ML_HW1_Report

107081028 常安彥

1. Train Test Split

Using the judgement of the label column, I could separate the wine data into three data frames with different labels. Then, train test split on each of the label respectively, split 18 instances into the testing set (**fig**1-1). Finally, concatenate the three training data frames into one, and the three testing data frames in another. Save them as csv file (**fig**1-2), in this way, the work required is done.

(**fig**. 1-1) Separate and train test split

```
### train test split label 1 data, label 2 data, label 3 data respectively
train_1, test_1 = train_test_split(data[data[0] == 1], test_size=18, random_state=10)
train_2, test_2 = train_test_split(data[data[0] == 2], test_size=18, random_state=10)
train_3, test_3 = train_test_split(data[data[0] == 3], test_size=18, random_state=10)
```

(**fig**. 1-2) Concatenate and save

```
### concatenate label 1, 2 ,3 data and save as csv
train_data = pd.concat([train_1, train_2, train_3], axis = 0)
test_data = pd.concat([test_1, test_2, test_3], axis = 0)
train_data.to_csv("train.csv",index=False ,header=None)
test_data.to_csv("test.csv",index=False, header=None)
```

2. Posterior Probabilities

First, calculate the mean and standard deviation of each feature in respective of the labels (**fig**2-1). Then, calculate the prior probabilities of each class (**fig**2-2).

(**fig**. 2-1) Calculate the statistic numbers with data frame function

```
col1_mean = train_1.mean(axis = 0)
col1_std = train_1.std(axis = 0)
col2_mean = train_2.mean(axis = 0)
col2_std = train_2.std(axis = 0)
col3_mean = train_3.mean(axis = 0)
col3_std = train_3.std(axis = 0)
```

(**fig**. 2-2) Calculate prior probabilities with the proportion

```
### Calculate the prior probabilities of 3 labels
prior_1 = len(train_1) / len(train_data)
prior_2 = len(train_2) / len(train_data)
prior_3 = len(train_3) / len(train_data)
```

Next, for each instance in the testing data, bring it in the likelihood function. Which means that, calculate the product of prior probability and the Gaussian probabilities of all features in respective of the three labels (**fig**2-3).

(**fig**. 2-3) Calculate posterior probability

```
### Calculate the posterior probabilities of each testing instance
###    by timimg up the prior probability and likelihood function of each column
### Take the greatest among three labels with argmax function as the prediction
correct_prediction = 0
for i in range(len(test_data)):
    label1_prob = prior_1
    label2_prob = prior_2
    label3_prob = prior_3
    for j in range(1, 14):    ### column 0 is the label, so shall not be included
        label1_prob *= GaussianProbability(col1_mean[j], col1_std[j], test_data.iloc[i][j].item())
        label2_prob *= GaussianProbability(col2_mean[j], col2_std[j], test_data.iloc[i][j].item())
        label3_prob *= GaussianProbability(col3_mean[j], col3_std[j], test_data.iloc[i][j].item())
```

Finally, use argmax function to obtain the prediction and compare to the ground truth, evaluating the accuracy of own function (fig2-4).

(**fig**. 2-4) Evaluation

```
    if test_data.iloc[i][0].item() == (np.argmax(labels_prob) + 1):
        correct_prediction += 1

### Evaluate the accuracy
print("Accuracy: " + str(100.0 * correct_prediction / len(test_data)))
```

Result: 96.296% acc (exceed 90% as required)

```
In [80]: runcell(2, 'D:/四下_實/ML/HW1/HW1/untitled0.py')
Accuracy: 96.29629629629629
```

3. Visualized Result

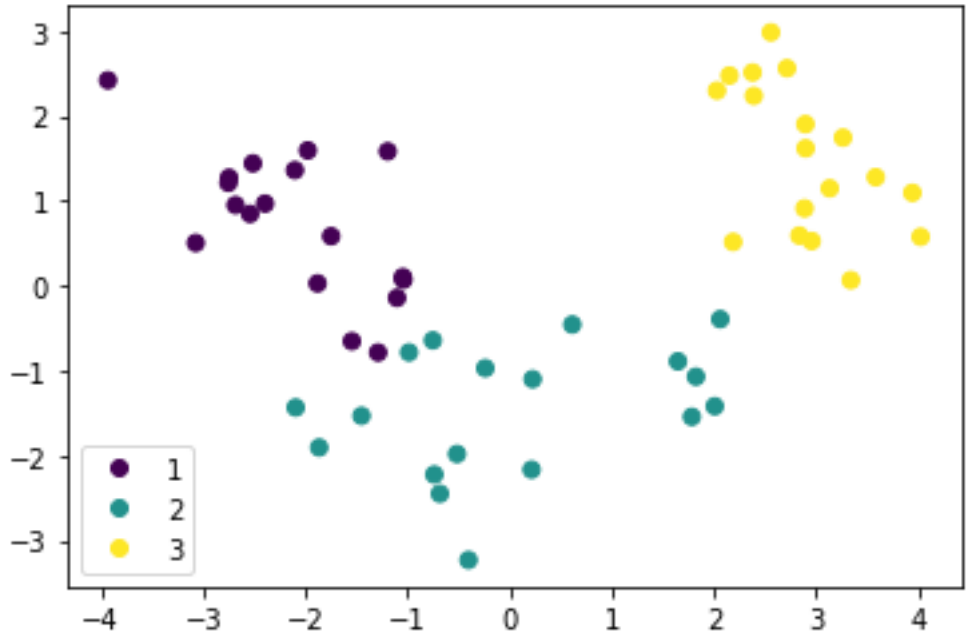
I implemented PCA function, fitting on training set and transforming on testing data set (**fig**3-1). According to the result(**fig**3-2), we can see that the testing instances are clearly classified by the two most significant components fit by the training set.

(**fig**. 3-1) PCA function with Standard Scalar

```
### Fit pca with the training set and transform on the testing set
test_data_nolabel = test_data.iloc[:, 1:]
test_data_label = test_data.iloc[:, 0]
train_data_nolabel = train_data.iloc[:, 1:]
pipe = Pipeline([('scaler', StandardScaler()), ('pca', PCA())])
_ = pipe.fit(train_data_nolabel)
X_pca = pipe.transform(test_data_nolabel)

### Plot the first two components on 2-d scatter plot
plot = plt.scatter(X_pca[:,0], X_pca[:,1], c=test_data_label)
plt.legend(handles=plot.legend_elements()[0], labels=list((1,2,3)))
plt.show()
```

(**fig**. 3-2) PCA Scatter Plot



4. Effect of Prior Distribution

I agree to the effect of prior distribution on the posterior probabilities, nevertheless, it is not significant in above implementation. I assume the reason is that there are too many features included in this wine example, the effect of prior distribution decreases as the number of features increases. Furthermore, the more imbalanced the data set is, the more significantly the prior probability is capable to affect the results.

To prove my assumption, I discarded some of the features in two ways:

a. Effect of Prior Probability with Random Features

I randomly shuffled the indices of the features and reserve the features that come up first (**fig**4-1).

(**fig**. 4-1) Randomly Shuffle Indices & Only use the first two of them

```
import random
l = list(i for i in range(1, 14))
random.shuffle(l)
```

```
for j in l[:2]:
    label1_prob *= GaussianProbability(col1_mean[j], col1_std[j], test_data.iloc[i][j].item())
    label2_prob *= GaussianProbability(col2_mean[j], col2_std[j], test_data.iloc[i][j].item())
    label3_prob *= GaussianProbability(col3_mean[j], col3_std[j], test_data.iloc[i][j].item())
    label1_prob_without_prior *= GaussianProbability(col1_mean[j], col1_std[j], test_data.iloc[i][j].item())
    label2_prob_without_prior *= GaussianProbability(col2_mean[j], col2_std[j], test_data.iloc[i][j].item())
    label3_prob_without_prior *= GaussianProbability(col3_mean[j], col3_std[j], test_data.iloc[i][j].item())
```

Also, as seen above, I define another variable as the likelihood without prior probabilities, and evaluate both situations.

Results:

```
----Effect of Prior Probability with Random Features----
Accuracy with prior: 64.81481481481481
Accuracy without prior: 61.111111111111114
Accuracy with prior: 70.37037037037037
Accuracy without prior: 68.51851851851852
```

Two results are shown here in the report, showing that the accuracy increased due to the prior probability.

b. Effect of Prior Probability with PCA

I decrease the dimension of features with the PCA function, and reserve the most significant component. Later on, calculate the statistic numbers again with the new PCA-transformed data set (**fig**4-2).

(**fig**. 4-2) PCA fit transforming data & new statistic numbers

```
### Fit PCA on training set and transform on testing set
### Calculate the new mean and std
train_data_nolabel = train_data.iloc[:, 1:]
pipe = Pipeline([('scaler', StandardScaler()), ('pca', PCA(n_components=1))])
_ = pipe.fit_transform(train_data_nolabel)
test_pca = pipe.transform(test_data_nolabel)
col1_mean_pca = _[train_data[0] == 1].mean(axis = 0)
col1_std_pca = _[train_data[0] == 1].std(axis = 0)
col2_mean_pca = _[train_data[0] == 2].mean(axis = 0)
col2_std_pca = _[train_data[0] == 2].std(axis = 0)
col3_mean_pca = _[train_data[0] == 3].mean(axis = 0)
col3_std_pca = _[train_data[0] == 3].std(axis = 0)
```

Similarly, calculate two cases, with and without prior probabilities, and evaluate both situations. But, this time, implement the function on PCA-transformed data.

Result:

```
----Effect of Prior Probability with PCA----
Accuracy with prior: 81.48148148148148
Accuracy without prior: 79.62962962962963
```

The result showed that the prior probability still affect the posterior probability.