Data cleaning

For part I, the dataset is one binary class classification and the data should be cleaned to remove the space, null, and irrelevant blanks. And loading the first 5 lines is a basic command to see the structure of dataset.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

# load .csv file as pandas dataframe
data = pd.read_csv("data_subset.csv")
# show first 5 rows
data.head(5)
```

Then the label corresponding to "normal" and "disease" should be picked up to deal with through using Boolean indexing.

```
data_binary = data[(data["label"]=="normal") |
(data["label"]=="disease")].copy()
data_binary.groupby("label").count() # using groupby() to see the number
of disease and normal of gene
```

then using numpy array to do slice of the specific area.

```
X = data_binary.iloc[:,2:-1]
X = X.to_numpy() # using X.shape() to see the structure of this specific
data area
```

Also do selection for the label area as well and using .shape() to check the result.

```
labels = data_binary.iloc[:,-1]
y = (labels=="disease").astype(int)
```

Spliting dataset

Then, create one pipeline to compute the result of each item after inserting the standard libraries. `cross_val_score` would be utilized to create one crossing validity result after 5 folders with scoring of 'f1', and finally compute the mean value.

```
from sklearn.model_selection import train_test_split
from sklearn.decomposition import PCA
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import plot_confusion_matrix
from sklearn.metrics import f1_score
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)
```

Then choose one supervised learning algorithm to compute the result. Basically, support-vector-machine(SVM) and Principal component analysis(PCA) are always used as most of them. Then the code could be seen as follows:

```
## Logistic Regression as supervised learning algorithm
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline
from sklearn.model_selection import cross_val_score

scaler = StandardScaler()
pca = PCA(n_components=20, random_state=42)
logreg = LogisticRegression()
pipe = Pipeline([('scaler', scaler), ('pca', pca), ('logreg', logreg)])
scores = cross_val_score(pipe, X_train, y_train, cv = 5, scoring='f1')
```

```
## Support Vector Machine would be as follows
from sklearn.model_selection import train_test_split
from sklearn.decomposition import PCA
from sklearn.svm import SVC
from sklearn.pipeline import Pipeline
from sklearn.metrics import plot_confusion_matrix
from sklearn.model_selection import GridSearchCV

scaler = StandardScaler()
pca = PCA(n_components=20, random_state=42)
svm = SVC()

pipe = Pipeline([('scaler', scaler), ('pca', pca), ('svm', svm)])
param_grid = {'svm__C': [0.01, 1, 100], 'svm__gamma': [1e-04, 1e-03, 1e-
02]}

search = GridSearchCV(pipe, param_grid,cv=5,scoring='f1')
search.fit(X_train, y_train)

## using search's object "search.best_score_" to show the result and
search's object "best_params_" to show the result of best parameters.
print("Best parameter are (CV score=%0.3f):" % search.best_score_)
print(search.best_params_)
```

Additionally, here using f1_score to show the accuracy with matrix format.

$$F1Score = 2 * (Precision * Recall)/(Precision + Recall)$$

For example, here using training accuracy and testing accuracy to express the result with pipeline and using f1-score to check its reliability.

```
from sklearn.metrics import f1_score

scaler = StandardScaler()
pca = PCA(n_components=20)
svm = SVC(C=100, gamma=0.001)

pipe = Pipeline([('scaler', scaler), ('pca', pca), ('svm', svm)])
pipe.fit(X_train,y_train)

train_accuracy = pipe.score(X_train,y_train)
test_accuracy  = pipe.score(X_test,y_test)

f1_train = f1_score(y_train, pipe.predict(X_train))
f1_test  = f1_score(y_test, pipe.predict(X_test))

print(train_accuracy, test_accuracy)
print(f1_train, f1_test)

plot_confusion_matrix(pipe, X_test, y_test, normalize='true')
plt.show()
```

Multiclass Classification

For part II, the dataset is the multiclass classification which are 'cell line', 'disease', 'neoplasm', 'normal'. Here, data would not be removed due to its four-class classification.

```
# count the gene's number after it is transferred to Numpy arrays from
Dataframe
X = data.iloc[:,2:-1]
X = X.to_numpy()
y = data.iloc[:,-1]
# if you want to check its structure, it could use X.shape or y.shape to
see the result
```

Then, transfer four text labels to numeric number as annotation.

```
set(y) # {'cell line', 'disease', 'neoplasm', 'normal'}
```

then using preprocessing to handle data from four labels.

```
from sklearn import preprocessing

fourLabels = preprocessing.LabelEncoder()
fourLabels.fit(['cell line', 'disease', 'neoplasm', 'normal'])
fourLabels.transform(y) # set the text labels to numeric numbers with an
array ([0,0,...,3,3,3])
```

Then using Grid search and Cross validation to optimize the data

```
from sklearn.svm import SVC
from sklearn.model_selection import StratifiedKFold
from sklearn.metrics import f1_score
import time

# split data with test_size = 0.2
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)
print(X_train.shape, X_test.shape)

# scale data
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

skf = StratifiedKFold(n_splits=3, shuffle=True, random_state=42)

f1_table = np.zeros((4,3*3)) # store f1-score for all params combination
cols = [] # store values of params combination

start_time = time.time() # track execution time
count = 0 # counter for number of params combination
for gamma in [1e-04, 1e-03, 1e-02]:
    for C in [0.001, 1, 100]:
        f1_fold = np.zeros((3,4)) # array to store f1-score for each fold
(3 folds) for each class (4 classes)
        fold=0 # indexing f1_fold array
        # cross-validation
        for train_ind, val_ind  in skf.split(X_train, y_train):
            # PCA
            pca = PCA(n_components=20,
random_state=42).fit(X_train[train_ind])
            Z_train = pca.transform(X_train[train_ind])
            Z_val   = pca.transform(X_train[val_ind])
            # SVM
            svm = SVC(gamma=gamma, C=C, random_state=42).fit(Z_train,
y_train[train_ind])
            # get evalutaion metrics on val set
            y_pred = svm.predict(Z_val)
            f1 = f1_score(y_train[val_ind], y_pred, average=None)
            # store f1-score
            f1_fold[fold]=f1
            fold+=1
        f1_table[:,count] = np.around(np.mean(f1_fold,axis=0),decimals=2)
        cols.append(str(gamma)+"; "+str(C))
        count+=1

print("--- %s seconds ---" % (time.time() - start_time))
```

Then same with binary classification to plot scores for test set.

```
fig, ax = plt.subplots()
ax.set_axis_off()
rows=['cell line', 'disease', 'neoplasm', 'normal']
the_table = ax.table(cellText=f1_table,
                     rowLabels=rows,
                     colLabels=cols,
                     cellLoc ='center',
                     loc ='center')
the_table.auto_set_font_size(False)
the_table.set_fontsize(24)
the_table.scale(6, 4)
ax.set_title("F1 scores", fontsize=34, fontweight ="bold", pad=30)
plt.show()
```

Then, it would go to the final classification to see the result.

```
from sklearn.metrics import f1_score

scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

pca = PCA(n_components=20, random_state=42).fit(X_train)
Z_train = pca.transform(X_train)
Z_test  = pca.transform(X_test)

svm = SVC(C=100, gamma=0.001).fit(Z_train, y_train)

f1_train = f1_score(y_train, svm.predict(Z_train),average=None)
f1_test  = f1_score(y_test, svm.predict(Z_test),average=None)

print(f1_train, f1_test)

plot_confusion_matrix(svm, Z_test, y_test, normalize='true')
plt.show()
```