run.py

```
# coding:utf-8

'''
 @project: classification for class
 @author:
 @file: main.py
 @ide: pyCharm
 @time: 2019.09
'''

import sys
import importlib
import numpy as np
import pandas as pd
from random import shuffle

class DataProcess():
    '''
        This is a class used for data preprocessing. \
        It mainly reads data from files, preprocesses data, \
        and performs feature engineering to adapt to the model.

        There are some config param.
        The main input param is filename.

        The main fun includes load and process
    '''
    def __init__(self):
        self.filename = './datas.csv'
        self.datas = ""
        self.datas_np = ""
        self.x = ""
        self.y = ""
        self.x_train = ""
        self.y_train = ""
        self.x_test = ""
        self.y_test = ""

        # config param
        self.shuffle = True
        self.random_state = 50
        self.test_size = 80

        # fun run
        self.load()
        self.process()

    def load(self):
        '''
            read csv file
            input: filename, output:datas(type:DataFrame)
        '''
        self.datas = pd.read_csv(self.filename)

    def feature_engineer(self):
        '''
            key process, For the model, make some feature transformation, \
                    fit the principle of the model, improve the final effect.
        '''
        # feature names to id
        self.datas["gender"]=pd.factorize(self.datas["gender"])[0].astype(np.uint16)
        self.datas["NationalITy"]=pd.factorize(self.datas["NationalITy"])[0].astype(np.uint16)
        self.datas["PlaceofBirth"]=pd.factorize(self.datas["PlaceofBirth"])[0].astype(np.uint16)
        self.datas["StageID"]=pd.factorize(self.datas["StageID"])[0].astype(np.uint16)
        self.datas["GradeID"]=pd.factorize(self.datas["GradeID"])[0].astype(np.uint16)
        self.datas["SectionID"]=pd.factorize(self.datas["SectionID"])[0].astype(np.uint16)
        self.datas["Topic"]=pd.factorize(self.datas["Topic"])[0].astype(np.uint16)
        self.datas["Semester"]=pd.factorize(self.datas["Semester"])[0].astype(np.uint16)
        self.datas["Relation"]=pd.factorize(self.datas["Relation"])[0].astype(np.uint16)
        self.datas["ParentAnsweringSurvey"]=pd.factorize(self.datas["ParentAnsweringSurvey"])[0].astype(np.uint16)
        self.datas["ParentschoolSatisfaction"]=pd.factorize(self.datas["ParentschoolSatisfaction"])[0].astype(np.uint16)
        self.datas["StudentAbsenceDays"]=pd.factorize(self.datas["StudentAbsenceDays"])[0].astype(np.uint16)
        self.datas["Class"]=pd.factorize(self.datas["Class"])[0].astype(np.uint16)

        # Processing continuous value
        bins = [0, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100]
        self.datas["raisedhands"] = pd.cut(self.datas["raisedhands"], bins)
        self.datas["raisedhands"]=pd.factorize(self.datas["raisedhands"])[0].astype(np.uint16)
        self.datas["VisITedResources"] = pd.cut(self.datas["VisITedResources"], bins)
```

```
        self.datas["VisITedResources"]=pd.factorize(self.datas["VisITedResources"])[0].astype(np.uint16)
        self.datas["AnnouncementsView"] = pd.cut(self.datas["AnnouncementsView"], bins)
        self.datas["AnnouncementsView"]=pd.factorize(self.datas["AnnouncementsView"])[0].astype(np.uint16)
        self.datas["Discussion"] = pd.cut(self.datas["Discussion"], bins)
        self.datas["Discussion"]=pd.factorize(self.datas["Discussion"])[0].astype(np.uint16)

    def gen_train_test(self):
        #self.datas_np = self.datas.as_matrix()
        #self.datas_np = self.datas.values
        self.datas_np = np.array(self.datas)

        if self.shuffle:
            shuffle(self.datas_np)

        self.x = self.datas_np[:,:-1]
        self.y = self.datas_np[:,-1]

        from sklearn.model_selection import train_test_split
        self.x_train, self.x_test, self.y_train, self.y_test = train_test_split(
                self.x, self.y, \
                test_size=self.test_size, \
                random_state=self.random_state)

    def process(self):
        # test
        #print ("feature: [StudentAbsenceDays]")
        #print ("before ")
        #print (self.datas['StudentAbsenceDays'])
        # feature_engineer
        self.feature_engineer()
        #print ("after")
        #print (self.datas['StudentAbsenceDays'])
        #print (self.datas.head())
        #print (self.datas.tail())

        self.gen_train_test()



class Model():
    '''
        This is a class used for model. \
        It mainly reads data from dataProcess, \
        and performs model train, model test and model evaluation.

        There are some config param.
        The main input param is dataProcess and the output is classifier and the classification report

        The main fun includes train, test and evaluation.
    '''

    def __init__(self, data):
        self.data = data
        self.y_predict = ""
        self.model = ""

        #config param
        self.model_type = 2 # 1: ID3, 2: RF

        # fun run
        self.process()

        self.predict()

        self.evaluation()

    def train(self):
        '''
            using ID3 or RF algotithm to solve the problem.
        '''
        if self.model_type == 1:
            from sklearn import tree
            classifier = tree.DecisionTreeClassifier(criterion="entropy")
        elif self.model_type == 2:
            from sklearn.ensemble import RandomForestClassifier
            classifier = RandomForestClassifier()

        classifier.fit(self.data.x_train, self.data.y_train)
        self.model = classifier

    def predict(self):
        '''
```

```
        '''predict the test datas
        '''
        self.y_predict = self.model.predict(self.data.x_test)


    def evaluation(self):
        '''
            The experimental results are compared and analyzed, and some conclusions are obtained.
        '''
        from sklearn.metrics import classification_report
        target_names = ["1", "2", "3"]
        print (classification_report(self.data.y_test, \
                self.y_predict, target_names = target_names))

    def process(self):

        self.train()



if __name__ == '__main__':
    dataProcess = DataProcess()
    model = Model(dataProcess)
```