INSTITUT FÜR INFORMATIK
Computer Vision, Computer Graphics
and Pattern Recognition

Universitätsstr. 1 D–40225 Düsseldorf

HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

Classification of data from the ATLAS experiments

Michael Janschek

Bachelor Thesis

Beginn der Arbeit:	10. Dezember 2015
Abgabe der Arbeit:	10. MÃd'rz 2016
Gutachter:	Prof. Dr. Stefan Harmeling
	Prof. Dr. Stefan Conrad

Erklärung

Hiermit versichere ich, dass ich diese Bachelor Thesis selbstständig verfasst habe. Ich habe dazu keine anderen als die angegebenen Quellen und Hilfsmittel verwendet. Alle Stellen, die aus den Quellen entnommen wurden, sind als solche kenntlich gemacht worden. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Düsseldorf, den 10. MÃd'rz 2016

Michael Janschek

Abstract

Hier kommt eine ca. einseitige Zusammenfassung der Arbeit rein.

Contents

1 Introduction

This chapter first presents the Higgs Boson Machine Learning Challenge and explains its motivation and goals. It is concluded by an overview of the thesis structure.

1.1 The Higgs Boson Machine Learning Challenge

Kaggle is an internet community of data scientists, it hosts several competitions posed by businesses or organizations. Further services involve open datasets, a "Jobs Board" and "Kaggle Rankings", a scoreboard based on performances of community-members in Kaggles competitions.

1.1.1 Motivation

1.1.2 Goal

> The goal of the Higgs Boson Machine Learning Challenge is to explore the potential of advanced machine learning methods to improve the discovery significance of the experiment. No knowledge of particle physics is required. Using simulated data with features characterizing events detected by ATLAS, your task is to classify events into "tau tau decay of a Higgs boson" versus "background." [higb]

1.2 Overview

In Chapter 2, we will derive the formal problem from the challenges task. We will understand the evaluation metric and finish the chapter by analysing the dataset [higa].

Chapter 3 will use first knowledge about the data to choose simple approaches for classification. After these we will describe several more specific and complex methods.

In Chapter 4 we will use the discussed methods and observe their performance on the challenges data. The thesis closes with a discussion about the approaches and their possible influence on other HEP-applications.

2 Understanding the Challenge

2.1 The formal problem

2.2 The Data

2.3 The Evaluation

```
import numpy as np
import matplotlib.pyplot as plt
import math
```

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:
        ams = math.sqrt(radicand)
        print("AMS:", ams)
        return ams
```

Following this definition, we can derive a maximum AMS by simply summing the weights of all positive labels.

```
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
```

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

```python
def calcMaxAMS(weights,labels):
    s,b = calcWeightSums(weights,labels,labels)
    ams = calcAMS(s,b)
    print("Found", int(labels.cumsum()[-1]), "signals.")
    print("Weightsums signal:", s, "| background:", b)
    print("Maximum AMS possible with this Data:", ams)
    return ams
```

We generate AMS with good seperable toy-data, starting with the maximum AMS. The data of the actual challenge is weighted to punish wrong-identified signals significantly harder than wrong background. Our toy-data will do so by using its signal-probability as weight, the features are randomized by normal distributions.

```python
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)
```

```python
def createToyData(n = 100,dim = 3,s_prob = 0.05):
    data= np.zeros(shape = (n,dim),dtype=float)
    if dim < 3:
        print("Operation canceled.",
              "Data should have at least one",
              "additional dimension besides weights and labels.",
              "(dim >=3)")
        return None
    data[:,0] = np.random.rand(n) #weights
    for i in range(0,n):
        if data[i,0] <= s_prob: # label-determination
            label = 1
        else:
            label = 0
        data[i,1] = label
        for j in range(2,dim):
            #mu_s=j*5
            #mu_b=j*20
            data[i,j]=generateFeature(label,mu_s=(j-1)*5,mu_b=(j-1)*20)
    return data
```

```
n = 100000
prob = 0.05
data = createToyData(n,dim=10,s_prob=prob)
```

```
weights = data[:,0]
labels = data[:,1]
calcMaxAMS(weights,labels);
```

```
AMS: 21.19484139384361
Found 5020 signals.
Weightsums signal: 124.326027277 | background: 0
Maximum AMS possible with this Data: 21.19484139384361
```

We randomly guess labels for a solution for a second AMS with knowledge about the toydatas signal-probability.

```
sol_weights = np.random.rand(n)
sol = np.zeros(n)
for i in range(0,n):
    if sol_weights[i] <= prob: # label-determination
        sol[i] = 1
    else:
        sol[i] = 0
```

```
s,b = calcWeightSums(weights,sol,labels)
br = 10
```

```
s*=10
```

```
b*=10
```

```
s+b+br
```

```
25055.584526883151
```

```
1.0 + s/(b+br)
```

```
1.002459486418638
```

```
math.log(1.0 + s/(b+br))
```

```
0.002456466831991091
```

```
math.sqrt( 2 *( (s+b+br) * math.log (1.0 + s/(b+br)) -s) )
```

```
0.38867392440571713
```

```
calcAMS(s,b);
```

```
AMS: 0.38867392440571713
```

```

```

3 Methods of classification

3.1 Logistic regression

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

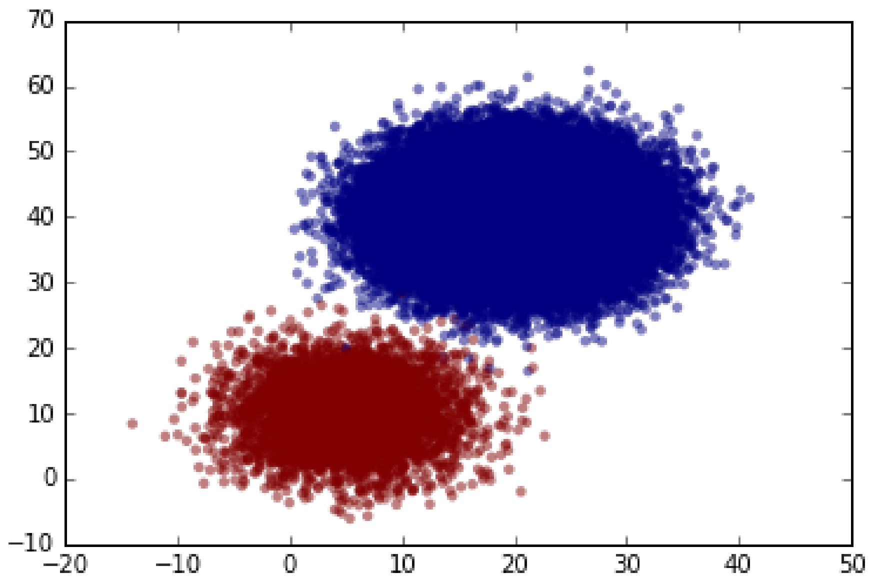
We generate toydata

```
#toydata shall have n vectors with 5 dimensions
n = 100000
#probability for signal-label
s_prob = 0.05
dim = 4
data = tb.createToyData(n,dim,s_prob)
weights = data[:,0]
labels = data[:,1]
x_1 = data[:,2]
x_2 = data[:,3]
```

visualize

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

```
Populating the interactive namespace from numpy and matplotlib
<matplotlib.collections.PathCollection at 0x57b5160>
```



split toydata into training- and testset for the classifier

```
n_train = int(n/10)

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

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

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

```
Found 4474 signals.
Weightsums signal: 112.519230558 | background: 0
Maximum AMS possible with this Data: 19.721820983450662
```

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)
```

```
Score: 0.999833333333
```

```
s,b = tb.calcWeightSums(test_weights,pred,test_labels)
print("AMS:",tb.calcAMS(s,b))
```

```
AMS: 18.392472651997792
```

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

```
import kaggleData as kD;
```

```
csvDict,header = kD.createCsvDictionary();
```

Trainingset has key "t"
Public Testset has key "b"
Private Testset has key "v"

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

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

How are labels and weights related?

```
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("Mean of background-weights:", background_weight/background_sum)
print("Mean of signal-weights:",signal_weight/signal_sum)
```

```
Mean of background-weights: 1.38702756616
Mean of signal-weights: 0.00450270106462
```

We observe that False signals are 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()
```

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

```
Found 153683 signals.
Weightsums signal: 691.988607714 | background: 0
Maximum AMS possible with this Data: 67.71112289514173
```

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])
    return predProb,pred,score
```

```
def logRegFor(fList):
    for feature in fList:
        print("Feature:",feature)
        trainList_x,testList_x = kD.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

```
def customThreshold(predProb,t = 0.5):
    newPred = np.zeros(len(predProb))
    for i in range(0,len(predProb)):
        if predProb[i][1] > t:
            newPred[i]=1
    return newPred
```

```
def bestThreshold(predProb):
    bestPred = predProb
    thresh = 0
    maxAMS = 0
    maxThresh = 0
    newSignals = 0
    for thresh in np.linspace(1.0,0.0,100):
        newPred = customThreshold(predProb,thresh)
        s,b = tb.calcWeightSums(test_weights,newPred,test_labels)
        #print("s:",s,"b:",b)
        ams = tb.calcAMS(s,b)
        #print("ams:",ams)
        if ams > maxAMS:
            bestPred = newPred
            maxThresh = thresh
            maxAMS = ams
            newSignals = int(newPred.cumsum()[-1])
    #print("Maximum AMS:",maxAMS, "with threshold", maxThresh)
    #print("Signals read:", newSignals)
    return bestPred,maxAMS,maxThresh,newSignals
```

```
def compareBinaryArrays(a,b):
    if shape(a) != shape(b):
        print("ERROR: Arrays must have same shape.")
        return None
    eq_total = 0
    eq_ones = 0
    eq_zeros = 0
    for i in range(0,len(a)):
        if a[i] == b[i]:
            eq_total += 1
            if a[i] == 1:
                eq_ones += 1
            else:
                eq_zeros += 1
    return eq_total,eq_ones,eq_zeros
```

```
def fullEvaluation(predProb,pred,test_weights,test_labels):
    signals = pred.cumsum()[-1]
    correct = np.equal(pred,test_labels).cumsum()[-1]
    s,b = tb.calcWeightSums(test_weights,pred,test_labels)
    ams = tb.calcAMS(s,b)

    newPred,maxAMS,maxThresh,newSignals = bestThreshold(predProb)
    newcorrect = np.equal(newPred,test_labels).cumsum()[-1]

    T_eq1,T_eq0 = compareBinaryArrays(test_labels,test_labels)[1:]
    o_eqt,o_eq1,o_eq0 = compareBinaryArrays(pred,test_labels)
    a_eqt,a_eq1,a_eq0 = compareBinaryArrays(newPred,test_labels)

    print("Signals in test-data:", test_labels.cumsum()[-1])
    print("Comparison of [o]riginal and [a]djusted predictions:\n"+
          " - [o]Signals read:", signals,"\n"+
          " - [a]Signals read:", newSignals,"\n"+
          " -- Difference:", (signals-newSignals),"\n"+
          " - [o]Correct labels:", o_eqt,"| signals:", o_eq1,"| background:",o_eq0,"\n"+
          " ----- wrong signals:", (T_eq1-o_eq1), "| background:", (T_eq0-o_eq0),"\n"+
          " - [a]Correct labels:", a_eqt,"| signals:", a_eq1,"| background:",a_eq0,"\n"+
          " ----- wrong signals:", (T_eq1-a_eq1), "| background:", (T_eq0-a_eq0),"\n"+
          " -- Difference labels:", (a_eqt-o_eqt),"| signals:", (a_eq1-o_eq1),"| background:"
          " - [o]AMS:", ams,"\n"+
          " - [a]AMS:", maxAMS,"(threshold =",maxThresh,")\n"+
          " -- Difference:", (ams-maxAMS),"\n"
         )
    return pred,newPred
```

```
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];
fullEvaluation(predProb,pred,test_weights,test_labels);
```

```
Signals in test-data: 153683.0
Comparison of [o]riginal and [a]djusted predictions:
 - [o]Signals read: 88314.0
 - [a]Signals read: 270377
 -- Difference: -182063.0
 - [o]Correct labels: 315491 | signals: 53744 | background: 261747
 ----- wrong signals: 99939 | background: 34570
 - [a]Correct labels: 275182 | signals: 124621 | background: 150561
 ----- wrong signals: 29062 | background: 145756
 -- Difference labels: -40309 | signals: 70877 | background: -111186
 - [o]AMS: 1.3192322666968355
 - [a]AMS: 1.4167498277657031 (threshold = 0.252525252525 )
 -- Difference: -0.09751756106886766
```

```
print("[o] correct signal/correct labels:",(53744/315491))
print("[o] correct background/correct labels:",(261747/315491))
print("[a] correct signal/correct labels:",(124621/275182))
print("[a] correct background/correct labels:",(150561/275182))
print("")
print("[o] wrong signal/wrong labels:",(99939/(len(test_labels)-315491)))
print("[o] wrong background/wrong labels:",(34570/(len(test_labels)-315491)))
print("[a] wrong signal/wrong labels:",(29062/(len(test_labels)-275182)))
print("[a] wrong background/wrong labels:",(145756/(len(test_labels)-275182)))
```

```
[o] correct signal/correct labels: 0.17035034279900219
[o] correct background/correct labels: 0.8296496572009978
[a] correct signal/correct labels: 0.4528675567442638
[a] correct background/correct labels: 0.5471324432557362

[o] wrong signal/wrong labels: 0.7429911753116892
[o] wrong background/wrong labels: 0.2570088246883108
[a] wrong signal/wrong labels: 0.16624146254962305
[a] wrong background/wrong labels: 0.833758537450377
```

By adjusting the threshold, we raise the rate of correct signals at the expense of correctly predicted background-events. This results in a higher AMS even though we softened our decision-threshold

```
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];
fullEvaluation(predProb,pred,test_weights,test_labels);
```

```
Signals in test-data: 153683.0
Comparison of [o]riginal and [a]djusted predictions:
 - [o]Signals read: 70612.0
 - [a]Signals read: 299182
 -- Difference: -228570.0
 - [o]Correct labels: 313525 | signals: 43910 | background: 269615
 ----- wrong signals: 109773 | background: 26702
 - [a]Correct labels: 259123 | signals: 130994 | background: 128129
 ----- wrong signals: 22689 | background: 168188
 -- Difference labels: -54402 | signals: 87084 | background: -141486
 - [o]AMS: 1.1512119152385953
 - [a]AMS: 1.3397092760005813 (threshold = 0.222222222222 )
 -- Difference: -0.18849736076198598
```

```
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];
fullEvaluation(predProb,pred,test_weights,test_labels);
```

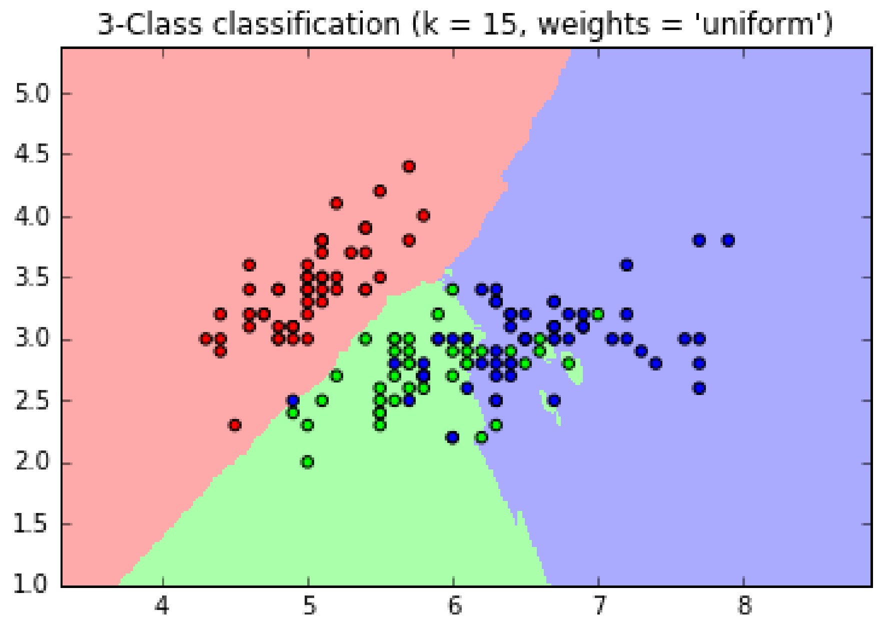
```
Signals in test-data: 153683.0
Comparison of [o]riginal and [a]djusted predictions:
 - [o]Signals read: 105546.0
 - [a]Signals read: 266339
 -- Difference: -160793.0
 - [o]Correct labels: 311453 | signals: 60341 | background: 251112
 ----- wrong signals: 93342 | background: 45205
 - [a]Correct labels: 273910 | signals: 121966 | background: 151944
 ----- wrong signals: 31717 | background: 144373
 -- Difference labels: -37543 | signals: 61625 | background: -99168
```

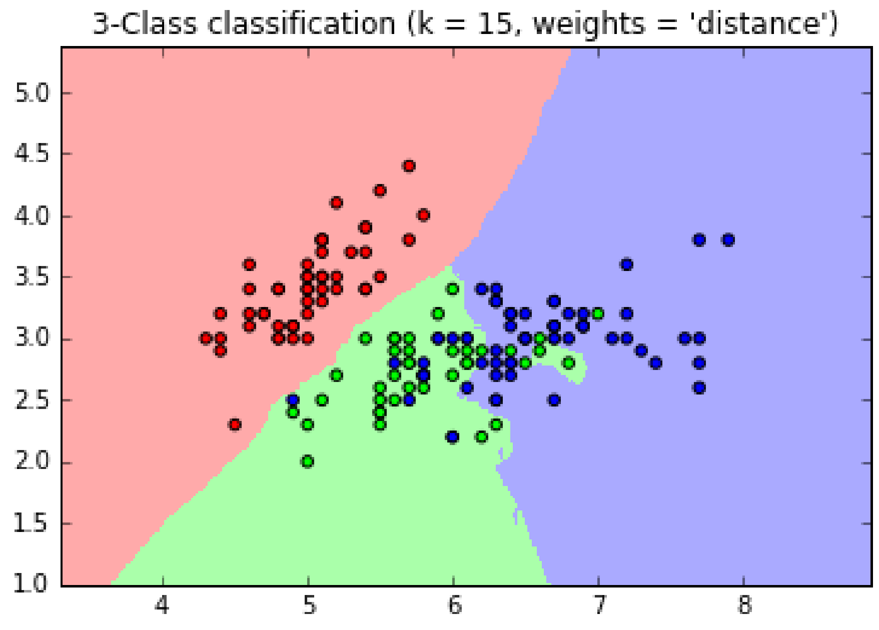
```
- [o]AMS: 1.3069521546244742
- [a]AMS: 1.4054130567873595 (threshold = 0.262626262626 )
-- Difference: -0.09846090216288528
```

3.2 k-nn classification

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap
from sklearn import neighbors, datasets


%pylab inline
n_neighbors = 15

# import some data to play with
iris = datasets.load_iris()
X = iris.data[:, :2]  # we only take the first two features. We could
                      # avoid this ugly slicing by using a two-dim dataset
y = iris.target

h = .02  # step size in the mesh

# Create color maps
cmap_light = ListedColormap(['#FFAAAA', '#AAFFAA', '#AAAAFF'])
cmap_bold = ListedColormap(['#FF0000', '#00FF00', '#0000FF'])

for weights in ['uniform', 'distance']:
    # we create an instance of Neighbours Classifier and fit the data.
    clf = neighbors.KNeighborsClassifier(n_neighbors, weights=weights)
    clf.fit(X, y)

    # Plot the decision boundary. For that, we will assign a color to each
    # point in the mesh [x_min, m_max]x[y_min, y_max].
    x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
    y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
    xx, yy = np.meshgrid(np.arange(x_min, x_max, h),
                         np.arange(y_min, y_max, h))
    Z = clf.predict(np.c_[xx.ravel(), yy.ravel()])

    # Put the result into a color plot
    Z = Z.reshape(xx.shape)
    plt.figure()
    plt.pcolormesh(xx, yy, Z, cmap=cmap_light)

    # Plot also the training points
    plt.scatter(X[:, 0], X[:, 1], c=y, cmap=cmap_bold)
    plt.xlim(xx.min(), xx.max())
    plt.ylim(yy.min(), yy.max())
    plt.title("3-Class classification (k = %i, weights = '%s')"
              % (n_neighbors, weights))


plt.show()
```

```
Populating the interactive namespace from numpy and matplotlib
```





3.3 The winning methods

3.3.1 Neural networks

3.3.2 Regularized greedy forest

3.3.3 XGBoost

4 Results on the Kaggle data

4.1 k-nn classification

4.2 Comparision of all methods

5 Discussion

References

[higa] *Dataset from the ATLAS Higgs Boson Machine Learning Challenge 2014*. http://www.http://opendata.cern.ch/record/328, . – Accessed: 2015-12-10

[higb] *Higgs Boson Machine Learning Challenge*. https://www.kaggle.com/c/higgs-boson, . – Accessed: 2016-01-03

List of Figures

List of Tables