```
import warnings
import numpy as np
import pandas as pd
import itertools
import matplotlib.pyplot as plt
from sklearn import preprocessing
from sklearn.metrics import confusion_matrix
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score
from sklearn.linear_model.logistic import LogisticRegression


warnings.filterwarnings("ignore")
table = pd.read_csv('data.csv',skiprows=1, header=None)
columns = table.values[0]
data_csv = table.values[1:]
data_frame = pd.DataFrame(data_csv, columns=columns)
data_frame = data_frame.drop(['ID'], axis=1)
train_data_frame, test_data_frame = train_test_split(data_frame,
test_size=0.33, random_state=42)

print('No preprocessing:')
# logis-l1
print('logis-l1')
result = {}
minimum_cv_error = 1
lmbd = [10 ** x for x in np.arange(-2, 2, 0.1)]
for lamda in lmbd:
    cv_accuracy = []
    for times in range(5):
        train, validation = train_test_split(train_data_frame, test_size=0.33,
random_state=42)
        # Logistic regression
        classify = LogisticRegression(C=1 / lamda, penalty='l1',
solver='liblinear')
        classify.fit(train.drop(['default payment next month'], axis=1),
train['default payment next month'])

        # cv for model selection
        cv_result_list = cross_val_score(classify, validation.drop(['default payment next month'], axis=1), validation['default payment next month'], cv=5)
        cv_accuracy.append(np.mean(cv_result_list))
    result[lamda] = 1 - np.mean(cv_accuracy)
    minimum_cv_error = min(minimum_cv_error, result[lamda])

# print all lambda and cv error
for key in result.keys():
    print('when lambda =', key, 'cv error rate =', result[key])

# select the best model
for key in result.keys():
    if result[key] == minimum_cv_error:
        print('the best model is when lambda =', key)
        print('cv error rate =', minimum_cv_error)
        train, validation = train_test_split(train_data_frame, test_size=0.33, random_state=42)

        classify = LogisticRegression(C=1 / key, penalty='l1', solver='liblinear')
        classify.fit(train.drop(['default payment next month'], axis=1), train['default payment next month'])

        # print test error
        print('test error =', 1 - classify.score(test_data_frame.drop(['default payment next month'], axis=1), test_data_frame['default payment next month']))

        # print train error for whole train set
        print('train error =', 1 - classify.score(train.drop(['default payment next month'], axis=1), train['default payment next month']))
        break
print('')



# logis-l2
print('logis-l2')
result = {}
minimum_cv_error = 1
lmbd = [10 ** x for x in np.arange(-2, 2, 0.1)]
for lamda in lmbd:
    cv_accuracy = []
    for times in range(5):
        train, validation = train_test_split(train_data_frame, test_size=0.33,
random_state=42)
        # Logistic regression
        L2_classify = LogisticRegression(C=1 / lamda, penalty='l2',
solver='liblinear')
        L2_classify.fit(train.drop(['default payment next month'], axis=1),
train['default payment next month'])

        # cv for model selection
        cv_result_list = cross_val_score(L2_classify, validation.drop(['default
payment next month'], axis=1), validation['default payment next month'], cv=5)
        cv_accuracy.append(np.mean(cv_result_list))
    result[lamda] = 1 - np.mean(cv_accuracy)
    minimum_cv_error = min(minimum_cv_error, result[lamda])

# print all lambda and cv error
for key in result.keys():
    print('when lambda =', key, 'cv error rate =', result[key])

# select the best model
for key in result.keys():
    if result[key] == minimum_cv_error:
        print('the best model is when lambda =', key)
        print('cv error rate =', minimum_cv_error)
        train, validation = train_test_split(train_data_frame, test_size=0.33,
random_state=42)

        L2_classify = LogisticRegression(C=1 / key, penalty='l2',
solver='liblinear')
        L2_classify.fit(train.drop(['default payment next month'], axis=1),
train['default payment next month'])

        # print test error
        print('test error =', 1 -
L2_classify.score(test_data_frame.drop(['default payment next month'], axis=1),
test_data_frame['default payment next month']))

        # print train error for whole train set
        print('train error =', 1 - L2_classify.score(train.drop(['default payment
next month'], axis=1), train['default payment next month']))
        break
print('')

# knn
print('knn')
result = {}
minimum_cv_error = 1
for i in range(1,200,10):
    cv_accuracy = []
    for times in range(5):
        train, validation = train_test_split(train_data_frame, test_size=0.33,
random_state=42)

        knn = KNeighborsClassifier(n_neighbors=i)
        knn.fit(train.drop(['default payment next month'], axis=1),
train['default payment next month'])

        # cv for model selection
        cv_result_list = cross_val_score(knn, validation.drop(['default payment
next month'], axis=1),validation['default payment next month'], cv=5)
        cv_accuracy.append(np.mean(cv_result_list))

    result[i] = 1 - np.mean(cv_accuracy)
    minimum_cv_error = min(minimum_cv_error, result[i])

for key in result.keys():
    print('when k =', key, 'cv error rate =', result[key])

# select the best model
for key in result.keys():
    if result[key] == minimum_cv_error:
        print('the best model is when k =', key)
        print('cv error rate =', minimum_cv_error)
        train, validation = train_test_split(train_data_frame, test_size=0.33,
random_state=42)

        knn = KNeighborsClassifier(n_neighbors=key)
        knn.fit(train.drop(['default payment next month'], axis=1),
train['default payment next month'])
```

```
    # print test error
    print('test error =', 1 - knn.score(test_data_frame.drop(['default
payment next month'], axis=1), test_data_frame['default payment next month']))

    # print train error for whole train set
    print('train error =', 1 - knn.score(train.drop(['default payment next
month'], axis=1), train['default payment next month']))

    confusionMatrix = confusion_matrix(test_data_frame['default payment next
month'], knn.predict(test_data_frame.drop(['default payment next month'],
axis=1)))
    plt.imshow(confusionMatrix)
    labels = ['AB', 'NO']
    xlocation = np.array(range(len(labels)))
    plt.xticks(xlocation, labels, rotation=0)
    plt.yticks(xlocation, labels)
    plt.title('Confusion Matrix')
    plt.xlabel('true test label')
    plt.ylabel('predict test label')
    plt.colorbar()

    for i, j in itertools.product(range(confusionMatrix.shape[0]),
range(confusionMatrix.shape[1])):
      plt.text(j, i, confusionMatrix[i, j], horizontalalignment='center')

    plt.show()
    break
print('')



# random forest
print('random forest')
train_err, val_err = [], []
for i in range(5):
  val_res, train_res = [], []
  train, validation = train_test_split(train_data_frame, test_size=0.33,
random_state=42)
  for b in range(1, 31):
    bag, rest = train_test_split(train, train_size=1/3)
    model = RandomForestClassifier(n_estimators=b, bootstrap=True,
max_features=3)
        model.fit(bag.drop(['default payment next month'], axis=1), bag['default
payment next month'])
        train_res.append(1-model.score(bag.drop(['default payment next month'],
axis=1), bag['default payment next month']))

        cv_result_list = cross_val_score(model, validation.drop(['default payment
next month'], axis=1),validation['default payment next month'], cv=5)
        val_res.append(1-np.mean(cv_result_list))
    train_err.append(train_res)
    val_err.append(val_res)
train_err = pd.DataFrame(data=train_err, columns=range(1,31), index=None)
val_err = pd.DataFrame(data=val_err, columns=range(1,31), index=None)

mean_err_val, mean_err_train = [], []
for i in range(1, 31):
    mean_err_val.append(np.mean(val_err[i]))
    mean_err_train.append(np.mean(train_err[i]))

print('the best model is when b =', mean_err_val.index(min(mean_err_val))+1)
print('cv error rate =', min(mean_err_val))

model =
RandomForestClassifier(n_estimators=mean_err_val.index(min(mean_err_val))+1,
bootstrap=True, max_features=3)
model.fit(train_data_frame.drop(['default payment next month'], axis=1),
train_data_frame['default payment next month'])

print('train error =', mean_err_train[mean_err_val.index(min(mean_err_val))])
print('test error =', 1 - model.score(test_data_frame.drop(['default payment
next month'], axis=1), test_data_frame['default payment next month']))


plt.figure(1)
plt.title('Mean error rate on test and train set')
plt.plot(range(1, 31), mean_err_train, c='r', label='train')
plt.plot(range(1, 31), mean_err_val, c='b', label='validation')
plt.xlabel('b')
plt.ylabel('error rate')
plt.legend()
plt.show()

print('')
print('Standardized:')

table = pd.read_csv('data.csv',skiprows=1, header=None)
columns = table.values[0]
data_csv = table.values[1:]
data_frame = pd.DataFrame(data_csv, columns=columns)
data_frame = data_frame.drop(['ID'], axis=1)
train_data_frame, test_data_frame = train_test_split(data_frame, test_size=0.33, random_state=42)

remain = train_data_frame[['SEX','EDUCATION','MARRIAGE','PAY_0','PAY_2','PAY_3','PAY_4','PAY_5','PAY_6','default payment next month']]
change = train_data_frame.drop(['SEX','EDUCATION','MARRIAGE','PAY_0','PAY_2','PAY_3','PAY_4','PAY_5','PAY_6','default payment next month'], axis=1)
col = change.columns.values.tolist()

function = preprocessing.StandardScaler().fit(change)
change = function.transform(change)

# change = scale(change)
d = pd.DataFrame(change, columns=col)

for ele in col:
    remain[ele] = d[ele].values

train_data_frame = remain



remain = test_data_frame[['SEX','EDUCATION','MARRIAGE','PAY_0','PAY_2','PAY_3','PAY_4','PAY_5','PAY_6','default payment next month']]
change = test_data_frame.drop(['SEX','EDUCATION','MARRIAGE','PAY_0','PAY_2','PAY_3','PAY_4','PAY_5','PAY_6','default payment next month'], axis=1)
col = change.columns.values.tolist()
change = function.transform(change)
```

```
d = pd.DataFrame(change, columns=col)

for ele in col:
    remain[ele] = d[ele].values

test_data_frame = remain

# logis-l1
print('logis-l1')
result = {}
minimum_cv_error = 1
lmbd = [10 ** x for x in np.arange(-2, 2, 0.1)]
for lamda in lmbd:
    cv_accuracy = []
    for times in range(5):
        train, validation = train_test_split(train_data_frame, test_size=0.33,
random_state=42)
        # Logistic regression
        classify = LogisticRegression(C=1 / lamda, penalty='l1',
solver='liblinear')
        classify.fit(train.drop(['default payment next month'], axis=1),
train['default payment next month'])

        # cv for model selection
        cv_result_list = cross_val_score(classify, validation.drop(['default
payment next month'], axis=1), validation['default payment next month'], cv=5)
        cv_accuracy.append(np.mean(cv_result_list))
    result[lamda] = 1 - np.mean(cv_accuracy)
    minimum_cv_error = min(minimum_cv_error, result[lamda])

# print all lambda and cv error
for key in result.keys():
    print('when lambda =', key, 'cv error rate =', result[key])

# select the best model
for key in result.keys():
    if result[key] == minimum_cv_error:
        print('the best model is when lambda =', key)
        print('cv error rate =', minimum_cv_error)
        train, validation = train_test_split(train_data_frame, test_size=0.33,
random_state=42)

        classify = LogisticRegression(C=1 / key, penalty='l1',
solver='liblinear')
        classify.fit(train.drop(['default payment next month'], axis=1),
train['default payment next month'])

        # print test error
        print('test error =', 1 - classify.score(test_data_frame.drop(['default
payment next month'], axis=1), test_data_frame['default payment next month']))

        # print train error for whole train set
        print('train error =', 1 - classify.score(train.drop(['default payment
next month'], axis=1), train['default payment next month']))
        break
print('')



# logis-l2
print('logis-l2')
result = {}
minimum_cv_error = 1
lmbd = [10 ** x for x in np.arange(-2, 2, 0.1)]
for lamda in lmbd:
    cv_accuracy = []
    for times in range(5):
        train, validation = train_test_split(train_data_frame, test_size=0.33,
random_state=42)
        # Logistic regression
        L2_classify = LogisticRegression(C=1 / lamda, penalty='l2',
solver='liblinear')
        L2_classify.fit(train.drop(['default payment next month'], axis=1),
train['default payment next month'])

        # cv for model selection
        cv_result_list = cross_val_score(L2_classify, validation.drop(['default
payment next month'], axis=1), validation['default payment next month'], cv=5)
        cv_accuracy.append(np.mean(cv_result_list))
    result[lamda] = 1 - np.mean(cv_accuracy)
    minimum_cv_error = min(minimum_cv_error, result[lamda])
```

```
# print all lambda and cv error
for key in result.keys():
  print('when lambda =', key, 'cv error rate =', result[key])

# select the best model
for key in result.keys():
  if result[key] == minimum_cv_error:
    print('the best model is when lambda =', key)
    print('cv error rate =', minimum_cv_error)
    train, validation = train_test_split(train_data_frame, test_size=0.33,
random_state=42)

    L2_classify = LogisticRegression(C=1 / key, penalty='l2',
solver='liblinear')
    L2_classify.fit(train.drop(['default payment next month'], axis=1),
train['default payment next month'])

    # print test error
    print('test error =', 1 -
L2_classify.score(test_data_frame.drop(['default payment next month'], axis=1),
test_data_frame['default payment next month']))

    # print train error for whole train set
    print('train error =', 1 - L2_classify.score(train.drop(['default payment
next month'], axis=1), train['default payment next month']))
    break
print('')

# knn
print('knn')
result = {}
minimum_cv_error = 1
for i in range(1,200,10):
  cv_accuracy = []
  for times in range(5):
    train, validation = train_test_split(train_data_frame, test_size=0.33,
random_state=42)

    knn = KNeighborsClassifier(n_neighbors=i)
    knn.fit(train.drop(['default payment next month'], axis=1),
train['default payment next month'])

        # cv for model selection
        cv_result_list = cross_val_score(knn, validation.drop(['default payment
next month'], axis=1),validation['default payment next month'], cv=5)
        cv_accuracy.append(np.mean(cv_result_list))

    result[i] = 1 - np.mean(cv_accuracy)
    minimum_cv_error = min(minimum_cv_error, result[i])

for key in result.keys():
    print('when k =', key, 'cv error rate =', result[key])

# select the best model
for key in result.keys():
    if result[key] == minimum_cv_error:
        print('the best model is when k =', key)
        print('cv error rate =', minimum_cv_error)
        train, validation = train_test_split(train_data_frame, test_size=0.33,
random_state=42)

        knn = KNeighborsClassifier(n_neighbors=key)
        knn.fit(train.drop(['default payment next month'], axis=1),
train['default payment next month'])

        # print test error
        print('test error =', 1 - knn.score(test_data_frame.drop(['default
payment next month'], axis=1), test_data_frame['default payment next month']))

        # print train error for whole train set
        print('train error =', 1 - knn.score(train.drop(['default payment next
month'], axis=1), train['default payment next month']))

        confusionMatrix = confusion_matrix(test_data_frame['default payment next
month'], knn.predict(test_data_frame.drop(['default payment next month'],
axis=1)))
        plt.imshow(confusionMatrix)
        labels = ['AB', 'NO']
        xlocation = np.array(range(len(labels)))
        plt.xticks(xlocation, labels, rotation=0)
        plt.yticks(xlocation, labels)
    plt.title('Confusion Matrix')
    plt.xlabel('true test label')
    plt.ylabel('predict test label')
    plt.colorbar()

    for i, j in itertools.product(range(confusionMatrix.shape[0]),
range(confusionMatrix.shape[1])):
      plt.text(j, i, confusionMatrix[i, j], horizontalalignment='center')

    plt.show()
    break
print('')



# random forest
print('random forest')
train_err, val_err = [], []
for i in range(5):
  val_res, train_res = [], []
  train, validation = train_test_split(train_data_frame, test_size=0.33,
random_state=42)
  for b in range(1, 31):
    bag, rest = train_test_split(train, train_size=1/3)
    model = RandomForestClassifier(n_estimators=b, bootstrap=True,
max_features=3)
    model.fit(bag.drop(['default payment next month'], axis=1), bag['default
payment next month'])
    train_res.append(1-model.score(bag.drop(['default payment next month'],
axis=1), bag['default payment next month']))

    cv_result_list = cross_val_score(model, validation.drop(['default payment
next month'], axis=1),validation['default payment next month'], cv=5)
    val_res.append(1-np.mean(cv_result_list))
  train_err.append(train_res)
  val_err.append(val_res)
train_err = pd.DataFrame(data=train_err, columns=range(1,31), index=None)
val_err = pd.DataFrame(data=val_err, columns=range(1,31), index=None)

mean_err_val, mean_err_train = [], []
for i in range(1, 31):
    mean_err_val.append(np.mean(val_err[i]))
    mean_err_train.append(np.mean(train_err[i]))

print('the best model is when b =', mean_err_val.index(min(mean_err_val))+1)
print('cv error rate =', min(mean_err_val))

model =
RandomForestClassifier(n_estimators=mean_err_val.index(min(mean_err_val))+1,
bootstrap=True, max_features=3)
model.fit(train_data_frame.drop(['default payment next month'], axis=1),
train_data_frame['default payment next month'])

print('train error =', mean_err_train[mean_err_val.index(min(mean_err_val))])
print('test error =', 1 - model.score(test_data_frame.drop(['default payment
next month'], axis=1), test_data_frame['default payment next month']))


plt.figure(1)
plt.title('Mean error rate on test and train set')
plt.plot(range(1, 31), mean_err_train, c='r', label='train')
plt.plot(range(1, 31), mean_err_val, c='b', label='validation')
plt.xlabel('b')
plt.ylabel('error rate')
plt.legend()

plt.show()

print('')
print('Features extraction:')

table = pd.read_csv('data.csv',skiprows=1, header=None)
columns = table.values[0]
data_csv = table.values[1:].astype('float')
data_frame = pd.DataFrame(data_csv, columns=columns)
data_frame = data_frame.drop(['ID'], axis=1)
train_data_frame, test_data_frame = train_test_split(data_frame,
test_size=0.33, random_state=42)

BILL =
train_data_frame[['BILL_AMT1','BILL_AMT2','BILL_AMT3','BILL_AMT4','BILL_AMT5','
BILL_AMT6']].values
```

```
PAY =
train_data_frame[['PAY_AMT1','PAY_AMT2','PAY_AMT3','PAY_AMT4','PAY_AMT5','PAY_AMT6']].values
LIMIT_BAL = train_data_frame['LIMIT_BAL'].values.reshape(-1,1)
remain =
train_data_frame.drop(['BILL_AMT1','BILL_AMT2','BILL_AMT3','BILL_AMT4','BILL_AMT5','BILL_AMT6',
'PAY_AMT1','PAY_AMT2','PAY_AMT3','PAY_AMT4','PAY_AMT5','PAY_AMT6','LIMIT_BAL'],
axis=1)

bill = []
pay = []
for i in range(len(BILL)):
    bill.append([np.mean(BILL[i]), np.std(BILL[i])])
    pay.append([np.mean(PAY[i]), np.std(PAY[i])])

function_bill = preprocessing.StandardScaler().fit(bill)
bill = function_bill.transform(bill)

function_pay = preprocessing.StandardScaler().fit(pay)
pay = function_bill.transform(pay)

function_limit = preprocessing.StandardScaler().fit(LIMIT_BAL)
LIMIT_BAL = function_limit.transform(LIMIT_BAL)

BILL = pd.DataFrame(bill, columns=['bill_mean','bill_std'])
PAY = pd.DataFrame(pay, columns=['pay_mean','pay_std'])
remain['LIMIT_BAL'] = LIMIT_BAL
remain['bill_mean'] = BILL['bill_mean'].values
remain['bill_std'] = BILL['bill_std'].values
remain['pay_mean'] = PAY['pay_mean'].values
remain['pay_std'] = PAY['pay_std'].values
train_data_frame = remain


BILL =
test_data_frame[['BILL_AMT1','BILL_AMT2','BILL_AMT3','BILL_AMT4','BILL_AMT5','BILL_AMT6']].values
PAY =
test_data_frame[['PAY_AMT1','PAY_AMT2','PAY_AMT3','PAY_AMT4','PAY_AMT5','PAY_AMT6']].values
```

```
LIMIT_BAL = test_data_frame['LIMIT_BAL'].values.reshape(-1,1)
remain =
test_data_frame.drop(['BILL_AMT1','BILL_AMT2','BILL_AMT3','BILL_AMT4','BILL_AMT
5','BILL_AMT6',
'PAY_AMT1','PAY_AMT2','PAY_AMT3','PAY_AMT4','PAY_AMT5','PAY_AMT6','LIMIT_BAL'],
axis=1)

bill = []
pay = []
for i in range(len(BILL)):
    bill.append([np.mean(BILL[i]), np.std(BILL[i])])
    pay.append([np.mean(PAY[i]), np.std(PAY[i])])

bill = function_bill.transform(bill)
pay = function_bill.transform(pay)
LIMIT_BAL = function_limit.transform(LIMIT_BAL)
BILL = pd.DataFrame(bill, columns=['bill_mean','bill_std'])
PAY = pd.DataFrame(pay, columns=['pay_mean','pay_std'])
remain['LIMIT_BAL'] = LIMIT_BAL
remain['bill_mean'] = BILL['bill_mean'].values
remain['bill_std'] = BILL['bill_std'].values
remain['pay_mean'] = PAY['pay_mean'].values
remain['pay_std'] = PAY['pay_std'].values
test_data_frame = remain


# logis-l1
print('logis-l1')
result = {}
minimum_cv_error = 1
lmbd = [10 ** x for x in np.arange(-2, 2, 0.1)]
for lamda in lmbd:
    cv_accuracy = []
    for times in range(5):
        train, validation = train_test_split(train_data_frame, test_size=0.33,
random_state=42)
        # Logistic regression
        classify = LogisticRegression(C=1 / lamda, penalty='l1',
solver='liblinear')
        classify.fit(train.drop(['default payment next month'], axis=1),
train['default payment next month'])
```

```
        # cv for model selection
        cv_result_list = cross_val_score(classify, validation.drop(['default
payment next month'], axis=1), validation['default payment next month'], cv=5)
        cv_accuracy.append(np.mean(cv_result_list))
    result[lamda] = 1 - np.mean(cv_accuracy)
    minimum_cv_error = min(minimum_cv_error, result[lamda])

# print all lambda and cv error
for key in result.keys():
    print('when lambda =', key, 'cv error rate =', result[key])

# select the best model
for key in result.keys():
    if result[key] == minimum_cv_error:
        print('the best model is when lambda =', key)
        print('cv error rate =', minimum_cv_error)
        train, validation = train_test_split(train_data_frame, test_size=0.33,
random_state=42)

        classify = LogisticRegression(C=1 / key, penalty='l1',
solver='liblinear')
        classify.fit(train.drop(['default payment next month'], axis=1),
train['default payment next month'])

        # print test error
        print('test error =', 1 - classify.score(test_data_frame.drop(['default
payment next month'], axis=1), test_data_frame['default payment next month']))

        # print train error for whole train set
        print('train error =', 1 - classify.score(train.drop(['default payment
next month'], axis=1), train['default payment next month']))
        break
print('')



# logis-l2
print('logis-l2')
result = {}
minimum_cv_error = 1
```

```
lmbd = [10 ** x for x in np.arange(-2, 2, 0.1)]
for lamda in lmbd:
    cv_accuracy = []
    for times in range(5):
        train, validation = train_test_split(train_data_frame, test_size=0.33,
random_state=42)
        # Logistic regression
        L2_classify = LogisticRegression(C=1 / lamda, penalty='l2',
solver='liblinear')
        L2_classify.fit(train.drop(['default payment next month'], axis=1),
train['default payment next month'])

        # cv for model selection
        cv_result_list = cross_val_score(L2_classify, validation.drop(['default
payment next month'], axis=1), validation['default payment next month'], cv=5)
        cv_accuracy.append(np.mean(cv_result_list))
    result[lamda] = 1 - np.mean(cv_accuracy)
    minimum_cv_error = min(minimum_cv_error, result[lamda])

# print all lambda and cv error
for key in result.keys():
    print('when lambda =', key, 'cv error rate =', result[key])

# select the best model
for key in result.keys():
    if result[key] == minimum_cv_error:
        print('the best model is when lambda =', key)
        print('cv error rate =', minimum_cv_error)
        train, validation = train_test_split(train_data_frame, test_size=0.33,
random_state=42)

        L2_classify = LogisticRegression(C=1 / key, penalty='l2',
solver='liblinear')
        L2_classify.fit(train.drop(['default payment next month'], axis=1),
train['default payment next month'])

        # print test error
        print('test error =', 1 -
L2_classify.score(test_data_frame.drop(['default payment next month'], axis=1),
test_data_frame['default payment next month']))
```

```
        # print train error for whole train set
        print('train error =', 1 - L2_classify.score(train.drop(['default payment
next month'], axis=1), train['default payment next month']))
        break
print('')

# knn
print('knn')
result = {}
minimum_cv_error = 1
for i in range(1,200,10):
    cv_accuracy = []
    for times in range(5):
        train, validation = train_test_split(train_data_frame, test_size=0.33,
random_state=42)

        knn = KNeighborsClassifier(n_neighbors=i)
        knn.fit(train.drop(['default payment next month'], axis=1),
train['default payment next month'])

        # cv for model selection
        cv_result_list = cross_val_score(knn, validation.drop(['default payment
next month'], axis=1),validation['default payment next month'], cv=5)
        cv_accuracy.append(np.mean(cv_result_list))

    result[i] = 1 - np.mean(cv_accuracy)
    minimum_cv_error = min(minimum_cv_error, result[i])

for key in result.keys():
    print('when k =', key, 'cv error rate =', result[key])

# select the best model
for key in result.keys():
    if result[key] == minimum_cv_error:
        print('the best model is when k =', key)
        print('cv error rate =', minimum_cv_error)
        train, validation = train_test_split(train_data_frame, test_size=0.33,
random_state=42)

        knn = KNeighborsClassifier(n_neighbors=key)
        knn.fit(train.drop(['default payment next month'], axis=1),
train['default payment next month'])

        # print test error
        print('test error =', 1 - knn.score(test_data_frame.drop(['default
payment next month'], axis=1), test_data_frame['default payment next month']))

        # print train error for whole train set
        print('train error =', 1 - knn.score(train.drop(['default payment next
month'], axis=1), train['default payment next month']))

        confusionMatrix = confusion_matrix(test_data_frame['default payment next
month'], knn.predict(test_data_frame.drop(['default payment next month'],
axis=1)))
        plt.imshow(confusionMatrix)
        labels = ['AB', 'NO']
        xlocation = np.array(range(len(labels)))
        plt.xticks(xlocation, labels, rotation=0)
        plt.yticks(xlocation, labels)
        plt.title('Confusion Matrix')
        plt.xlabel('true test label')
        plt.ylabel('predict test label')
        plt.colorbar()

        for i, j in itertools.product(range(confusionMatrix.shape[0]),
range(confusionMatrix.shape[1])):
            plt.text(j, i, confusionMatrix[i, j], horizontalalignment='center')

        plt.show()
        break
print('')



# random forest
print('random forest')
train_err, val_err = [], []
for i in range(5):
    val_res, train_res = [], []
    train, validation = train_test_split(train_data_frame, test_size=0.33,
random_state=42)
    for b in range(1, 31):
        bag, rest = train_test_split(train, train_size=1/3)
        model = RandomForestClassifier(n_estimators=b, bootstrap=True, max_features=3)
        model.fit(bag.drop(['default payment next month'], axis=1), bag['default payment next month'])
        train_res.append(1-model.score(bag.drop(['default payment next month'], axis=1), bag['default payment next month']))

        cv_result_list = cross_val_score(model, validation.drop(['default payment next month'], axis=1),validation['default payment next month'], cv=5)
        val_res.append(1-np.mean(cv_result_list))
    train_err.append(train_res)
    val_err.append(val_res)
train_err = pd.DataFrame(data=train_err, columns=range(1,31), index=None)
val_err = pd.DataFrame(data=val_err, columns=range(1,31), index=None)

mean_err_val, mean_err_train = [], []
for i in range(1, 31):
    mean_err_val.append(np.mean(val_err[i]))
    mean_err_train.append(np.mean(train_err[i]))

print('the best model is when b =', mean_err_val.index(min(mean_err_val))+1)
print('cv error rate =', min(mean_err_val))

model = RandomForestClassifier(n_estimators=mean_err_val.index(min(mean_err_val))+1, bootstrap=True, max_features=3)
model.fit(train_data_frame.drop(['default payment next month'], axis=1), train_data_frame['default payment next month'])

print('train error =', mean_err_train[mean_err_val.index(min(mean_err_val))])
print('test error =', 1 - model.score(test_data_frame.drop(['default payment next month'], axis=1), test_data_frame['default payment next month']))


plt.figure(1)
plt.title('Mean error rate on test and train set')
plt.plot(range(1, 31), mean_err_train, c='r', label='train')
plt.plot(range(1, 31), mean_err_val, c='b', label='validation')
plt.xlabel('b')
plt.ylabel('error rate')
plt.legend()

plt.show()
```