CSCI 6364 - Machine Learning

Project 3 - Adult Census Income

Student: Shifeng Yuan

GWid: G32115270

Language: Python

Resource: Adult Census Income from kaggle

1. Dataset preprocessing and interpretation

a. Abandon samples with missing terms.

In [383]:
```
import pandas as pd
import numpy as np
columns = ['age','workclass','fnlwgt','education','education_num','marital_status',
           'occupation','relationship','race','sex','capital_gain','capital_loss',
           'hours_per_week','native_country','income']
# Read the dataset and then print the head
dataset = pd.read_csv('adult.csv', names=columns)
# If any row contains "?", delete it from the dataset
dataset = dataset[~(dataset.astype(str) == '?').any(1)]
dataset.drop(dataset.index[0],inplace=True)
dataset = dataset.sample(frac=0.1, replace=True)
```

b. Dealing with discrete (categorical) features. Here, I switch the discrete features to numbers:

In [384]:
```
dataset.replace(
    {'marital_status' : {'Married-spouse-absent': 7, 'Married-civ-spouse': 6, 'Married-AF-spouse': 5,'Divorced':
      'Widowed': 1
     },
     'relationship': {'Wife': 1,'Own-child': 1,'Husband': 2, 'Not-in-family': 3, 'Other-relative': 4, 'Unmarried
     },
     'workclass' : {'Private' : 15, 'Self-emp-not-inc' : 3, 'Self-emp-inc' : 5, 'Federal-gov' : 12, 'Local-gov'
     },
     'occupation' : {'Tech-support' : 11, 'Craft-repair' : 2, 'Other-service' : 7, 'Sales' : 10, 'Exec-manageria
     },
     'race' : {'White' : 5, 'Asian-Pac-Islander' : 4, 'Amer-Indian-Eskimo' : 3, 'Other' : 2, 'Black' : 1
     },
     'sex' : {'Female' : 1, 'Male' : 0
     }
    }, inplace=True)
dataset.replace(['United-States', 'Cambodia', 'England', 'Puerto-Rico', 'Canada', 'Germany', 'Outlying-US(Guam-U
del dataset['education']
dataset.replace(['<=50K', '>50K'], [-1, 1], inplace = True)
print(dataset.head())
```

```
       age  workclass  fnlwgt education_num  marital_status  occupation  \
22801  23         15  322674            10               4           5
1754   68          8  242095            13               6           3
4800   35         15  292472            16               6          12
4901   37         15  280282            11               6          11
22175  26         15  118497             9               6           9

        relationship  race  sex capital_gain capital_loss hours_per_week  \
22801              3     5    0            0            0             40
1754               2     5    0        20051            0             40
4800               2     4    0            0            0             40
4901               1     5    1            0            0             24
22175              2     5    0            0            0             50

        native_country  income
22801                1      -1
1754                 1       1
4800                 0       1
4901                 1       1
22175                1      -1
```

c. Split the dataset for stratified 10-fold-cross validation.

In [395]:

```
# split the dataset into 10 folds
def ten_folds_split(data):
    size = len(data)
    step = size // 10
    folds = [data[i: i+step].sample(frac=1) for i in range(0, size, step)]
    return folds

# split the <=50k and >50k data
more_than_50k = dataset[dataset['income'] == 1]
less_than_50k = dataset[dataset['income'] == -1]
# split the more and less into 10 folds respectively
more_than_50k = ten_folds_split(more_than_50k)
less_than_50k = ten_folds_split(less_than_50k)
# make new dataset in 10 folds
def make_new_dataset(more_than_50k, less_than_50k):
    new_data = pd.DataFrame(columns=['age','workclass','fnlwgt','education_num','marital_status',
           'occupation','relationship','race','sex','capital_gain','capital_loss',
           'hours_per_week','native_country','income'])
    for i in range(10):
        frames1 = [more_than_50k[i], less_than_50k[i]]
        fold = pd.concat(frames1)
        frames2 = [new_data, fold]
        new_data = pd.concat(frames2)
    new_data = ten_folds_split(new_data)
    return new_data
new_dataset = make_new_dataset(more_than_50k, less_than_50k)
```

d. Analyze the features and make a scatter plot with the two features that have the highest information gain.

In [396]:

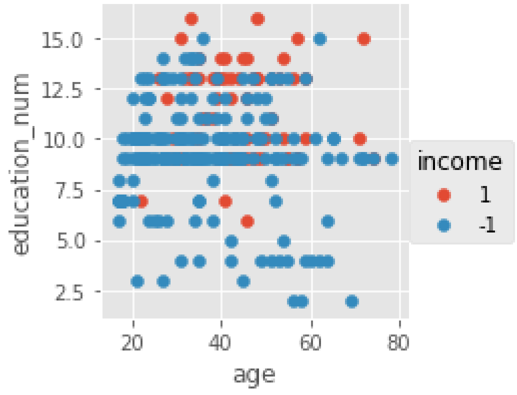
```python
# function of calculating information gain
import math
def information_gain(feature, dataset, label):
    # for a feature, e.g. age, the amount of every value
    feature_num_dict = {}
    for key in feature:
        feature_num_dict[key] = feature_num_dict.get(key, 0) + 1
    # calculate the amount of each feature value with respect to the result
    # every value with the income is more than 50k
    feature_more_than_50k_num_dict = {}
    # every value with the income is less than 50k
    feature_less_than_50k_num_dict = {}
    for key in feature_num_dict.keys():
        for income in dataset[feature.isin([key])].income:
            if income == 1:
                feature_more_than_50k_num_dict[key] = feature_more_than_50k_num_dict.get(key, 0) + 1
            else:
                feature_less_than_50k_num_dict[key] = feature_less_than_50k_num_dict.get(key, 0) + 1
    # calculate p of every value with respect to income
    feature_more_than_50k_p_dict = {}
    feature_less_than_50k_p_dict = {}
    for key in feature_num_dict.keys():
        if key in feature_more_than_50k_num_dict.keys():
            feature_more_than_50k_p_dict[key] = feature_more_than_50k_num_dict[key] / feature_num_dict[key]
        if key in feature_less_than_50k_num_dict.keys():
            feature_less_than_50k_p_dict[key] = feature_less_than_50k_num_dict[key] / feature_num_dict[key]
    weighted_feature_entropy = {}
    for key in feature_num_dict.keys():
        temp1 = 0
        temp2 = 0
        if key in feature_more_than_50k_p_dict.keys():
            temp1 = feature_more_than_50k_p_dict[key] * math.log(feature_more_than_50k_p_dict[key])
        if key in feature_less_than_50k_p_dict.keys():
            temp2 = feature_less_than_50k_p_dict[key] * math.log(feature_less_than_50k_p_dict[key])
        weighted_feature_entropy[key] = -(temp1 + temp2) * feature_num_dict[key] / len(feature)
    p_income = {}
    for key in label:
        p_income[key] = p_income.get(key, 0) + 1
    for key in p_income.keys():
        p_income[key] = p_income[key] / len(feature)
    # calculate the entropy of the label
    entropy_income = 0
    for key in p_income.keys():
        temp = - (p_income[key] * math.log(p_income[key], 2))
        entropy_income = entropy_income + temp
    # calculate the information gain
    information_gain = 0
    for key in weighted_feature_entropy.keys():
        information_gain = entropy_income - weighted_feature_entropy[key]
#         print(information_gain, len(weighted_feature_entropy))
    return information_gain

print("__________________ information gain of each feature: ________________")
print('age: ', information_gain(new_dataset[1].age, new_dataset[1], new_dataset[1].income))
print('workclass: ', information_gain(new_dataset[1].workclass, new_dataset[1], new_dataset[1].income))
print('fnlwgt: ', information_gain(new_dataset[1].fnlwgt, new_dataset[1], new_dataset[1].income))
print('education_num: ', information_gain(new_dataset[1].education_num, new_dataset[1], new_dataset[1].income))
print('marital_status: ', information_gain(new_dataset[1].marital_status, new_dataset[1], new_dataset[1].income)
print('occupation: ', information_gain(new_dataset[1].occupation, new_dataset[1], new_dataset[1].income))
print('relationship: ', information_gain(new_dataset[1].relationship, new_dataset[1], new_dataset[1].income))
print('race: ', information_gain(new_dataset[1].race, new_dataset[1], new_dataset[1].income))
print('sex: ', information_gain(new_dataset[1].sex, new_dataset[1], new_dataset[1].income))
print('capital_gain: ', information_gain(new_dataset[1].capital_gain, new_dataset[1], new_dataset[1].income))
print('capital_loss: ', information_gain(new_dataset[1].capital_loss, new_dataset[1], new_dataset[1].income))
print('hours_per_week: ', information_gain(new_dataset[1].hours_per_week, new_dataset[1], new_dataset[1].income)
print('native_country: ', information_gain(new_dataset[1].native_country, new_dataset[1], new_dataset[1].income)
```

```
__________________ information gain of each feature: ________________
age:  0.8152035337512442
workclass:  0.8152035337512442
fnlwgt:  0.8152035337512442
education_num:  0.8152035337512442
marital_status:  0.8077306414639511
occupation:  0.8152035337512442
relationship:  0.8047733381944758
race:  0.8152035337512442
sex:  0.40005459578626396
capital_gain:  0.8152035337512442
capital_loss:  0.8152035337512442
hours_per_week:  0.8152035337512442
native_country:  0.7751952754055002
```

As we can see from avove, we can select 'age' and 'education_num' as the two features

In [397]:
```
import matplotlib.pyplot as plt
import seaborn as sns
plt.style.use('ggplot')
plt.figure(figsize=(80,40))

income_type = [1, -1]
df = pd.DataFrame({
    'age': pd.to_numeric(new_dataset[1].age.values),
    'education_num': pd.to_numeric(new_dataset[1].education_num.values),
    'income': pd.to_numeric(new_dataset[1].income.values)
})
fg = sns.FacetGrid(data=df, hue='income', hue_order=income_type)
fg.map(plt.scatter, 'age', 'education_num').add_legend()
print("_____________  income '1' is > 50k, '-1' is <= 50k  ______________")
plt.show()
```

```
_____________  income '1' is > 50k, '-1' is <= 50k  ______________

<Figure size 5760x2880 with 0 Axes>
```



2. Implement a linear soft-margin SVM

a. Train your SVM with stratified 10-fold-cross-validation on the 2 features you selected and visualize your boundary. i.e. plot the support vectors and draw the decision boundary.

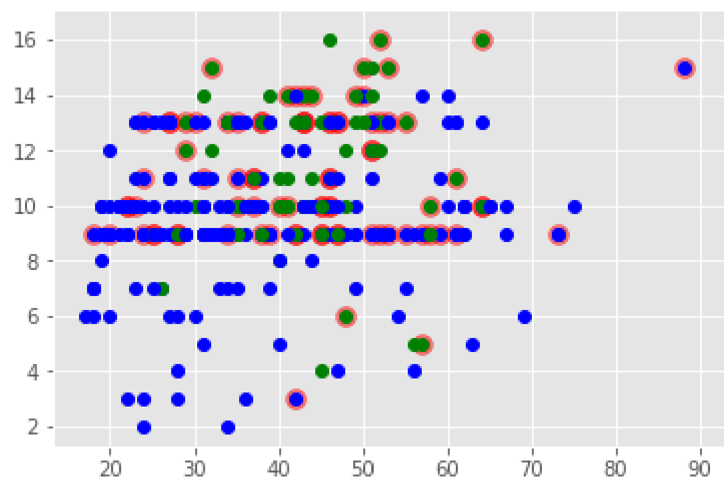
In [505]:

```
from numpy import *
np.seterr(divide='ignore', invalid='ignore')

def alpha_prune(a, top, bottom):
    if a > top:
        a = top
    if bottom > a:
        a = bottom
    return a

def e_calc(svm, i):
    matrix = np.multiply(svm.alphas,svm.labelMat).T
    temp = matrix * svm.Ker[:,i] + svm.b
    f = float(temp)
    e = f - float(svm.labelMat[i])
    return e

# load dataset
def read_data(dataset):
    dataset = pd.DataFrame(dataset)
    col_length = dataset.columns.size
    features = dataset.iloc[:, [0, 3]].apply(pd.to_numeric, errors='ignore')
    label = dataset.iloc[:, -1].apply(pd.to_numeric, errors='ignore').values
    return features, label

def kernel_calc(tree, data_mat, K_type):
    row, col = np.shape(tree)
    # k is a matrix filled with zeros
    Ker = np.mat(np.zeros((row,1)))
    if K_type[0]=='linear': # linear kernel
        Ker = tree * data_mat.T
    elif K_type[0]=='rbf': # gaussian kernel
        for i in range(row):
            d = tree[i,:] - data_mat
            Ker[i] = d * d.T
        Ker = np.exp(Ker/(-1*K_type[1]**2))
    else:
        raise NameError('Kernel Type not included')
    return Ker

def random_select(a, b):
    num = a
    while (num == a):
        num = int(random.uniform(0,b))
    return num

#定义类，方便存储数据
class data_struct:
    def __init__(self, feature_data, data_type, Cons, stoper, ker_type):
        self.b = 0
        self.s = stoper
        self.feature = feature_data
        self.row = np.shape(feature_data)[0]
        self.labelMat = data_type
        self.Cons = Cons
        self.alphas = np.mat(np.zeros((self.row,1)))
        self.eCache = np.mat(np.zeros((self.row,2)))
        self.Ker = np.mat(np.zeros((self.row,self.row)))
        for i in range(self.row):
            temp = kernel_calc(self.feature, self.feature[i,:], ker_type)
            self.Ker[:,i] = temp

#随机选取aj，并返回其E值
def aj_selector(i, svm, Ei):
    k_m = -1
    del_e_m = 0
    e = 0
    svm.eCache[i] = [1,Ei]
    temp = svm.eCache[:,0].A
    e_list = nonzero(temp)[0]
    e_length = len(e_list)
    if (e_length) > 1:
        for e_k in e_list:
            if e_k == i:
                continue
            Ek = e_calc(svm, e_k)
            deltaE = abs(Ei - Ek)
            if (deltaE > del_e_m):
                k_m = e_k
                del_e_m = deltaE
                e = Ek
        return k_m, e
    else:
        j = random_select(i, svm.row)
        e = e_calc(svm, j)
    return j, e
```

```python
def e_updator(svm, i):
    e = e_calc(svm, i)
    svm.eCache[i] = [1,e]


def alphas_opt(i, svm):
    e_i = e_calc(svm, i)
    if ((svm.labelMat[i]*e_i < -svm.s) and (svm.alphas[i] < svm.Cons)) or ((svm.labelMat[i]*e_i > svm.s) and (sv
        j,Ej = aj_selector(i, svm, e_i)
        al_I_old = svm.alphas[i].copy()
        al_J_old = svm.alphas[j].copy()
        if (svm.labelMat[i] != svm.labelMat[j]):
            var = svm.alphas[j] - svm.alphas[i]
            bottom = max(0, var)
            temp = svm.Cons + svm.alphas[j] - svm.alphas[i]
            top = min(svm.Cons, temp)
        else:
            temp = svm.alphas[j] + svm.alphas[i] - svm.Cons
            bottom = max(0, temp)
            top = min(svm.Cons, svm.alphas[j] + svm.alphas[i])
        if bottom == top:
#              print("L==H")
            return 0
        eta = 2.0 * svm.Ker[i,j] - svm.Ker[i,i] - svm.Ker[j,j]
        if eta >= 0:
#              print("eta>=0")
            return 0
        svm.alphas[j] -= svm.labelMat[j]*(e_i - Ej)/eta
        svm.alphas[j] = alpha_prune(svm.alphas[j],top,bottom)
        e_updator(svm, j)
        if (abs(svm.alphas[j] - al_J_old) < svm.s):
#              print("j not moving enough")
            return 0
        svm.alphas[i] += svm.labelMat[j]*svm.labelMat[i]*(al_J_old - svm.alphas[j])
        e_updator(svm, i)
        b1 = svm.b - e_i- svm.labelMat[i]*(svm.alphas[i]-al_I_old)*svm.Ker[i,i] - svm.labelMat[j]*(svm.alphas[j]
        b2 = svm.b - Ej- svm.labelMat[i]*(svm.alphas[i]-al_I_old)*svm.Ker[i,j]- svm.labelMat[j]*(svm.alphas[j]-a
        if (0 < svm.alphas[i] <svm.Cons):
            svm.b = b1
        elif (0 < svm.alphas[j] <svm.Cons):
            svm.b = b2
        else:
            svm.b = (b1 + b2)/2.0
        return 1
    else:
        return 0


def smoP(feature_data, data_type, C, stoper, m_num,ker_type):
    svm = data_struct(np.mat(feature_data),np.mat(data_type).transpose(),C,stoper, ker_type)
    num = 0
    whole = True
    al_change = 0
    while (num < m_num) and ((al_change > 0) or (whole)):
        al_change = 0
        if whole:
            for i in range(svm.row):
                al_change = al_change + alphas_opt(i,svm)
#                  print("fullSet, iter: %d i:%d, pairs changed %d" % (iter,i,alphaPairsChanged))
            num = num + 1
        else:
            n_dound = nonzero((svm.alphas.A > 0) * (svm.alphas.A < C))[0]
            for i in n_dound:
                al_change += alphas_opt(i,svm)
#                  print("non-bound, iter: %d i:%d, pairs changed %d" % (iter,i,alphaPairsChanged))
            num = num + 1
        if whole:
            whole = False
        elif (al_change == 0):
            whole = True
#          print("iteration number: %d" % iter)
    return svm

def plot_svm(svm):
    if svm.feature.shape[1] != 2:
        print("Sorry! I can not draw because the dimension of your data is not 2!")
        return 1

    # draw all samples
    for i in range(svm.row):
        if svm.labelMat[i] == -1:
            plt.plot(svm.feature[i, 0], svm.feature[i, 1], 'ob')
        elif svm.labelMat[i] == 1:
            plt.plot(svm.feature[i, 0], svm.feature[i, 1], 'og')

    non_zero_alphas = np.array(svm.alphas)[np.nonzero(svm.alphas)[0]]
    support_vectors = svm.feature[np.nonzero(svm.alphas)[0]]

    y=np.array(svm.labelMat)[np.nonzero(svm.alphas)].T
```

```
    plt.scatter([support_vectors[:,0]],[support_vectors[:,1]],s=100,c='r',alpha=0.5,marker='o')

def train_svm(train_dataset, ker_type):
    train_x, train_y = read_data(train_dataset)
    svm = smoP(train_x, train_y, 1, 0.0001, 10000, ker_type)
    b = svm.b
    alphas = svm.alphas
    datMat = np.mat(train_x)
    labelMat = np.mat(train_y).transpose()
    svInd = nonzero(alphas)[0]
    sVs = datMat[svInd]
    labelSV = labelMat[svInd]
#     print("there are %d Support Vectors" % np.shape(sVs)[0])
    m,n = np.shape(datMat)
    errorCount = 0
    for i in range(m):
        kernelEval = kernel_calc(sVs,datMat[i,:],ker_type)
        predict=kernelEval.T * np.multiply(labelSV,alphas[svInd]) + b
        if sign(predict)!=sign(train_y[i]):
            errorCount += 1
#     print("the training error rate is: %f" % (float(errorCount)/m))
    return svm

def test_svm(test_dataset, svm, ker_type):
    test_x, test_y = read_data(test_dataset)
#     print(test_y)
    b = svm.b
    alphas = svm.alphas
    datMat = np.mat(test_x)
    labelMat = np.mat(test_y).transpose()
    svInd = nonzero(alphas)[0]
    sVs = datMat[svInd]
    labelSV = labelMat[svInd]
    print("there are %d Support Vectors" % np.shape(sVs)[0])
    errorCount_test = 0
    datMat_test=mat(test_x)
    labelMat = mat(test_y).transpose()
    m,n = np.shape(datMat_test)
    for i in range(m):
        kernelEval = kernel_calc(sVs,datMat_test[i,:],ker_type)
        predict=kernelEval.T * np.multiply(labelSV,alphas[svInd]) + b
        if sign(predict)!=sign(test_y[i]):
            errorCount_test += 1
#         print(sign(predict), sign(test_y[i]))
    print("the test error rate is: %f" % (float(errorCount_test)/m))


#主程序
def main():
    # 1. train svm in 10-cross-validation
    print("_________ train svm with 10-cross-validation _________")
    print("__________ With the parameter C of 1 ________")
    test_data=new_dataset[9]
    for i in range(0,9):
        train_data= new_dataset[i]
        svm_classifier = train_svm(train_data, ('rbf', 1.3))
    print("####### Test #######")
    test_svm(test_data, svm_classifier, ('rbf', 1.3))
    plot_svm(svm_classifier)
    print("\n")

if __name__=='__main__':
    main()
```

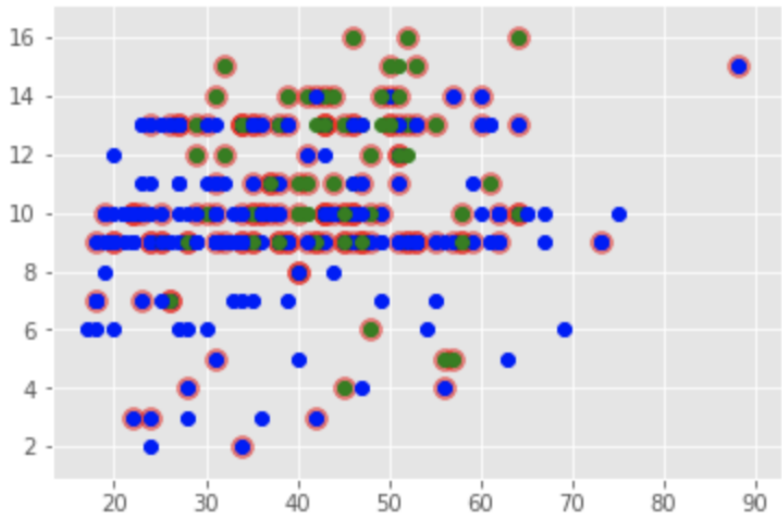
```
_________ train svm with 10-cross-validation _________
__________ With the parameter C of 1 ________
####### Test #######
there are 85 Support Vectors
the test error rate is: 0.352159
```



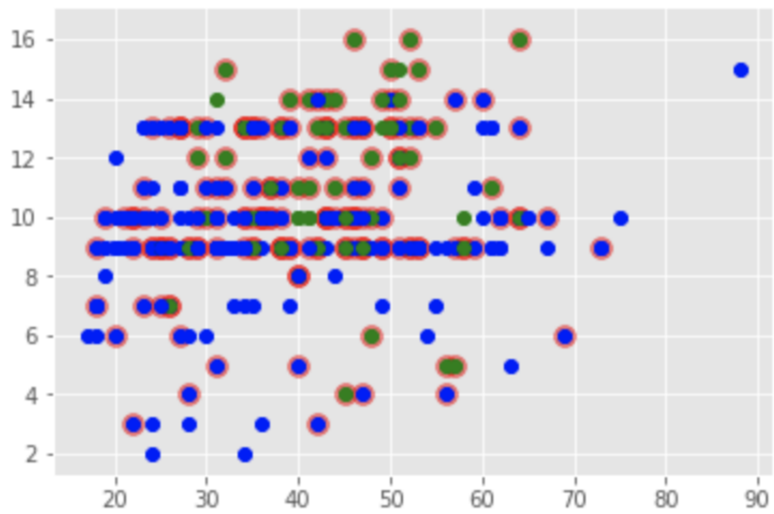
I plot the data which is lower than 50k in blue, the data larger than 50k in green. I did not find a way to draw the decision boundary, while I found the support vectors and made them circled with red shadow.

b. Change the C parameters from small to larger values. Report your observations on how the value of C would affect SVM's performance.

```
__________ With the parameter C of 10 ________
_________ train svm with 10-cross-validation _________
####### Test #######
there are 155 Support Vectors
the test error rate is: 0.176080
```




```
_________ train svm with 10-cross-validation _________
__________ With the parameter C of 100 ________
####### Test #######
there are 174 Support Vectors
the test error rate is: 0.225914
```



In this part, as I change the C from 1 to 10 and 100, respectively, we can see with the C grows, the number of support vectors increases which surely makes the decision boundary shrink. We can also see the error rate grows, so we can have a conclusion that with a bigger C, the accuracy of the svm become worse.

c. Train the SVM using all the features

```
_________ train svm with 10-cross-validation _________
__________ With the parameter C of 1 ________
####### Test #######
there are 155 Support Vectors
The test accuracy is: 0.528239
```

As we can see from the result, the accuray is only about 52.8%, which is low. This is because there are too many features, which makes it overfitting.

3. Implement a kernel SVM

a. Compare the performance (precision, recall, f1-score, and variance) of different kernels: Linear, RBF, and polynomial.

In [506]:

```
from numpy import *
np.seterr(divide='ignore', invalid='ignore')

def alpha_prune(a, top, bottom):
    if a > top:
        a = top
    if bottom > a:
        a = bottom
    return a

def e_calc(svm, i):
    matrix = np.multiply(svm.alphas,svm.labelMat).T
    temp = matrix * svm.Ker[:,i] + svm.b
    f = float(temp)
    e = f - float(svm.labelMat[i])
    return e

# load dataset
def read_data(dataset):
    dataset = pd.DataFrame(dataset)
    col_length = dataset.columns.size
    features = dataset.iloc[:, [0, 3]].apply(pd.to_numeric, errors='ignore')
    label = dataset.iloc[:, -1].apply(pd.to_numeric, errors='ignore').values
    return features, label

def kernel_calc(tree, data_mat, K_type):
    row, col = np.shape(tree)
    # k is a matrix filled with zeros
    Ker = np.mat(np.zeros((row,1)))
    if K_type[0]=='linear': # linear kernel
        Ker = tree * data_mat.T
    elif K_type[0]=='rbf': # gaussian kernel
        for i in range(row):
            d = tree[i,:] - data_mat
            Ker[i] = d * d.T
        Ker = np.exp(Ker/(-1*K_type[1]**2))
    else:
        raise NameError('Kernel Type not included')
    return Ker

def random_select(a, b):
    num = a
    while (num == a):
        num = int(random.uniform(0,b))
    return num

class data_struct:
    def __init__(self, feature_data, data_type, Cons, stoper, ker_type):
        self.b = 0
        self.s = stoper
        self.feature = feature_data
        self.row = np.shape(feature_data)[0]
        self.labelMat = data_type
        self.Cons = Cons
        self.alphas = np.mat(np.zeros((self.row,1)))
        self.eCache = np.mat(np.zeros((self.row,2)))
        self.Ker = np.mat(np.zeros((self.row,self.row)))
        for i in range(self.row):
            temp = kernel_calc(self.feature, self.feature[i,:], ker_type)
            self.Ker[:,i] = temp


def aj_selector(i, svm, Ei):
    k_m = -1
    del_e_m = 0
    e = 0
    svm.eCache[i] = [1,Ei]
    temp = svm.eCache[:,0].A
    e_list = nonzero(temp)[0]
    e_length = len(e_list)
    if (e_length) > 1:
        for e_k in e_list:
            if e_k == i:
                continue
            Ek = e_calc(svm, e_k)
            deltaE = abs(Ei - Ek)
            if (deltaE > del_e_m):
                k_m = e_k
                del_e_m = deltaE
                e = Ek
        return k_m, e
    else:
        j = random_select(i, svm.row)
        e = e_calc(svm, j)
    return j, e


def e_updator(svm, i):
```

```python
    e = e_calc(svm, i)
    svm.eCache[i] = [1,e]

def alphas_opt(i, svm):
    e_i = e_calc(svm, i)
    if ((svm.labelMat[i]*e_i < -svm.s) and (svm.alphas[i] < svm.Cons)) or ((svm.labelMat[i]*e_i > svm.s) and (sv
        j,Ej = aj_selector(i, svm, e_i)
        al_I_old = svm.alphas[i].copy()
        al_J_old = svm.alphas[j].copy()
        if (svm.labelMat[i] != svm.labelMat[j]):
            var = svm.alphas[j] - svm.alphas[i]
            bottom = max(0, var)
            temp = svm.Cons + svm.alphas[j] - svm.alphas[i]
            top = min(svm.Cons, temp)
        else:
            temp = svm.alphas[j] + svm.alphas[i] - svm.Cons
            bottom = max(0, temp)
            top = min(svm.Cons, svm.alphas[j] + svm.alphas[i])
        if bottom == top:
#              print("L==H")
            return 0
        eta = 2.0 * svm.Ker[i,j] - svm.Ker[i,i] - svm.Ker[j,j]
        if eta >= 0:
#              print("eta>=0")
            return 0
        svm.alphas[j] -= svm.labelMat[j]*(e_i - Ej)/eta
        svm.alphas[j] = alpha_prune(svm.alphas[j],top,bottom)
        e_updator(svm, j)
        if (abs(svm.alphas[j] - al_J_old) < svm.s):
#              print("j not moving enough")
            return 0
        svm.alphas[i] += svm.labelMat[j]*svm.labelMat[i]*(al_J_old - svm.alphas[j])
        e_updator(svm, i)

        b1 = svm.b - e_i- svm.labelMat[i]*(svm.alphas[i]-al_I_old)*svm.Ker[i,i] - svm.labelMat[j]*(svm.alphas[j]
        b2 = svm.b - Ej- svm.labelMat[i]*(svm.alphas[i]-al_I_old)*svm.Ker[i,j]- svm.labelMat[j]*(svm.alphas[j]-a
        if (0 < svm.alphas[i] <svm.Cons):
            svm.b = b1
        elif (0 < svm.alphas[j] <svm.Cons):
            svm.b = b2
        else:
            svm.b = (b1 + b2)/2.0
        return 1
    else:
        return 0


def smoP(feature_data, data_type, C, stoper, m_num,ker_type):
    svm = data_struct(np.mat(feature_data),np.mat(data_type).transpose(),C,stoper, ker_type)
    num = 0
    whole = True
    al_change = 0
    while (num < m_num) and ((al_change > 0) or (whole)):
        al_change = 0
        if whole:
            for i in range(svm.row):
                al_change = al_change + alphas_opt(i,svm)
#                  print("fullSet, iter: %d i:%d, pairs changed %d" % (iter,i,alphaPairsChanged))
            num = num + 1
        else:
            n_dound = nonzero((svm.alphas.A > 0) * (svm.alphas.A < C))[0]
            for i in n_dound:
                al_change += alphas_opt(i,svm)
#                  print("non-bound, iter: %d i:%d, pairs changed %d" % (iter,i,alphaPairsChanged))
            num = num + 1
        if whole:
            whole = False
        elif (al_change == 0):
            whole = True
#          print("iteration number: %d" % iter)
    return svm

def plot_svm(svm):
    if svm.feature.shape[1] != 2:
        print("Sorry! I can not draw because the dimension of your data is not 2!")
        return 1

    # draw all samples
    for i in range(svm.row):
        if svm.labelMat[i] == -1:
            plt.plot(svm.feature[i, 0], svm.feature[i, 1], 'ob')
        elif svm.labelMat[i] == 1:
            plt.plot(svm.feature[i, 0], svm.feature[i, 1], 'og')

    non_zero_alphas = np.array(svm.alphas)[np.nonzero(svm.alphas)[0]]
    support_vectors = svm.feature[np.nonzero(svm.alphas)[0]]

    y=np.array(svm.labelMat)[np.nonzero(svm.alphas)].T
    plt.scatter([support_vectors[:,0]],[support_vectors[:,1]],s=100,c='r',alpha=0.5,marker='o')
```

```python
def train_svm(train_dataset, ker_type):
    train_x, train_y = read_data(train_dataset)
    svm = smoP(train_x, train_y, 1, 0.0001, 10000, ker_type)
    b = svm.b
    alphas = svm.alphas
    datMat = np.mat(train_x)
    labelMat = np.mat(train_y).transpose()
    svInd = nonzero(alphas)[0]
    sVs = datMat[svInd]
    labelSV = labelMat[svInd]
#     print("there are %d Support Vectors" % np.shape(sVs)[0])
    m,n = np.shape(datMat)
    errorCount = 0
    for i in range(m):
        kernelEval = kernel_calc(sVs,datMat[i,:],ker_type)
        predict=kernelEval.T * np.multiply(labelSV,alphas[svInd]) + b
        if sign(predict)!=sign(train_y[i]):
            errorCount += 1
#     print("the training error rate is: %f" % (float(errorCount)/m))
    return svm

def test_svm(test_dataset, svm, ker_type):
    test_x, test_y = read_data(test_dataset)
#     print(test_y)
    b = svm.b
    alphas = svm.alphas
    datMat = np.mat(test_x)
    labelMat = np.mat(test_y).transpose()
    svInd = nonzero(alphas)[0]
    sVs = datMat[svInd]
    labelSV = labelMat[svInd]
    print("there are %d Support Vectors" % np.shape(sVs)[0])
    errorCount_test = 0
    datMat_test=mat(test_x)
    labelMat = mat(test_y).transpose()
    m,n = np.shape(datMat_test)
    for i in range(m):
        kernelEval = kernel_calc(sVs,datMat_test[i,:],ker_type)
        predict=kernelEval.T * np.multiply(labelSV,alphas[svInd]) + b
        if sign(predict)!=sign(test_y[i]):
            errorCount_test += 1
#         print(sign(predict), sign(test_y[i]))
    print("the test error rate is: %f" % (float(errorCount_test)/m))


def main():
    # 1. train svm in 10-cross-validation
    print("_________ Linear Kernel _________")
    print("__________ With the parameter C of 1 ________")
    test_data=new_dataset[9]
    for i in range(0,9):
        train_data= new_dataset[i]
        svm_classifier = train_svm(train_data, ('linear', 1.3))
    print("####### Test #######")
    test_svm(test_data, svm_classifier, ('linear', 1.3))
#     plot_svm(svm_classifier)
    print("\n")

if __name__=='__main__':
    main()
```

```
_________ Linear Kernel _________
__________ With the parameter C of 1 ________
####### Test #######
there are 172 Support Vectors
the test error rate is: 0.252492
```

As we can see from the result, for linear kernel, the test error is 0.25492 while the error of rbf kernel(tested in the second section) is 0.176080. We can have a conclusion that the performance of rbf kernel is better than linear kernel

Reference

Some of the analysis and code learned from the following website:

https://blog.csdn.net/csqazwsxedc/article/details/71513197 (https://blog.csdn.net/csqazwsxedc/article/details/71513197)

https://github.com/HYL13/SimpleMachineLearning/blob/master/SVM.py (https://github.com/HYL13/SimpleMachineLearning/blob/master/SVM.py)

https://blog.csdn.net/shuiyungtiang/article/details/50996229 (https://blog.csdn.net/shuiyungtiang/article/details/50996229)

In [ ]: