# AIS Classifier
# (draft)

Thomas Nordli

Tønsberg: Høgskolen i Vestfold, 2011
Notat 2/2011

Notat nr 2/2011 Høgskolen i Vestfold



ISSN: 0808-131X

# AIS Classifier (draft)

Thomas Nordli <tn@hive.no>

June 29, 2011

## Contents

# 1 Abstract

This report describes a system that makes it possible to explore the AIS data broadcasted by ships. The goal of such a system is to determine how/if it is possible to predict the ship type solely based on AIS messages sent by ships, ignoring the ships self proclaimed type.

The system is intended to be used to discover anomalous behavior by comparing the predicted type with the self proclaimed one. Eventually it is suppose to work online, processing live feeds of data. By now it is only working offline.

The report is to be considered a draft, as the prototype is still missing vital functions, and is not thoroughly tested.

# 2 Introduction

## 2.1 What is AIS?

Automatic Identification System, is a system that obligates some ships to transmit some data over VHF radio. The obligation is a part of the SOLAS convention and applies to all passenger ships, ships with tonnage of 300 or above in international waters and cargo ships trafficing national with tonnage of 500 or above [4].

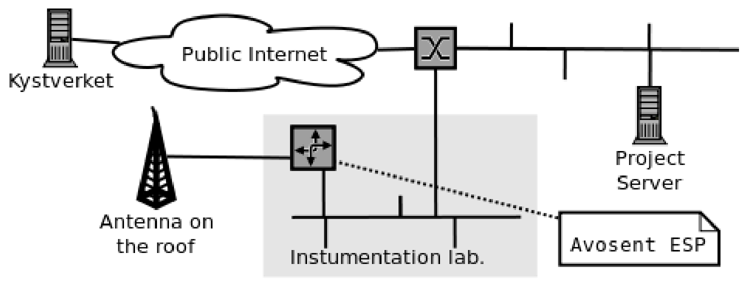
## 2.2 Problem definition

Is it possible to classify the ships based on the fields in the AIS messages (excluding the ship type) alone? The prototype system described here is a contribution to a framework for investigating this question.If the answer to the question turns out to be positive, a software module based on the prototype system developed, can be used for spotting anomalies and raising alarms. Anomaly detection systems have usually a high frequency of false positives, therefore the system must have parameters that may be tuned to lower the false positive rates. Such a software can be standalone or integrated with existing alarm systems.

## 2.3 Assumptions and limitations

This work is done under the assumption that there is detectable differences in the normal behavior of different classes of ships.AIS includes 27 message types. Only message of type 1, and 5 for self proclaimed type, is treated in this report, as these are the only one used in the prototype system.Since this project have not yet identified any effective discriminating features, only four of the fields from message type 1, are included. These fields are rotation (ROT), course over ground (COG), speed over ground (SOG) and heading (HEAD). The mean and variance of these are used as features, giving feature vectors with eight dimensions. These fields and the statistics were chosen because they were easy to implement, as examples to demonstrate the concept. They turned out to have poor ciscriminatroy capabilities.

# 3 Overview of the system.



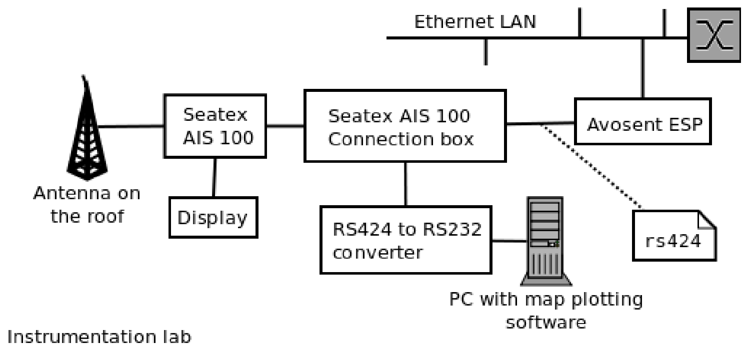

## 3.1 Kystverket

Kystverket, the Norwegian Coastal Administration, is the national agency for coastal management, maritime safety and -communication in Norway. They provide live AIS data for this project via a TCP server.

## 3.2 Project Server

A server running linux is set up to capture, prosess and store the data. The software described below is run on this server.

## 3.3 Instrumentation laboratory




Vestfold University College has an maritime instrumentation lab with some AIS equippement connected to an antenna on the roof. While waiting for Kystverket to process our application for access to live AIS data, a Avosent ESP was purchased, and connected to Seatex AIS 100 via an Connection box. The Avosent ESP was configured to recieve data on an rs424 link and distributing it via TCP on the lab's ethernet LAN. Via the university college ethernet switch, the data was thereby availiable to the project server.

# 4 Expected Quality of the Data

Harati-Mokthari et al. discusses errors found in live data. Many of the ship types are very vage or misleading, such as "vessel" or "cargo". For several types the regulations of use are ambiguous. One of the studies investigated by Harati-Mokthari et al., found vessels that were registered with the same vessel type in Lloyds register database, transmitted different vessel types in the AIS messages[2].

Kjerstad also describes some sources of errors that can be found in AIS data[4].

# 5 Software

## 5.1 netcat

Netcat [3] is called *the swiss army knife of TCP/IP*. It is in this project used to connect to the data soucres, recieve the data and printing it out to *standard output*.

Command: nc nmea.hive.no 4002

## 5.2 devtools_ais

*devtools_ais*, a AIVDM/AIVDO decoder, is part of *gpsd*[6]. It is accompanied with a detailed documentation of how to decode AIVDM/AIVDO sentences [5]. In this project it is used as a command line tool to convert AIVDM data, that comes from the AIS recievers, to a common data format with comma-separated values (CSV).

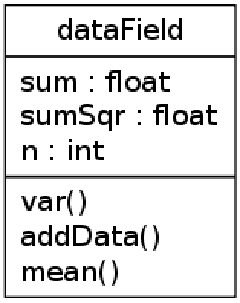
Command: `./devtools_ais.py -c`

## 5.3 TrackSplitter

The CSV file produced by devtools_ais, contains all recieved messages from all ships. The features that we need, should be per ship, therefore a little script is made, that reads the CSV file and splits it into multiple files named in a directory. Each of the files produced contains messages from one ship only. They are named with the ship's MMSI number. The script is called *tracksplitter*, and you can find it listed in appendix A.

## 5.4 TrackDescriber

Each of the files produced by tracksplitter, is to be processed by trackdescriber. Trackdescriber reads data from standard input (STDIN) and makes features that characterize a 'track'. By track is here meant all messages comming from one ship.As there is still not identified any discriminating features, only a few example features is implemented. Four fields are used: ROT, SOG, COG and HEAD. Each field is represented by an instance of a class called 'dataField'.




The class keeps track of three attributes: The sum, the sum of squares and the number of recieved messages (n). Each time a new message arrives, the value is passed to the method addData, and the three attributes are updated. If the script is left running reading real time data (from a live feed, and not from a file), it will eventuall produce an arithmetic overflow. The issue of arithmetic overflow is not yet dealt with.Four objects of this class is instantiated, one from each of the above mentioned AIS fields. STDIN is read line by line, adding data for each line. Each line represents one AIS message. When the features are to be generated (and printed to STDOUT, var() and mean() are called for, producing the mean and variance. This gives a feature vector of eigth dimensions. This printing is preceded with a aproriate header with meta data intended for reading with a classifier written using the python scripting interface from the data mining tool *orange*[1].

TrackDescriber is listed in appendix B

### 5.5 TrackClassifier

The script trackclassifier reads the feature vectors generated by TrackDescriber, and predict a ship type. As there are still no classifiers developed, all the script does at this point in time, is to do some evalution of the discriminating powers of the perliminary features using some of the data mining techniques included in Orange [1]. It currently uses the *AIS ship type* as the class. The reason for this not being a good idea is explained in section 4.

TrackClassifier is listed in appendix C

Below you can the result of a run.

```
$ ./trackClassifier.py
Classes: 26
Attributes: 8

After feature subset selection with margin 0.010 (8 attributes):
Outliers are now removed
Before feature subset selection (8 attributes):
0.069 varCog
0.058 meanHead
0.049 varRot
0.048 meanSog
0.047 varSog
0.040 meanRot
0.039 meanCog
```

```
0.035 varHead
Classification accuracies:
bayes 0.0142857142857
tree 0.201388888889
Classification accuracies (leave one out):
bayes 0.0208333333333
tree 0.201388888889
$
```

## 6 Generation of the dataset

Data is comming in as a stream of text over TCP connections, one line per AIS packet. It is recieceved and logged to a file with the unix command 'nc' (netcat).

```
 $ head -n5 data/hive/nohup.out
!AIVDO,1,1,,,13o02wnP000gk;LQv:uh0?vv0000,0*07
!AIVDM,1,1,,B,D02R3f1HpNfq6DO6D0,4*4F
!AIVDO,1,1,,,13o02wnP000gk;HQv:uP0?w20000,0*7E
!AIVDM,1,1,,A,13mDIj0P000h05FR0@:@0?w000O,0*36
!AIVDM,1,1,,B,13oE6P002Q0h;0VQwfTbS`Q200f,0*71
```

To start caputuring data the following command was used to log data comming in through the antenna on thr roof.`nohup nc nmea.hive.no 4002 &`

The command nohup makes sure that the process created by the rest of the commandline is disconnected from the current session, so that it will continue after the session is terminated and the user that issued the command has logged out. The logging will now continue until an authorized user stops it, the systems shuts down or the storage space is filled up.The tool 'devtools_ais' extracts the packed bitfields from the aquired AIS messages and converted to a 'csv' (comma separated values) format.

```
$ head -n5 data/hive/nohup.out | ./devtools_ais.py -c
1,0,258999039,6,-128,0,0,6265198,35621623,0,511,31,0,0,0
20,0,2655160,1422,1,7,750,1125,1,7,1125,0
1,0,258999039,6,-128,0,0,6265196,35621622,0,511,33,0,0,0
1,0,257235400,0,-128,0,0,6291627,35655721,0,511,32,0,0,49183
1,0,259344000,0,0,161,0,6314003,35647122,2702,272,33,0,0,49198
```

Trackplitter reads the data produced by 'devtools_ais.py -c'. The data is split by mmsi, and written to temporary files in './tmpfiles'. Each file represents data from one ship, and is named with its mmsi number.

```
$  ls tmpfiles/ | head -n5
156198689
205203000
205585000
209324000
209350000
```

Each file in the directory contains information on one ship.

```
$  head -n5 tmpfiles/205203000
1,0,205203000,0,0,192,1,6336077,35873338,1810,180,54,0,0,81956
3,0,205203000,0,-15,193,1,6329874,35849725,1820,181,26,0,0,0
1,0,205203000,0,11,184,1,6369048,35786658,1740,174,54,0,0,81945
1,0,205203000,0,0,190,1,6370747,35779543,1730,173,12,0,0,21076
1,0,205203000,0,0,189,1,6371729,35775168,1730,173,36,0,0,81948
```

The track_describer Reads AIS messages from STDIN. It assumes that the messages is formatted as produced by './devtools.py -c', and that all the messages originates in the same ship. Features are calculated and output is printed on STDOUT.To be scalable, and to be able to continously processing data from live streams, It is desiged in a way that it does not have to store all the data. Only aggregations of data is stored (and updated for each incomming packet).The following command is used to write features of all the tracks to the file tabulator separated file called 'data.tab'.

```
for F in tmpfiles/* ;do ./track_describer.py  $F;done > data.tab
```

The file 'data.tab' has to be provided with a header, giving names and types to the columns. The column indicating the class (ship type) is also marked. This is done giving the file å header.

```
$  head -n5 data.tab
meanSog stdSog  meanHead stdHead meanCog stdCog  meanRot stdRot type
c       c       c        c       c       c       c       c      d
                                                                class
0.188781  29.984842 188.666667 27.537797   1802.664509 17832.525553  \
180.325782 173.444507  35
-0.452115 48.898434 41.386802  1782.776197 2222.091371 860973.688331 \
112.157699 6503.311616 70
```

The first line contains the field names, The second names declears type (c = continous, d = discrete). The third line marks the column that indicates the column that the classifier should aim to predict.The file 'data.tab' can now be used by the tool trackClassifier.

# 7 Result

The project has developed a system for producing data set, preprocessing the data, extracting features and analyzing them. The software works only offline, but is developed with the goal of beeing able to work both offline and online.Two datasets are produced. One based on local messages recived from the antenna on the roof. And one based on data comming from Kystverket.

# 8 Future work

This project has produced a framework for further investigation. In this section you will find some a brief description of some of the work that should be done, if the project is continued.

### 8.1 Features

As there is not yet identified any features, this is something to do further work on. When features are identified more preprossessing on some of the fields have to be done. The details of the domains of fields can be found in *AIVDM/AIVDO protocol decoding*[5]

### 8.2 Classes

Search for classes – the AIS field ship type, is not usable as a class field in a classifing algorithm. This is due to the unambigous use and regultations of the field, as explained in section 4.

### 8.3 Data

Add a message timestamp to each recieved message.Consider other types of data sources, in addition to AIS. This may be data from radar or from external databases.

### 8.4 Implementation

Put the data in relational database. This will make it easier to query the data. The issue of arithmetic overflow in the trackdesciber.dataField.addData() must be dealt with if the system is to be used in online processing.

## 9 Conclusion

A lot of work have been done to prepeare the ground for investigating the possibilities of data mining the AIS messages. To answer the question whether it is possible to classify the ships based on AIS data, while distrusting the ship type field, the project have to be continued.

## A Tracksplitter

The script is written as a part of this project to demultiplex the messages, based on their origin.

Listing 1: tracksplitter.py

```
#!/usr/bin/python
""" Reads data from standard input.
Data should be produced by 'devtools_ais.py -c'.
The data is split by mmsi, writing files
in the './tmp' directory
or the directory given with option '-d'.

Each file represents data from one ship,
and is named with the mmsi number """


import os,sys,getopt

if __name__ == "__main__":
```

```

    datafiledir='tmp'
    o, a = getopt.getopt(sys.argv[1:], 'd:')
    opts = {}
    for k,v in o:
        if k == '-d':
            datafiledir=v


    ships = {}

    # Makes sure the temporary directory is empty
    if os.path.isdir(datafiledir):
        for f in os.listdir(datafiledir):
            os.unlink("%s/%s"%(datafiledir,f))
    else:
        os.mkdir(datafiledir)

    while 1:

        line = sys.stdin.readline() # Read from stdin
        if not line:                # EOF
            break

        mmsi=line.split(',')[2] # finds mmsi

        if not ships.has_key(mmsi):

            # stores file objects in dictionary
            ships[mmsi] = open("%s/%s" % (datafiledir, mmsi), 'a')

        ships[mmsi].write(line)
```

# B Track Describer

The script is written as a part of this project to produce the (preliminary) features describing a ship based on its AIS brodcasts.

Listing 2: trackdescriber.py

```
#!/usr/bin/python

"""Reads AIS messages from STDIN.
Input format: CSV (as from './devtools.py -c | ./tracksplitter.py')

Output: Features printed on STDOUT

"""

import sys, os

class dataField:
    """ Class representing one of the fields of an AIS message. """

    def __init__(self):
        self.sumSqr    = 0.0
        self.sum       = 0.0
        self.n         = 0

    def addData(self,newData):
        self.sum += newData
```

```
        self.sumSqr += newData**2
        self.n +=1

    def mean(self):
        return self.sum / self.n

    def var(self):
        return ( self.sumSqr - self.sum*(self.sum/n) ) / (self.n -1 )

    def __str__(self):
        return "%f\t%f" % ( self.mean(), self.var() )

def readData():

    mmsi = '' # Maritime Mobile Service Identity
    # navgitation status
    rot  = 0  # Rate of turna
    sog  = 0  # Speed over ground
    # position accuracy
    # longitude
    # latitude
    cog  = 0  # Course over ground
    head = 0  # True heading
    # time stamp
    # manouver indicator
    # RAIM flag
    # Radio status

    sType = int('100') # Ship type (100 is out of range)
    n    = 0  # Number of samples

    fieldnames = ['rot', 'sog', 'cog', 'head']
    flds       = {}
    for f in fieldnames:
        flds[f] = dataField();


    while 1:

        line = sys.stdin.readline()  # Read from stdin.
        if not line:                 # EOF
            break

        fields = line.split(',')

        messageType = int(fields[0])

        if messageType == 5:          # !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
            sType = int( fields[7] )
            continue

        if messageType != 1:
            continue

        n += 1
        flds['rot'].addData(  int( fields[4]  ) )
        flds['sog'].addData(  int( fields[5]  ) )
        flds['cog'].addData(  int( fields[9]  ) )
        flds['head'].addData( int( fields[10] ) )

```

```
        mmsi  = fields[2]
        return (flds, sType, n, mmsi)

if __name__ == "__main__":

        (flds, sType, n, mmsi) = readData()

        if (n>1):
            print "%s\t%s\t%s\t%s\t%d" % (flds['rot'], flds['sog'], flds['cog'], flds['head'], sType)
            #print "%s (%d):\t%s\t%s\t%s\t%s\t%d" % (mmsi,n,flds['rot'], flds['sog'], flds['cog'], flds['head'], sType)
```

# C TrackClassifier

The script is written as a part of this project to start the development of a script that constitutes the module responisble for classifying the ships based on the feature vectors comming from TrackDescriber. As of now, it only do some test on the preliminary features' discriminatory capabilities. It also uses the *AIS ship type* as class. As explained in section 4.

Listing 3: trackClassifier.py

```
#!/usr/bin/python
import orngOutlier
import orngTree
import orange, orngFSS
import orngDisc, orngTest, orngStat
import orngClustering
import os

def accuracy(test_data, classifiers):
    correct = [0.0]*len(classifiers)
    for ex in test_data:
        for i in range(len(classifiers)):
            if classifiers[i](ex) == ex.getclass():
                correct[i] += 1
    for i in range(len(correct)):
        correct[i] = correct[i] / len(test_data)
    return correct


def cross_validation(data, learners, k=10):
    acc = [0.0]*len(learners)
    selection = orange.MakeRandomIndicesCV(data, folds=k)
    for test_fold in range(k):
        train_data = data.select(selection, test_fold, negate=1)
        test_data = data.select(selection, test_fold)
        classifiers = []
        for l in learners:
            classifiers.append(l(train_data))
        acc1 = accuracy(test_data, classifiers)
        for j in range(len(learners)):
            acc[j] += acc1[j]
    for j in range(len(learners)):
        acc[j] = acc[j]/k
    return acc


def leave_one_out(data, learners):
```

```python

    acc = [0.0]*len(learners)
    selection = [1] * len(data)
    last = 0
    for i in range(len(data)):
        selection[last] = 1
        selection[i] = 0
        train_data = data.select(selection, 1)
        for j in range(len(learners)):
            classifier = learners[j](train_data)
            if classifier(data[i]) == data[i].getclass():
                acc[j] += 1
        last = i

    for j in range(len(learners)):
        acc[j] = acc[j]/len(data)
    return acc


def report_relevance(data):
  m = orngFSS.attMeasure(data)
  for i in m:
    print "%5.3f %s" % (i[1], i[0])


def setMargins():
    global data
    marg = 0.01
    filter = orngFSS.FilterRelief(margin=marg)
    ndata = filter(data)
    data = ndata
    print "\nAfter feature subset selection with margin %5.3f (%d attributes):" % \
        (marg, len(data.domain.attributes))


def removeOutliers():

    global data
    outlierDet = orngOutlier.OutlierDetection()
    ndata=orange.ExampleTable(data.domain)

    outlierDet.setExamples(data)
    z=outlierDet.zValues()
    for i in range (len(data)):
        if abs(z[1])<1.5:
            ndata.append(data[i])

    print "Outliers are now removed"
    data=ndata


def kmeans():
    global data
    for k in range(2,18):

        km = orngClustering.KMeans(data, k, initialization=orngClustering.kmeans_init_diversity)
        score = orngClustering.score_silhouette(km)
        print km.clusters, k, score


```

```
if __name__ == "__main__":

    data = orange.ExampleTable("data.tab")

    # report on number of classes and attributes

    print "Classes:", len(data.domain.classVar.values)
    print "Attributes:", len(data.domain.attributes)

    setMargins()
    removeOutliers()

    print "Before feature subset selection (%d attributes):" % len(data.domain.attributes)

    report_relevance(data)

    # commented out -- reporting result is not implemented, giving verbose output
    # kmeans()


    bayes = orange.BayesLearner()
    tree = orngTree.TreeLearner(mForPruning=2)

    bayes.name = 'bayes'
    tree.name = 'tree'
    learners = [bayes,tree]


    acc = cross_validation(data, learners, k=10)

    print "Classification accuracies:"
    for i in range(len(learners)):
        print learners[i].name, acc[i]


        acc = leave_one_out(data, learners)

    print "Classification accuracies (leave_one_out):"
    for i in range(len(learners)):
        print learners[i].name, acc[i]
```

# References


[1] Orange. http://orange.biolab.si/.

[2] A. Harati-Mokhtari, A. Wall, P. Brooks, and J. Wang. Automatic Identification System (AIS): A Human Factors Approach. In *The Nautical Institute AIS Forum*, pages 1–11. Citeseer, 2007.

[3] hobbit@avian.org hobbit@avian.org. netcat.

[4] Norvald Kjerstad. *Elektroniske og akustiske navigasjonssystemer: for maritime studier*. Number 3. utg. Tapir akademisk forl., Trondheim, 2008.

[5] Eric S. Raymond. Aivdm/aivdo protocol decoding v1.25, June 2010. http://gpsd.berlios.de/AIVDM.html.

[6] Eric S. Raymond et al. gpsd — a gps service daemon.