Exercise 4 - Support Vector Machines

May 20, 2019

by Wajid Ghafoor and Benjamin Ostendorf

0.0.1 1) Download and install libSVM from https://www.csie.ntu.edu.tw/~cjlin/libsvm/

We donwloaded it via the git path. Therfore we cloned the repository and than followed the installation instructions in the Readme file

0.0.2 2) Run the experiments specified in the readme file to test that your installation is correct

[20]:
```
#Tutorial aboout the usage of support vector mashines
y, x = svm_read_problem('heart_scale')
m = svm_train(y[:200], x[:200], '-c 4')
p_label, p_acc, p_val = svm_predict(y[200:], x[200:], m)
```

```
Accuracy = 84.2857% (59/70) (classification)
```

As we can see the installation was successfully and in the above cell it predicted with an accuracy of 84,2857%

0.0.3 (3) Apply support vector machines to classify the Chagas parasite images using thefeature vectors that you generated in exercise 3. You need to run experiments using the linear, polynomial, sigmoid and radial basis kernels.

[21]:
```
# Imports for exercise 4!
from PIL import Image, ImageColor
import numpy as np
from PIL import ImageFilter
import math as m
from functools import reduce
from svmutil import *
from itertools import zip_longest
```

[22]:
```
# All the features used in exercise 3
def extrema_red(image):
    return image.getextrema()[0][0]
def extrema_green(image):
    return image.getextrema()[1][0]
```

```python
def variance_red(image):
    return np.var(image.split()[0])
def variance_green(image):
    return np.var(image.split()[1])
def mean_blue(image):
    return np.mean(image.split()[2])
```

[23]:

```python
#create feature 5-tuple into a list!
def create_feature(image):
    er = extrema_red(image)
    eg = extrema_green(image)
    vr = variance_red(image)
    vg = variance_green(image)
    mb = mean_blue(image)
    return [er,eg,vr,vg,mb]

def create_features(img_list):
    return list(map(create_feature, img_list))
```

[24]:

```python
#Read the images and create features for positive and negative ones
pos_image_list = [Image.open("positives/p0" + str(i) + ".png") if i < 10 else Image.open("positives/p" + str(i) + ".png")
                  for i in range(1,31)]
neg_image_list = [Image.open("negatives/n0" + str(i) + ".png") if i < 10 else Image.open("negatives/n" + str(i) + ".png")
                  for i in range(1,31)]

positive_features_full = create_features(pos_image_list)
negative_features_full = create_features(neg_image_list)

#split dataset into traning and testing
positive_features_half = create_features(pos_image_list[0:15])
negative_features_half = create_features(neg_image_list[0:15])
```

The pair “<index>:<value>” gives a feature (attribute) value: is an integer starting from 1 and is a real number. The only exception is the precomputed kernel, where starts from 0; see the section of precomputed kernels. Indices must be in ASCENDING order.

[25]:

```python
# create from the feature_list a LIst with dictionarys with the form for each list: list[f1,f2,...,f_n] -> dict[1:f1,2:f2,...,n:f_n]
def prepare_feature_dict(features):

        return [dict(zip_longest(*[iter([1,p[0],2,p[1],3,p[2],4,p[3],5,p[4]])] * 2, fillvalue="")) for p in features]
```

[26]:

```python
# specify the length of training and test set
train = 25
```

```
test = 5
```

[27]:
```
# create labels,train and test set for support vector mashines!
feature_dict_pos_train = prepare_feature_dict(positive_features_full[0:train])
feature_dict_neg_train = prepare_feature_dict(negative_features_full[0:train])
feature_dict_pos_test = prepare_feature_dict(positive_features_full[train:])
feature_dict_neg_test = prepare_feature_dict(negative_features_full[train:])
```

[28]:
```
#concatenate negative and positive features and labels
feature_list_dict_train = feature_dict_pos_train + feature_dict_neg_train
label_list_train = [1 for i in range(train)] + [-1 for i in range(train)]
feature_list_dict_test = feature_dict_pos_test + feature_dict_neg_test
label_list_test = [1 for i in range(test)] + [-1 for i in range(test)]
```

0.0.4 (4) Prepare a report containing your final results.

Kerneltype: 0 – linear: u'*v

[29]:
```
#Also good results about 90 %
#Best Accuracy for train,test =(25,5)
m = svm_train(label_list_train, feature_list_dict_train, '-t 0')
label, acc, val = svm_predict(label_list_test, feature_list_dict_test, m)
```

```
Accuracy = 90% (9/10) (classification)
```

Kerneltype: 1 polynomial: (gammau'v + coef0)^degree

[30]:
```
#This one results in really good prediction rate even if the train set holds of␣
↪1% ?!?!
#Best Accuracy for train,test =(25,5)
m = svm_train(label_list_train, feature_list_dict_train, '-t 1')
label, acc, val = svm_predict(label_list_test, feature_list_dict_test, m)
```

```
Accuracy = 100% (10/10) (classification)
```

Kerneltype: 2 – radial basis function: exp(-gamma*|u-v|^2)

[31]:
```
# Round about a liitle bit more than 50 percent are predicted
m = svm_train(label_list_train, feature_list_dict_train, '-t 2')
label, acc, val = svm_predict(label_list_test, feature_list_dict_test, m)
```

```
Accuracy = 50% (5/10) (classification)
```

Kerneltype: 3 – sigmoid: tanh(gammau'v + coef0)

[32]:
```
# in mean about 50 % Prediction rate
m = svm_train(label_list_train, feature_list_dict_train, '-t 3')
label, acc, val = svm_predict(label_list_test, feature_list_dict_test, m)
```

```
Accuracy = 50% (5/10) (classification)
```