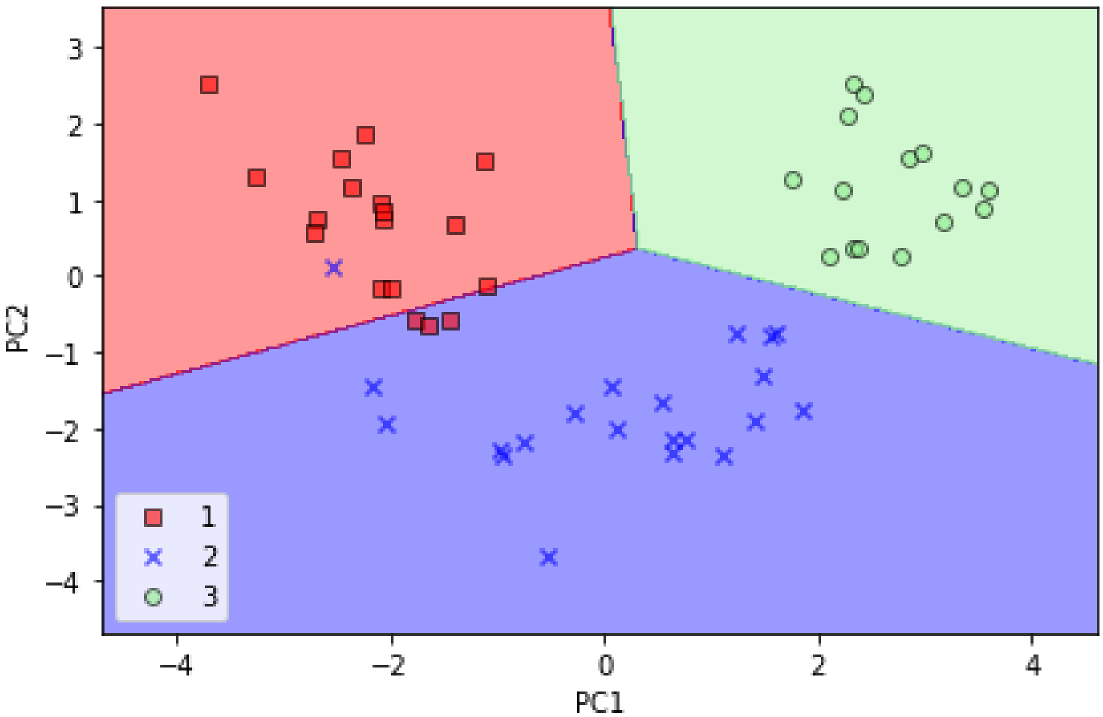
```
                    color=cmap(idx),
                    edgecolor='black',
                    marker=markers[idx],
                    label=cl)
```

[11]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.decomposition import PCA
pca = PCA(n_components=2)
lr = LogisticRegression(multi_class='ovr',random_state=1,solver='lbfgs')
X_train_pca = pca.fit_transform(X_train_std)
X_test_pca = pca.transform(X_test_std)
lr.fit(X_train_pca, y_train)
plot_decision_regions(X_train_pca, y_train, classifier=lr)
plt.xlabel('PC 1')
plt.ylabel('PC 2')
plt.legend(loc='lower left')
plt.tight_layout()
plt.show()
```



4 This shows the decision regions of the training data reduced to two principle componant axes.

[12]:
```
plot_decision_regions(X_test_pca, y_test, classifier=lr)
plt.xlabel('PC1')
plt.ylabel('PC2')
plt.legend(loc='lower left')
plt.tight_layout()
plt.show()
```



5 What this plot is telling us is that logistic regression works well on two dimensional feature subspace and only has a few misclassifications.

[13]:
```
pca = PCA(n_components=None)
X_train_pca = pca.fit_transform(X_train_std)
pca.explained_variance_ratio_
```

[13]:
```
array([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])
```