Part 2. Programming assignment

(2.)

Result:

```
y_test: [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2]
predict: [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1
 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2]
Predicted classes: [0 1 2]
Accuracy: 98.333333%
```

y_test → data of test which is no label

Illustrate:

```
# classes classification type (0,1,2) ; counts records the number of this type
classes, counts = np.unique(y_train, return_counts = True )

# prior probability = counts of type / total trainimg data
priors = counts / len(y_train)
```

Calculate prior probability

```
# Calculate mean and variance for each feature for each class

num_classes = len(classes)
means = np.zeros((num_classes, x_train.shape[1]))
variances = np.zeros((num_classes, x_train.shape[1]))

for i, c in enumerate(classes):
    x_c = x_train[y_train == c]
    means[i, :] = x_c.mean(axis=0)
    variances[i, :] = x_c.var(axis=0)
```

Calculate the number of variations and average for each label

```python
# define the posterier function
def posterior_function(x) :
    posterior = np.array([]) #create the null space of posterior function
    for i in range(num_classes):
        log_prior = np.log(priors[i])
        gauss = stats.norm.pdf(x,loc=means[i],scale=np.sqrt(variances))
        log_likehood = np.sum(np.log(gauss))
        A = log_likehood + log_prior
        posterior=np.append(posterior,A)

    return classes[np.argmax(posterior)]
```

Calculates the conditional probability density of data point 'x' for the 'I' category using the probability density function of the Gaussian (normal) distribution.

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

The idea of likelihood is to add up all gauss values and take the natural logarithm.

What more , the formula of posterior is "Posterior probability = prior probability + new evidence (called likelihood)."(https://unacademy.com/content/cat/study-material/data-interpretation-and-logical-reasoning/what-is-posterior-probability/)

Finally return the index of the maximum posterior probability.

```python
predict = np.array([posterior_function(i) for i in x_test])

predict = np.array(predict)
```

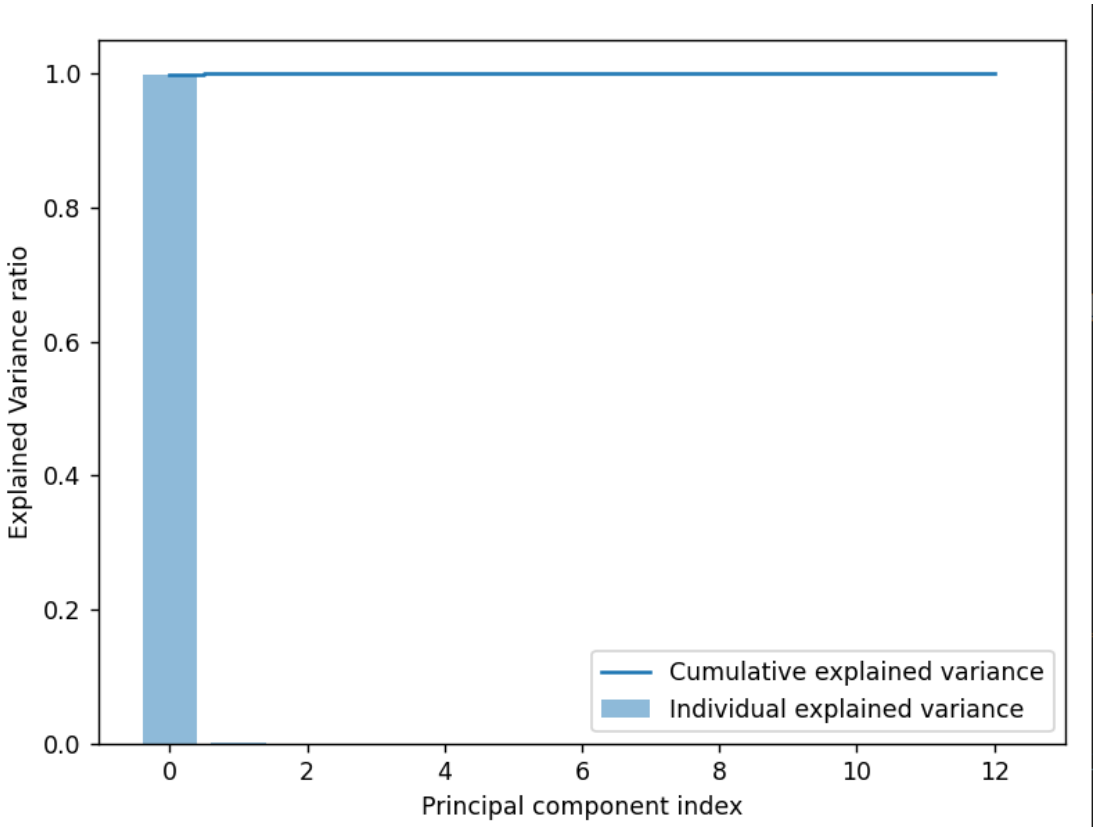
The test data into posterior_function() through loop call

```python
# Calculate Accuray

accuracy = np.mean (predict == y_test)
print(f'Accuracy: {accuracy * 100:f}%')
```

Calculate the accuracy rate by comparing Predict and test data

(3.)



Explained variance ratio is a measure of the proportion of the total variance in the original dataset that is explained by each principal component. The explained variance ratio of a principal component is equal to the ratio of its eigenvalue to the sum of the eigenvalues of all the principal components.

In Sklearn PCA, the explained variance ratio of each principal component can be accessed through the explained_variance_ratio_ attribute. For example, if pca is a Sklearn PCA object, pca.explained_variance_ratio_[i] gives the explained variance ratio of the i-th principal component.

(https://saturncloud.io/blog/what-is-sklearn-pca-explained-variance-and-explained-variance-ratio-difference/)

So my solution shows that index = 0 has the largest variance ratio, and index = 0 means "alcohol" in x_test. The explained variation measures the absolute amount of variation explained by each principal component. This is useful when you want to know how many principal components need to be retained to capture a certain percentage of the total variation in the original data set.

The explained variance ratio measures the relative amount of variance explained by each principal component. This is useful when you want to know how much information each principal component contributes to the overall structure of the data.

Code :

```
# plot the PCA visualized result of testing data

print(x_test)
pca = PCA(n_components=13) # each feature
newX = pca.fit_transform(x_test)
invX = pca.inverse_transform(newX)
#print(newX)
#print(invX)
#print(pca.explained_variance_)
#print(pca.explained_variance_ratio_)
#print(pca.explained_variance_ratio_.cumsum)
exp_var_pca = pca.explained_variance_ratio_
cum_sum_eigenvalues = np.cumsum(exp_var_pca)
plt.bar(range(0,len(exp_var_pca)),exp_var_pca, alpha = 0.5,
    align = 'center',label = 'Individual explained variance')
plt.step(range(0,len(cum_sum_eigenvalues)),cum_sum_eigenvalues,
    where = 'mid',label='Cumulative explained variance')
plt.xlabel('Principal component index')
plt.ylabel('Explained Variance ratio')
plt.legend(loc='best')
plt.tight_layout()
plt.show()
```

Reference:

https://blog.csdn.net/qq_20135597/article/details/95247381

https://vitalflux.com/pca-explained-variance-concept-python-example/

(4)

The formula mentioned earlier:

Posterior probability = prior probability + new evidence (called likelihood).

How will the prior probability be changed? That is, the label sampling is different from my original training. Judging from the formula, if the value is changed, it may cause the value of the posterior to change, which will affect the final accuracy.

You can see that I adjusted the proportion of each label in test.csv and train.csv, and the accuracy rate dropped. Then, looking at the confusion matrix, you can also see how many pieces of data were predicted incorrectly.

[Test1.csv →(wine 0: 15 ; wine1:30 : wine2:15)]

```
Accuracy: 95.000000%
```

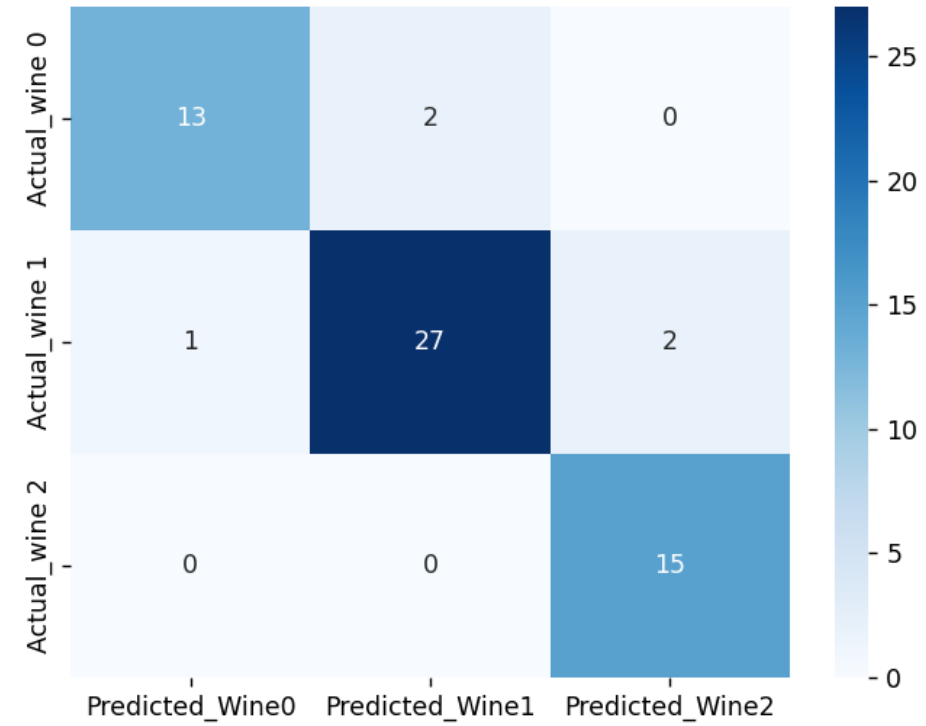
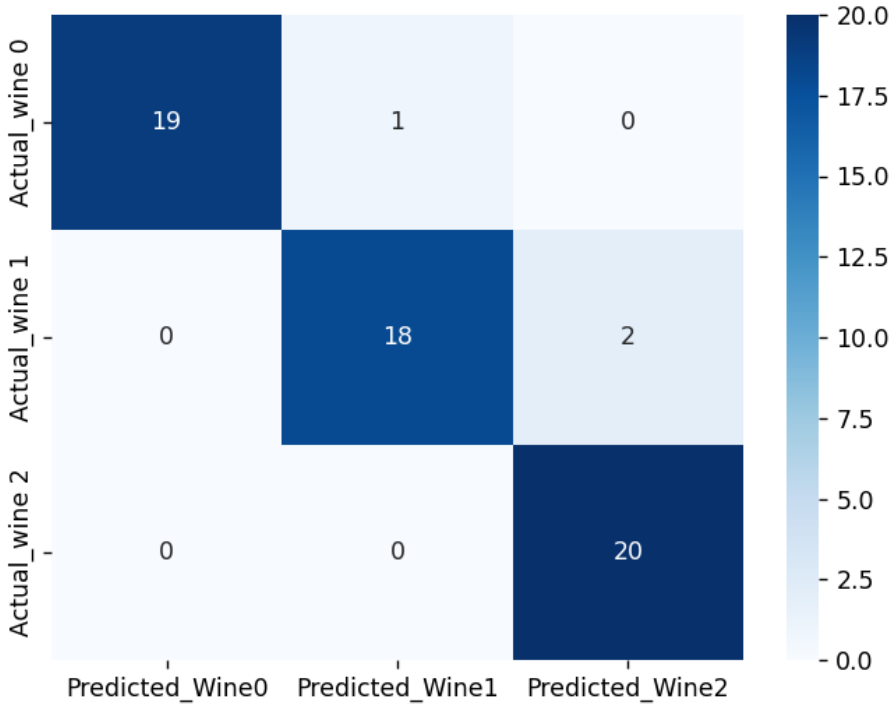


Fig.1 Test1.csv and using Train1.csv to train

(5.)



The confusion matrix can help us check the quality of the model in machine learning in the future, and can be extended to accuracy, precision, recall, F-score, etc. These metrics can help us comprehensively evaluate the performance of the model in different aspects.

From our results, we can see 3 prediction errors, and we can see the accuracy from the confusion matrix. I think that if we want to increase the accuracy, we should increase the number of training data or find a better module.

code:

```
# define plot function

def plot(y_true,y_pred):
    #labels = classes
    column = [f'Predicted_Wine{label}' for label in classes]
    indices = [f'Actual_wine {label}' for label in classes]
    table = pd.DataFrame(confusion_matrix(y_true,y_pred),columns = column , index = indices)

    return sns.heatmap(table,cmap='Blues',annot=True,fmt='d',cbar=True)

# confusion_matrix

matrix_array=confusion_matrix(y_test,predict)
print(matrix_array)
plot(y_test,predict)
plt.show()
```

Reference: https://www.youtube.com/watch?v=9w4HJ0VUy2g