noteLogReg

January 5, 2016

1 Using logistic Regression on Toydata to get a high AMS

```
import numpy as np
import matplotlib.pyplot as plt
import math
from sklearn import linear_model as linMod
```

Data shall have the form of $[w, y, x_1, x_2]$ where

- w is a weight in the intervall $[0, 1)$
- y is the label "0" for "background" or "1" for "signal"
- x_n are randomly generated features with respect to the label

```
def generateFeature(label, mu_s, mu_b, sigma_s=5, sigma_b=5):
    if label is 1:
        mu = mu_s
        sigma = sigma_s
    else:
        mu = mu_b
        sigma = sigma_b
    return np.random.normal(mu,sigma)
```

Approximate Median Significance (AMS) defined as:

$$AMS = \sqrt{2(s + b + b_r)log[1 + (s/(b + b_{reg}))] - s}$$

where

- $b_{reg} = 10$ is a regulization term (set by the contest),
- $b = \sum_{i=1}^{n} w_i, y_i = 0$ is sum of weighted background (incorrectly classified as signal),
- $s = \sum_{i=1}^{n} w_i, y_i = 1$ is sum of weighted signals (correctly classified as signal),
- log is natural logarithm

```
def calcAMS(s,b):
    br = 10.0
    radicand = 2 *( (s+b+br) * math.log (1.0 + s/(b+br)) -s)
    if radicand < 0:
        print('radicand is negative. Exiting')
        exit()
    else:
        return math.sqrt(radicand)
```

```
def calcWeightSums(weights,preds,labels):
    s = 0
    b = 0
    for j in list(range(0,len(preds))):
        pred = preds[j]
        label = labels[j]
        weight = weights[j]
        if pred > 0.:
            if label > 0.:
                s += weight
            else:
                b += weight
    return s,b
```

actually generate data

```
#toydata shall have n vectors with 5 dimensions
n = 100000
#probability for signal-label
s_prob = 0.05
#random values will be used as weights for evaluation later
weights = np.random.rand(n)
labels = np.zeros(n)
x_1 = np.zeros(n)
x_2 = np.zeros(n)

for i in range(0,n):
    if weights[i] <= s_prob:
        label = 1
    else:
        label = 0
    labels[i] = label
    x_1[i]=generateFeature(label,mu_s=5,mu_b=20)
    x_2[i]=generateFeature(label,mu_s=5,mu_b=25)
```

visualize

```
%pylab inline
plt.scatter(x_1, x_2, edgecolor="", c=labels, alpha=0.5)
```

```
def splitList(xList,n):
    aList = xList[:n]
    bList = xList[n:]
    return aList,bList
```

split toydata into training- and testset for the classifier

```
n_train = int(n/10)

train_x_1,test_x_1 = splitList(x_1,n_train)
train_x_2,test_x_2 = splitList(x_2,n_train)
train_labels,test_labels = splitList(labels,n_train)
test_weights = splitList(weights,n_train)[1]
```

For Comparison, we calculate the best possible AMS
(case: every signal correctly detected)

```
def calcMaxAMS(weights,labels):
    s,b = calcWeightSums(weights,labels,labels)
    ams = calcAMS(s,b)
    print("Maximum AMS possible with this Data:", ams)
    return ams
```

```
calcMaxAMS(test_weights,test_labels);
```

we initialize the Logistic Regression Classifier, shape the input-data and fit the model

```
logReg = linMod.LogisticRegression(C=1e5)

train_x = np.array([train_x_1,train_x_2]).transpose()
test_x = np.array([test_x_1,test_x_2]).transpose()
train_labels = np.array(train_labels).transpose()
test_labels = np.array(test_labels).transpose()

logReg.fit(train_x,train_labels)

logReg.sparsify()

predProb = logReg.predict_proba(test_x)
pred = logReg.predict(test_x)
score = logReg.score(test_x,test_labels)

print("Score:", score)
```

```
s,b = calcWeightSums(test_weights,pred,test_labels)
calcAMS(s,b)
```

We successfully tested logistic Regression, now let's use it on actual CERN-Data.

```
import KaggleData;
```

```
csvDict,header = KaggleData.createCsvDictionary()
```

Trainingset has key "t"
Public Testset has key "p" (note: "p" won't work, using private Testset ("v"))

```
def getFeatureSets(featureName):
    trainFeature = KaggleData.getFeatureAsNpArray(
        csvDict,header,featureName,["t"],hasErrorValues = True)
    testFeature = KaggleData.getFeatureAsNpArray(
        csvDict,header,featureName,["v"],hasErrorValues = True)
    return trainFeature, testFeature
```

```
train_eventList,test_eventList = getFeatureSets("EventId")
train_labels,test_labels = getFeatureSets("Label")
test_weights = getFeatureSets("KaggleWeight")[1]
```

We observe the relation Label <=> Weight

```
signal_sum = int(test_labels.cumsum()[-1])
background_sum = int(len(test_labels)-signal_sum)
signal_weight = 0
background_weight = 0
for i in range(0,len(test_labels)):
    if test_labels[i] > 0:
        signal_weight += test_weights[i]
    else:
        background_weight += test_weights[i]
print(background_weight/background_sum)
print(signal_weight/signal_sum)
```

We can observe, that False signals will be weighted a lot heavier than True signals.
If a classifier achieved a higher AMS while detecting less signals,
we can make statements about the usabilty of the features, the classifier used.
We choose features with beneficial properties for classifying.

```
(train_DER_met_phi_centrality,
 test_DER_met_phi_centrality) = getFeatureSets("DER_met_phi_centrality")
(train_DER_pt_ratio_lep_tau,
 test_DER_pt_ratio_lep_tau) = getFeatureSets("DER_pt_ratio_lep_tau")
```

Using DER_mass_MMC was not allowed in the former contest, we use it here anyway to test our classifier

```
(train_DER_mass_MMC,
 test_DER_mass_MMC) = getFeatureSets("DER_mass_MMC")
```

```
train_labels = np.array(train_labels).transpose()
test_labels = np.array(test_labels).transpose()
```

```
calcMaxAMS(test_weights,test_labels)
print("True Signals:",int(test_labels.cumsum()[-1]))
```

We start with one feature and add more with every regression to see improvement of the AMS

```
def logisticReg(train_x,train_labels,test_x,test_labels):
    logReg = None
    logReg = linMod.LogisticRegression(C=1e5)
    logReg.fit(train_x,train_labels)
    logReg.sparsify()
    predProb = logReg.predict_proba(test_x)
    pred = logReg.predict(test_x)
    signals = int(pred.cumsum()[-1])
    print("signals read:", signals)
    if signals is not 0:
        s,b = calcWeightSums(test_weights,pred,test_labels)
        ams = calcAMS(s,b)
    else:
        ams = 0
    print("AMS:",ams)
    return predProb,pred,score
```

```
train_x = np.array(
    [train_DER_met_phi_centrality,
     train_DER_pt_ratio_lep_tau]).transpose()
test_x = np.array(
    [test_DER_met_phi_centrality,
     test_DER_pt_ratio_lep_tau]).transpose()
pred = logisticReg(
    train_x,train_labels,
    test_x,test_labels)[1];
pred.cumsum()
```

```
def logRegFor(fList):
    for feature in fList:
        print("Feature:",feature)
        trainList_x,testList_x = getFeatureSets(feature)
        train_x = np.array([trainList_x]).transpose()
        test_x = np.array([testList_x]).transpose()
        logisticReg(train_x,train_labels,test_x,test_labels)[1];
```

```
(train_PRI_tau_pt,
 test_PRI_tau_pt) = getFeatureSets("PRI_tau_pt")
(train_DER_met_phi_centrality,
 test_DER_met_phi_centrality) = getFeatureSets("DER_met_phi_centrality")
```

```
(train_DER_pt_h,
 test_DER_pt_h) = getFeatureSets("DER_pt_h")
(train_DER_pt_ratio_lep_tau,
 test_DER_pt_ratio_lep_tau) = getFeatureSets("DER_pt_ratio_lep_tau")
(train_DER_mass_transverse_met_lep,
 test_DER_mass_transverse_met_lep) = getFeatureSets("DER_mass_transverse_met_lep")
```

we are able to achieve a higher AMS by adjusting the decision-threshold (around 0.25)

```
def bestThreshold(predProb):
    thresh = 0
    maxAMS = 0
    maxThresh = 0
    for thresh in np.linspace(0.2,1.0,100):
        newPred = np.zeros(len(predProb))
        for i in range(0,len(predProb)):
            if predProb[i][1] > thresh:
                newPred[i]=1
        s,b = calcWeightSums(test_weights,newPred,test_labels)
        ams = calcAMS(s,b)
        if ams > maxAMS:
            maxThresh = thresh
            maxAMS = ams
            signals = int(newPred.cumsum()[-1])
    print("Maximum AMS:",maxAMS, "with threshold", maxThresh)
    print("Signals read:", signals)
```

```
train_x = np.array(
    [train_PRI_tau_pt,
     train_DER_met_phi_centrality,
     train_DER_pt_h,
     train_DER_pt_ratio_lep_tau]).transpose()
test_x = np.array(
    [test_PRI_tau_pt,
     test_DER_met_phi_centrality,
     test_DER_pt_h,
     test_DER_pt_ratio_lep_tau]).transpose()
(predProb,
 pred) = logisticReg(
    train_x,
    train_labels,
    test_x,
    test_labels)[0:2];
bestThreshold(predProb)
```

```
train_x = np.array(
    [train_PRI_tau_pt,
     train_DER_met_phi_centrality]).transpose()
test_x = np.array(
    [test_PRI_tau_pt,
     test_DER_met_phi_centrality]).transpose()
predProb,pred = logisticReg(
    train_x,
    train_labels,
    test_x,
    test_labels)[0:2];
bestThreshold(predProb)
```

```
train_x = np.array(
    [train_DER_met_phi_centrality,
     train_DER_pt_ratio_lep_tau]).transpose()
```

```
test_x = np.array(
    [test_DER_met_phi_centrality,
     test_DER_pt_ratio_lep_tau]).transpose()
predProb,pred = logisticReg(
    train_x,train_labels,
    test_x,
    test_labels)[0:2];
bestThreshold(predProb)
```

```
train_x = np.array(
    [train_DER_met_phi_centrality,
     train_PRI_tau_pt]).transpose()
test_x = np.array(
    [test_DER_met_phi_centrality,
     test_PRI_tau_pt]).transpose()
predProb,pred = logisticReg(
    train_x,
    train_labels,
    test_x,test_labels)[0:2];
```