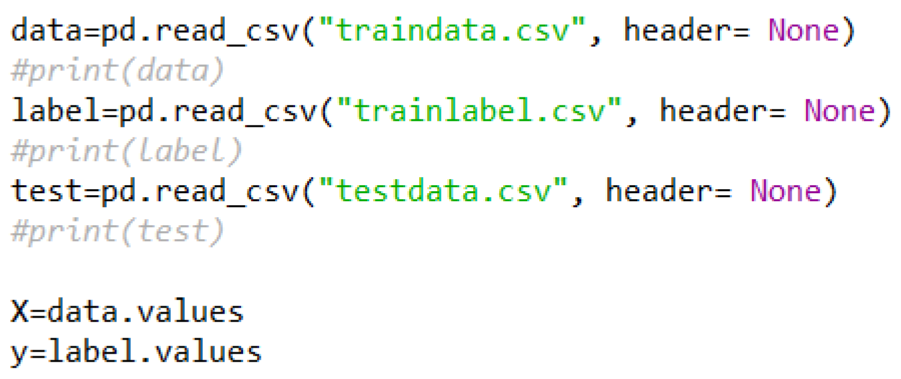
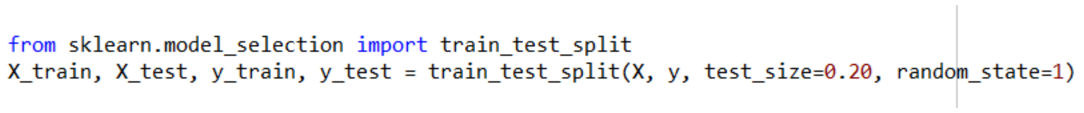
Firstly, the task is to import data.



```
data=pd.read_csv("traindata.csv", header= None)
#print(data)
label=pd.read_csv("trainlabel.csv", header= None)
#print(label)
test=pd.read_csv("testdata.csv", header= None)
#print(test)

X=data.values
y=label.values
```

Then, split those data with labels into training and testing sets.



```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, random_state=1)
```

After that the task will be the preprocessing of data. Since in this case, there are no missing data in this dataset, we do not need to deal with it. What’s more, there are no categorical data or non-numerical data, so we can safely ignore this step. As a result, we just apply standardization.

Dimensional reduction through PCA is tried at beginning but finally removed since it did not improve much performance. There may be two reasons. First, we have more than 2500 samples with 57 features for the training set, as a 2-categorical question the number of features is suitable. Second the meaning of feature is a black-box to us so it may be difficult to choose the number of n-components or manually selecting meaningful features.

For easy manipulation, pipeline, which can chain together multiple ML components into a single entity, is adopted. Five classification methods are used in this case, which are perception, Logistic Regression, SVM, Decision Tree and Random Forest.

(Code next page)

```python
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline

from sklearn.linear_model import Perceptron
ppn = Perceptron(n_iter=40, eta0=0.1, random_state=0)
pipe_ppn = Pipeline([('scl', StandardScaler()),
                     ('clf', ppn)])

from sklearn.linear_model import LogisticRegression
lr = LogisticRegression(C=1000.0, random_state=0)
pipe_lr = Pipeline([('scl', StandardScaler()),
                    ('clf', lr)])

from sklearn.svm import SVC
svm = SVC(kernel='rbf', random_state=0, gamma=0.2, C=1.0)
pipe_svm = Pipeline([('scl', StandardScaler()),
                     ('clf', svm)])

from sklearn.tree import DecisionTreeClassifier
tree = DecisionTreeClassifier(criterion='entropy', max_depth=3, random_state=0)
pipe_tree = Pipeline([('scl', StandardScaler()),
                      ('clf', tree)])

from sklearn.ensemble import RandomForestClassifier
forest = RandomForestClassifier(criterion='entropy',
                                n_estimators=10,
                                random_state=1,
                                n_jobs=2)
pipe_forest = Pipeline([('scl', StandardScaler()),
                        ('clf', forest)])
```

The hyperparameters are chosen through grid search cross validation. The search process has been deleted from code.

Next, train these models and judge their performance according to the classification accuracy on the testing sets.

```python
pipes = [pipe_ppn, pipe_lr, pipe_svm, pipe_tree, pipe_forest]

for pipe in pipes:
        pipe.fit(X_train, y_train)
        score = pipe.score(X_test, y_test)
        print(score)
```

The accuracies are finally:

```
0.888198757764
0.931677018634
0.854037267081
0.858695652174
0.950310559006
```

So we can conclude that decision tree has a better performance, and we will use it for this project.

Finally, make the prediction and save the results to a csv file.

```
prediction = pipe_tree.predict(test.values)
#print(prediction)
prediction = pd.DataFrame(prediction)
prediction.to_csv("project1_20461901.csv", header=False, index=False)
```