Str Gheorghe Titeica nr 6, Sector 2,
Bucuresti
Romania

+4 031 4055287
+4 031 4012234
+4 0721 368 127

Website:
http://www.4esoft.com

email: office@4esoft.com

Client experimental GoDriveCarBox

Arhitectura modelelor de predictie a defectarii componetelor autovehiculului

Proiect	GoDrive
Beneficiar	GODRIVE SRL
Contract	Nr. 2/25.11.2016
Data modificare	2017.03.08
Data creere	2017.02.02
Versiune	1.0.0.2
Descriere	Arhitectura modele predictie defecte autovehicule

Problema predictiei defectarii componentelor autovehicolului comporta multiple aspecte ce trebuie luate in consideratie la implementare atat din punct de vedere tehnic cat si din punct de vedere al modelarii de date/informatii. In acest sens s-au identificat urmatoarele aspecte:

Tabelul analizei solutiilor arhitecturale

Nr	Problema	Natura	Descriere	Solutie
1	Modelarea predictie	Modelare	In vederea determinarii unei predictii a unei potentiale defectari de componenta este	Se vor folosi modele de predictie avansate bazate pe tehnologiile actuale ale invatarii automatizate (stadiul curent al tehnologiei, cercetarii si dezvoltarii). S-a optat

Str Gheorghe Titeica nr 6, Sector 2,
Bucuresti
Romania

+4 031 4055287
+4 031 4012234
+4 0721 368 127

Website:
http://www.4esoft.com

email: office@4esoft.com

Nr	Problema	Natura	Descriere	Solutie
			necasara analiza factorilor si a modului de inferenta a corelatiei optime. Chiar daca statistic se poate determina o inferenta este necesara gasirea unui algoritm auto-adaptabil care sa "invete" sa gaseasca corelatii si sa determine predictii	pentru utilizarea unui model de invatarea automatizata bazat pe retele neurale cu multi-nivele ascunse. In "***Tabelul de descriere a solutiei de modelare***" sunt prezentate detaliile propuse ale arhitecturii matematice solutiei.
2	Metode de determinare a predictorilor	Modelare	Indentificarea fluxurilor de date ce pot genera potentiale inferente privind functionarea autovehicolului: avand un model de baza matematic este necesara determinarea variabilelor principale si a ponderilor	S-au analizat numeroase lucrari stiintifice publicate in ultimii 5 ani in domeniul tehnologiilor bazate pe Inteligenta Artificiala. Din aceste lucrari amintim doua dintre ele axate in special pe utilizarea sistemelor cu invatare automatizata in modele de predictibilitate a intretinerii flotelor de autovehicule si auto-utilitare:

Str Gheorghe Titeica nr 6, Sector 2,
Bucuresti
Romania

+4 031 4055287
+4 031 4012234
+4 0721 368 127

Website:
http://www.4esoft.com

email: office@4esoft.com

Nr	Problema	Natura	Descriere	Solutie
			acestora in determinarea functiei ipoteza de invatare automata si predictie	A) R. Prytz, S. Nowaczyk, T. Rögnvaldsson, S. Byttner, "Analysis of Truck Compressor Failures Based on Logged Vehicle Data", in In Proceedings of the 9th International Conference on Data Mining (DMIN'13), Las Vegas, NV, USA. July 2013 B) T. Rögnvaldsson, S. Byttner, R. Prytz, S Nowaczyk, "Wisdom of Crowds for Self-organized Intelligent Monitoring of Vehicle Fleets", submitted to IEEE Transactions on Knowledge and Data Engineering (TKDE), 2014
3	Rularea modelului atat in mediu computational intensiv cat si in mediu local	Tehnica	Modelul predictiv ales bazat pe tehnologiile state-of-the-art de deep leaning are inconvenientul necesitatii unei puteri	Solutia aleasa consta in spargerea modelarii in doua etape: A) Etapa de preantrenare a modelului cu invatare automatizata la nivelul

Str Gheorghe Titeica nr 6, Sector 2,
Bucuresti
Romania

+4 031 4055287
+4 031 4012234
+4 0721 368 127

Website:
http://www.4esoft.com

email: office@4esoft.com

Nr	Problema	Natura	Descriere	Solutie
			mari de calcul precum si a	serverului GoDrive. In cadrul acestei etape modelele bazate pe relete neurale adanci vor fi antrenate prin invatare automatizata si se va genera un model initial al ponderilor predictorilor din retele. Se vor utiliza date off-line. B) Etapa de auto-invatare continua la nivelul dispozitivului incorporat GoDrive va asigura ajustarea ponderilor predictorilor si a nivelelor ascunse adanci din retelele neurale in functie de dinamica datelor colectate si analizate din mediul real.
4	Disponibilitatea datelor la faza de experimentare	Modelare / Tehnica	In vederea realizarii procesului de invatare automatizata este necesara si obligatorie existenta unor seturi	Se vor achizitiona seturi de date din surse externe: A) UCI Machine Learning Repository collection of

Str Gheorghe Titeica nr 6, Sector 2,
Bucuresti
Romania

+4 031 4055287
+4 031 4012234
+4 0721 368 127

Website:
http://www.4esoft.com

email: office@4esoft.com

Nr	Problema	Natura	Descriere	Solutie
			pe baza carora sa se faca experimentarea modelarii intiale inferentiale	databases, domain theories, and data generators, Center for Machine Learning and Intelligent Systems Bren School of Information and Computer Science, University of California, Irvine B) Vehicle safety defect investigations, recalls and collision investigations data for Great Britain, Driver and Vehicle Standards Agency, UK

Tabelul de descriere a solutiei de modelare

Algoritm	Model auto-modelare de predictor si predictie a potentialelor defecte in autovehicul
Versiune	1.0.2
Data creatiei	2017.02.01
Data ultimei modificari	2017.04.09

Str Gheorghe Titeica nr 6, Sector 2,
Bucuresti
Romania

+4 031 4055287
+4 031 4012234
+4 0721 368 127

Website:
http://www.4esoft.com

email: office@4esoft.com

Tip algoritm	Model cu invatare automatizata
Model de optimizare	Optimizare cu invatare automatizata bazata pe calculul gradientilor erorii functiei de cost
Descriere generala	Determinarea atat a factorilor care determina defectarea componentelor autovehicolului cat si a momentului in care o anumita componenta se defecteaza poate fi vazuta ca o functie cu complexitate ridicata H(X)=Y multi-variata (considerand X ca fiind un vector de date stohastice generate de catre autovehicol) si multi-nomiala de date (consideram Y ca fiind un vector multi-nomial in care fiecare element reprezinta un rezultat de predictie sau inferenta). Datorita caracterului atat euristic cat si stohastic al procesului de analiza al variabilelor si de determinare a rezultatelor este putin probabila aplicarea unui algoritm complex prin care sa se defineasca functia H(X)=Y, motiv pentru care functia ipoteza H(X) va fi determinata de un model avansat cu invatare automatizata bazat pe retele neurale cu nivele multiple ascunse precum si convolutiile necesare procesului de determinare a variabilelor calculate.
Date de intrare	Vectorul X de parametrii va contine toate datele obtenabile prin intermediul interfetei On Board Diagnostics
Date de iesire	In urma analizei literaturii si lucrarilor de specialitate precum si in urma achizitionarii seturilor de date propuse se vor identifica variabilele “tinta” urmarite care sunt in directa corelatie (aproximativ 1-la-1) cu predictia defectarii unuia sau mai multor componente
Metoda de calcul	Plecand de la premiza ca intreg modelul reprezinta o functie ipoteza de determinare in baza unui vector multi-variat a unui

Str Gheorghe Titeica nr 6, Sector 2,
Bucuresti
Romania

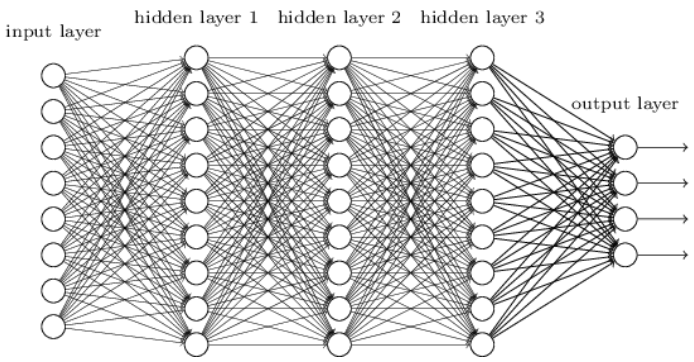
+4 031 4055287
+4 031 4012234
+4 0721 368 127

Website:
http://www.4esoft.com

email: office@4esoft.com

rezultat multi-nomial se construieste o retea neurala pe structura intuitiva a neuronilor biologici in care avem:

- Fiecare neuron primeste informatii de la mai multi neuroni conectati de la axoni acestora la dendritele sale
- Neuronul individual realizeaza o peratie liniara dupa care trece rezultatul operatiei liniare printr-o functie non-liniara de activare (de exemplu sigmoid $\sigma(z) = (\frac{1}{1+e^{-z}})$)
- Prin introducerea unui numar mare de neuroni pe nivele multiple se poate “inmagazina” o cantitate mare de informatie in vederea obtinerii unei functii cu complexitate foarte ridicata non-liniara. Un exemplu este dat de imaginea de mai jos:



- In vederea calculului functiei ipoteza H(X)=Y se aplica un algoritm de “mers inainte” de forma:

$$A_i = \sigma(A_{i-1} * W_i)$$

Unde W(i) este un vector de ponderi al dendritelor neuronului respectiv ce se aplica intrarilor din neuronii nivelului neural anterior

Str Gheorghe Titeica nr 6, Sector 2,
Bucuresti
Romania

+4 031 4055287
+4 031 4012234
+4 0721 368 127

Website:
http://www.4esoft.com

email: office@4esoft.com

	- In final se calculeaza eroarea intre valoarea multi-nomiala asteptata si cea rezultata din functie ipoteza cu: $$E = \sqrt{\frac{(Out_{corect} - A_n)^2}{2}}$$ - Invatarea efectiva consta in ajustarea ponderilor (vectorii/matricile W(i,j) unde i,j reprezinta nivelele retelei neurala) automatizata se realizeaza prin metoda propagarii inapoi a gradientului erorii determinate si prezentate anterior. Concret pentru cazul particular al determinarii ponderilor dendritelor pentru nivelul 3 intr-o retea neurala cu 5 nivele (nivelul 5 fiind nivelul de calcul final al vectorului multi-nomial de iesire): $$\frac{\partial E}{\partial \theta_3} = \frac{\partial E}{\partial a_5} * \frac{\partial a_5}{\partial z_5} * \frac{\partial z_5}{\partial a_4} * \frac{\partial a_4}{\partial z_4} * \frac{\partial z_4}{\partial a_3} * \frac{\partial a_3}{\partial z_3} * \frac{\partial z_3}{\partial \theta_3}$$ In cazul de mai sus vectorul/variabile Θ este echivalenta cu W. Ajustarea efectiva a ponderilor din matricile respective se face prin aplicarea unui algoritm de tip gradient-descent - ajustarea periodica cu fragmente ale gradientului pana la atingerea unui minim al functiei de cost E.
Metoda de implementare	Implementarea se va realiza prin utilizarea unui limbaj de tip multi-platforma independent de dispozitivul hardware – in cazul de fata limbajul Python.
Mediul de rulare la nivelul dispozitivului incorporat	La nivelul dispozitivului incorporat se va utiliza un micro-interpretor Python capabil sa ruleze pe dispozitive cu capacitati extrem de reduse (Embedded Python)
Variante de modele propuse	Varianta A: in primul stadiu al experimentarii se propune un model cu invatare superficiala specializat pe functionarea in medii on-line

Str Gheorghe Titeica nr 6, Sector 2,
Bucuresti
Romania

+4 031 4055287
+4 031 4012234
+4 0721 368 127

Website:
http://www.4esoft.com

email: office@4esoft.com

	Varianta B: in stadiul avansat al proiectului se va face trecerea de la modelul superficial la un model cu parametrizare adanca de tip retea convolutionala adanca

Anexa 1 - Cod sursa modul predictiv bazat pe arhitectura superficiala

```
# -*- coding: utf-8 -*-
"""
@application: Online Classifier Engine
@created: 2017-01-04
@author: 4E SOFTWARE SRL

=======

TODO:
    Momentum/Velocity for Online learning setting
    NEO LineSearch (adapted for online)
    VotingClassification


"""

import pandas as pd
import numpy as np

from scipy.special import expit
from scipy import stats

import matplotlib.pyplot as plt #  for debug error plotting
from time import time
import sys

class oce_utils:

    def FeatureNormalize(self,X_data, method = 'z-score'):
        if method == 'z-score':
            min_val = X_data.mean(axis=0)
            div_val = X_data.std(axis=0)
        elif method =="minmax":
            ## min-max
```

Str Gheorghe Titeica nr 6, Sector 2,
Bucuresti
Romania

+4 031 4055287
+4 031 4012234
+4 0721 368 127

Website:
http://www.4esoft.com

email: office@4esoft.com

```
            min_val = X_data.min(axis=0)
            div_val = X_data.max(axis=0)
        else:
            raise Exception("Unknown scale/norm method: "+str(method))

        div_val[div_val == 0] = 1.

        X_norm = X_data - min_val
        X_norm = np.array(X_norm,dtype = float) / div_val

        return X_norm, min_val, div_val

    def TestDataNormalize(self, X_test, min_val,div_val):
        X_norm = X_test - min_val
        X_norm = np.array(X_norm,dtype = float) / div_val

        return X_norm

    def loaddata(self, file):
        return pd.read_csv(file)

    #
    # Kappa: duplicated and generalized from OnlineClassifier version
    #
    def Kappa(self,y_pred,y_truth, classes):
        nr_classes = len(classes)
        classes = list(classes)
        TP = np.zeros(shape=(nr_classes))
        FP = np.zeros(shape=(nr_classes))
        TN = np.zeros(shape=(nr_classes))
        FN = np.zeros(shape=(nr_classes))
        class_pred = np.zeros(shape=(nr_classes))
        class_real = np.zeros(shape=(nr_classes))

        for (i,c_class) in zip(range(nr_classes),classes):
            TP[i] = np.logical_and( y_pred == c_class, y_truth == c_class ).sum()
            TN[i] = np.logical_and( y_pred != c_class, y_truth != c_class ).sum()
            FP[i] = np.logical_and( y_pred == c_class, y_truth != c_class ).sum()
            FN[i] = np.logical_and( y_pred != c_class, y_truth == c_class ).sum()
            class_pred[i] = TP[i] + FP[i]
            class_real[i] = TP[i] + FN[i]

        all_ex = TP[0]+TN[0]+FP[0]+FN[0]
        observed_accuracy = np.sum(TP) / all_ex
```

Str Gheorghe Titeica nr 6, Sector 2,
Bucuresti
Romania

+4 031 4055287
+4 031 4012234
+4 0721 368 127

Website:
http://www.4esoft.com

email: office@4esoft.com

```
        expected_accuracy = (np.sum(class_pred*class_real) / all_ex) / all_ex
        kappa = (observed_accuracy - expected_accuracy) / \
                (1 - expected_accuracy)
        # conf_matrix !!!
        return kappa

    #
    # ROC: duplicated and generalized from OnlineClassifier version
    #
    def ROC(self,y_prc,y_label, labels):
        nr_labels = len(labels)
        if y_label.ndim>1:
            y_label_list=y_label[:,0]
        thresholds = np.linspace(1, 0, 101)
        if nr_labels == 2:
            nr_ROCs = 1
        else:
            nr_ROCs = nr_labels

        TPR = np.zeros(shape=(101,nr_ROCs))
        FPR = np.zeros(shape=(101,nr_ROCs))
        AUC = np.zeros(shape=(nr_ROCs))

        for cROC in range(nr_ROCs):
            if nr_ROCs==1:
                c_label=1
            else:
                c_label = labels[cROC]
            for i in range(101):
                c_thr = thresholds[i]
                # Classifier / label agree and disagreements for current threshold.
                if i==50:
                    k=1
                TP = np.logical_and( y_prc[:,cROC] > c_thr, y_label_list==c_label ).sum()
                TN = np.logical_and( y_prc[:,cROC] <=c_thr, y_label_list!=c_label ).sum()
                FP = np.logical_and( y_prc[:,cROC] > c_thr, y_label_list!=c_label ).sum()
                FN = np.logical_and( y_prc[:,cROC] <=c_thr, y_label_list==c_label ).sum()

                # Compute false positive rate for current threshold.
                FPR[i,cROC] = FP / float(FP + TN)

                # Compute true  positive rate for current threshold.
                TPR[i,cROC] = TP / float(TP + FN)
```

Str Gheorghe Titeica nr 6, Sector 2,
Bucuresti
Romania

+4 031 4055287
+4 031 4012234
+4 0721 368 127

Website:
http://www.4esoft.com

email: office@4esoft.com

```
            # compute the AUC score for the ROC curve using the trapezoidal method
            AUC[cROC] = 0.
            for i in range(100):
                AUC[cROC] += (FPR[i+1,cROC]-FPR[i,cROC]) * (TPR[i+1,cROC]+TPR[i,cROC])

            AUC[cROC] *= 0.5


        return TPR,FPR, AUC


    ##
    ## train_online_classifier() simulates a real life
    ## feed of data to our OnlineClassifier
    ## cross-validation is used to obtain best J(Theta)
    ##
    def train_online_classifier(self, clf, X_train,y_train, X_cross = None,y_cross =
None,batch_size=1):
        nr_examples = X_train.shape[0]
        nr_batches = nr_examples / batch_size
        for i in range(nr_batches):
            xi = X_train[(i*batch_size):((i+1)*batch_size),:]
            yi = y_train[(i*batch_size):((i+1)*batch_size)]
            clf.OnlineTrain(xi,yi,X_cross=X_cross,y_cross=y_cross)
        return clf


    ##
    ## ck_train_online_classifier() simulates a real life
    ## feed of data to our OnlineClassifier
    ##
    ##
    def ck_train_online_classifier(self, clf, X_train,y_train, X_cross = None,y_cross =
None,batch_size=1):
        nr_examples = X_train.shape[0]
        nr_batches = nr_examples / batch_size
        for i in range(nr_batches):
            xi = X_train[(i*batch_size):((i+1)*batch_size),:]
            yi = y_train[(i*batch_size):((i+1)*batch_size)]
            clf.OnlineTrain(xi,yi,X_cross=X_cross,y_cross=y_cross)
        return clf

##
## implement OnlineClassifier
```

Str Gheorghe Titeica nr 6, Sector 2,
Bucuresti
Romania

+4 031 4055287
+4 031 4012234
+4 0721 368 127

Website:
http://www.4esoft.com

email: office@4esoft.com

```
## both multi-class (one-vs-all) and single-class logistic regression
## y is either multi-class or True/False
##
##
class OnlineClassifier:
    def __init__(self,nr_features,classes=[0,1],
                 alpha=1.0, DecreasingAlpha=False,
                 alpha_coef=-1, softmax_alpha_search = False,
                 polyfeats=1,method='sigmoid',
                 lmbd=0,
                 random_init=False,
                 Verbose = 5, NoVerbose = False,
                 back_train = 1):

        self.back_train = back_train
        self.softmax_alpha_search = softmax_alpha_search
        self.alpha_search_epochs = 100
        self.alpha_search_iter = 0;
        self.Classes = list(classes)  # class labels binary default
        self.eps = 1e-15  #  constant used for clipping
        self.Verbose = Verbose #  this is the verbose level: the higher the most-import-only
info is displayed
        self.NoVerbose = NoVerbose #  force to ignore Verbose property
        self._standard_binary_classes = [0,1]
        self.lmbd = lmbd  # lambda for reguralization DEFAULT 0 (no reg)
        self.methods = ['softmax','sigmoid','perceptron']  #"methodation" function
        if not (method in self.methods):
            raise Exception("Unknown method: "+method)
        self.method =  method  # can be default='sigmoid' or 'softmax'
        self.DecreasingAlpha = DecreasingAlpha  # alpha gradient step decreases ?
        self.alpha_coef = alpha_coef    # coef for alpha decrease = actually not used
        self.base_alpha = alpha     # alpha
        self.nr_Classes = len(self.Classes) #    number of classes (2 default)
        self.original_n = nr_features +1    # original number of features MUST be precoded
includes intercept
        self.n = nr_features*polyfeats + 1  # number of features +1 (poly features + 1 used
only if poly = true)
        self.m = 0
        self.alpha = alpha
        self.alpha_0 = alpha
        self.alpha_array = np.empty((0,1),float)
        self.MultiClass = False
        if method == 'sigmoid':
            self.Costs = np.empty((0,self.nr_Classes), float)
```

Str Gheorghe Titeica nr 6, Sector 2,
Bucuresti
Romania

+4 031 4055287
+4 031 4012234
+4 0721 368 127

Website:
http://www.4esoft.com

email: office@4esoft.com

```
            self.J_array = np.empty((0,self.nr_Classes), float)
        else:
            self.Costs = np.empty((1,0), float)
            self.J_array = np.empty((1,0), float)
        self.xi = np.array([])
        self.y = None
        self.BestAccuracy = 0
        self.BestFeed = 0
        self.polyfeats = polyfeats # >1 if using polynomial feats remapping
        self.all_X = np.empty((0,self.n), float) # all xi in one matrix
        self.all_y = np.empty((0), dtype = object)
        nr_thetas=1
        if (self.nr_Classes>2) or (self.method == 'softmax'):
            nr_thetas=self.nr_Classes
            self.MultiClass = True

        self.BestTheta = None
        self.random_init = random_init
        if random_init:
            ###
            ### random Theta initilization
            ### wih "noise" values (-0.05 to +0.05)
            ###
            self.Theta = np.random.uniform(low=-0.05, high=0.05, size=(self.n,nr_thetas))
        else:
            self.Theta = np.zeros(shape=(self.n,nr_thetas))

        ## now we need a mechanism to preserve all gradients for each class
        ## we will use a 3d matrix (iteration,class,actual_theta)
        ## this way we can analyse exploding gradients
        self.gradients = np.empty((0,nr_thetas,self.n),float)
        self._nr_thetas = nr_thetas

        self.LastGrad = None
        self.LastYOHM = None
        self.LastYHat = None
        self.LastYERR = None
        self.LastGThe = None
        self.LastThet = None
        self.LastJ = None
        self.LastAlph = None

        self.BestAlphas = list()
```

Str Gheorghe Titeica nr 6, Sector 2,
Bucuresti
Romania

+4 031 4055287
+4 031 4012234
+4 0721 368 127

Website:
http://www.4esoft.com

email: office@4esoft.com

```
    def SearchBestAlpha(self,x, ohmy,Verbose = True):
        bestAlpha = 0
        bestDiff = -1e100
        if self.LastJ == None:
            return bestAlpha

        alphas = np.array([1e-5,5e-5,1e-4,5e-4,1e-3,5e-3,1e-2,5e-2,0.1,0.5,1,5])
        diff_list = list()
        for i in range(alphas.size):
            test_alpha = alphas[i]
            ##
            ## now compute test weights based on previous weights
            ## updated with previous gradient and tested alpha
            ## then compute current J(theta) and determine
            ## best previous update step (best previous alpha)
            TestTheta = self.LastGThe - test_alpha*self.LastGrad
            m = np.float64(x.shape[0])        # batch update size not all obs !!!
            Theta = np.array(TestTheta)
            xT = x.dot(Theta)
            yhat = self.softmax(xT)
            yhat = np.clip(yhat,self.eps,1-self.eps)
            # now final calc incl reguralization
            J = self._log_loss_reg(ohmy,yhat, self.lmbd, Theta, m)

            J_diff = self.LastJ - J
            diff_list.append(J_diff)

            if J_diff > bestDiff:
                bestAlpha = test_alpha
                bestDiff = J_diff

        if Verbose:
            self.DebugInfo("[DEBUG]     BestAlpha = {:.5f}".format(bestAlpha), 10)
        self.alpha_search_iter = self.alpha_search_iter  + 1

        return bestAlpha




    def DebugInfo(self, Value, lvl=0):
```

Str Gheorghe Titeica nr 6, Sector 2,
Bucuresti
Romania

+4 031 4055287
+4 031 4012234
+4 0721 368 127

Website:
http://www.4esoft.com

email: office@4esoft.com

```
        if self.NoVerbose:
            return
        if lvl<=self.Verbose:
            return
        text = ""
        #text = str(type(Value))
        #text += ':\n'
        text += str(Value)
        if self.Verbose:
            print text
            sys.stdout.flush()

    def GetShortHyperParams(self):
        return "Method={} Poly={} BatchSize={} Alpha0={}".format(self.method,
                                                                 self.polyfeats,
                                                                 self.batchsize,
                                                                 self.alpha_0)

    def GetHyperParams(self):
        str_params  = "\nHyper Parameters:"
        str_params += "\nHyFunction: "+str(self.method)
        str_params += "\nAlpha-init: "+str(self.alpha_0)
        str_params += "\nDecrAlpha : "+str(self.DecreasingAlpha)
        str_params += "\nAlphaCoef : "+str(self.alpha_coef)
        str_params += "\nSM-alpsrch: "+str(self.softmax_alpha_search)
        str_params += "\nAlpha-last: "+str(self.alpha_array[-3:])
        str_params += "\nPolynomial: "+str(self.polyfeats)
        str_params += "\nClasses   : "+str(self.Classes)
        str_params += "\nRegLambda : "+str(self.lmbd)
        str_params += "\nRandTheta : "+str(self.random_init)
        str_params += "\n"
        return str_params

    def Kappa(self,y_pred,y_truth, classes):
        nr_classes = len(classes)
        classes = list(classes)
        TP = np.zeros(shape=(nr_classes))
        FP = np.zeros(shape=(nr_classes))
        TN = np.zeros(shape=(nr_classes))
        FN = np.zeros(shape=(nr_classes))
        class_pred = np.zeros(shape=(nr_classes))
        class_real = np.zeros(shape=(nr_classes))

        for (i,c_class) in zip(range(nr_classes),classes):
```

Str Gheorghe Titeica nr 6, Sector 2,
Bucuresti
Romania

+4 031 4055287
+4 031 4012234
+4 0721 368 127

Website:
http://www.4esoft.com

email: office@4esoft.com

```
            TP[i] = np.logical_and( y_pred == c_class, y_truth == c_class ).sum()
            TN[i] = np.logical_and( y_pred != c_class, y_truth != c_class ).sum()
            FP[i] = np.logical_and( y_pred == c_class, y_truth != c_class ).sum()
            FN[i] = np.logical_and( y_pred != c_class, y_truth == c_class ).sum()
            class_pred[i] = TP[i] + FP[i]
            class_real[i] = TP[i] + FN[i]

        all_ex = TP[0]+TN[0]+FP[0]+FN[0]
        observed_accuracy = np.sum(TP) / all_ex
        expected_accuracy = (np.sum(class_pred*class_real) / all_ex) / all_ex
        kappa = (observed_accuracy - expected_accuracy) / \
                (1 - expected_accuracy)
        # conf_matrix !!!
        return kappa

    def ROC(self,y_prc,y_label, labels):
        nr_labels = len(labels)
        if y_label.ndim>1:
            y_label_list=y_label[:,0]
        thresholds = np.linspace(1, 0, 101)
        if nr_labels == 2:
            nr_ROCs = 1
        else:
            nr_ROCs = nr_labels

        TPR = np.zeros(shape=(101,nr_ROCs))
        FPR = np.zeros(shape=(101,nr_ROCs))
        AUC = np.zeros(shape=(nr_ROCs))

        for cROC in range(nr_ROCs):
            if nr_ROCs==1:
                c_label=1
            else:
                c_label = labels[cROC]
            for i in range(101):
                c_thr = thresholds[i]
                # Classifier / label agree and disagreements for current threshold.
                if i==50:
                    k=1
                TP = np.logical_and( y_prc[:,cROC] > c_thr, y_label_list==c_label ).sum()
                TN = np.logical_and( y_prc[:,cROC] <=c_thr, y_label_list!=c_label ).sum()
                FP = np.logical_and( y_prc[:,cROC] > c_thr, y_label_list!=c_label ).sum()
                FN = np.logical_and( y_prc[:,cROC] <=c_thr, y_label_list==c_label ).sum()
```

Str Gheorghe Titeica nr 6, Sector 2,
Bucuresti
Romania

+4 031 4055287
+4 031 4012234
+4 0721 368 127

Website:
http://www.4esoft.com

email: office@4esoft.com

```
                # Compute false positive rate for current threshold.
                FPR[i,cROC] = FP / float(FP + TN)

                # Compute true  positive rate for current threshold.
                TPR[i,cROC] = TP / float(TP + FN)

            # compute the AUC score for the ROC curve using the trapezoidal method
            AUC[cROC] = 0.
            for i in range(100):
                AUC[cROC] += (FPR[i+1,cROC]-FPR[i,cROC]) * (TPR[i+1,cROC]+TPR[i,cROC])

            AUC[cROC] *= 0.5


        return TPR,FPR, AUC

    def GetConfusionMatrix(self, y_pred,y_label):
        nr_preds = y_pred.size
        pred_classes = np.unique(y_pred)
        labl_classes = np.unique(y_label)
        all_classes  = np.unique(np.r_[pred_classes,labl_classes])

        conf_df = pd.DataFrame(index= all_classes ,columns = all_classes)
        conf_df.index.name = "Truth"
        for row in range(all_classes.size):
            for col in range(all_classes.size):
                c_preds = y_pred==all_classes[col]
                c_label = y_label == all_classes[row]
                val = np.logical_and( c_preds , c_label ).sum()
                conf_df.at[all_classes[row],all_classes[col]]= val

        return conf_df


    def add_observation(self, x,y):

        self.all_X = np.r_[self.all_X, x]
        self.all_y = np.append(self.all_y,y)

        return

    def get_train_obs(self):
        nr_all_x = self.all_X.shape[0]
        last_obs = np.arange(nr_all_x-self.batchsize,nr_all_x)
```

Str Gheorghe Titeica nr 6, Sector 2,
Bucuresti
Romania

+4 031 4055287
+4 031 4012234
+4 0721 368 127

Website:
http://www.4esoft.com

email: office@4esoft.com

```
        nr_obs = int(round(self.batchsize * self.back_train))

        extra_obs = nr_obs - self.batchsize

        if (nr_all_x >= nr_obs) and (extra_obs>0):
            old_idx = np.arange(0,nr_all_x -self.batchsize)
            np.random.shuffle(old_idx)
            all_obs = np.append(old_idx[:extra_obs],last_obs)
        else:
            all_obs = last_obs

        xi = self.all_X[all_obs,:]
        yi = self.all_y[all_obs]
        return xi,yi


    def prepare_x(self, x): # add intercept and poly feats

        # convert to a 1xN matrix if single observation
        x_temp = np.array(x,ndmin=2)
        x_prepared = np.array(x_temp)

        mini_batch_size = x_prepared.shape[0]
        ones_column = np.ones(shape=(mini_batch_size))
        x_prepared = np.c_[ones_column,x_prepared] # add intercept
        if self.polyfeats>1:
            for rank in range (2,self.polyfeats+1):
                x_prepared = np.c_[x_prepared, np.power(x_temp,rank)]

        return x_prepared

    def Calc_pValues(self):
        yHat,ydf = self.Predict(self.all_X)
        y = self.all_y
        X = self.all_X

        ## THIS IS NOT YET OK !
        sse = np.sum((yHat - y) ** 2, axis=0) / float(X.shape[0] - X.shape[1])
        #se = np.array([ np.sqrt(np.diagonal(sse[i] * np.linalg.inv(np.dot(X.T, X)))) for i in
range(sse.shape[0]) ])
        se = np.array([np.sqrt(np.diagonal(sse * np.linalg.inv(np.dot(X.T, X))))])
        self.t = self.Theta / se
        self.p = 2 * (1 - stats.t.cdf(np.abs(self.t), y.shape[0] - X.shape[1]))
```

Str Gheorghe Titeica nr 6, Sector 2,
Bucuresti
Romania

+4 031 4055287
+4 031 4012234
+4 0721 368 127

Website:
http://www.4esoft.com

email: office@4esoft.com

```
        return self


    def softmax(self,z):
        # z is MxK where M=observation K=classes
        # first shift the values of f so that the
        # highest number is 0:
        z -= np.max(z)

        ez = np.exp(z)

        p = (ez.T / np.sum(ez, axis=1)).T
        return p

    def sigmoid(self,z):
        return expit(z)
        #return 1 / (1 + math.exp(-z))


    def _simple_cross_entropy_loss(self, y, ht):
        # two class cross-entropy
        crs_entr_loss = (-1.0 )* np.sum(y*np.log(ht)+(1-y)*np.log(1-ht))
        return crs_entr_loss



    def CostFunctionLogistic(self,i_theta,x,y):
        # implements sigmoid logistic cost function and grad,
        # i_theta = current theta
        # works both for single and mini-batch updates !

        if self.method != 'sigmoid':
            raise Exception('Sigmoid cost function called from non-sigmoid classifier')

        if x.ndim != 2:
            raise Exception('Sigmoid function received x with ndim!=2')

        m = np.float64(x.shape[0])        # batch update size !!! NOT ALL OBS

        Theta = np.array(self.Theta[:,i_theta])
        Theta0 = np.array(Theta)
        Theta0[0] = 0
        xT = x.dot(Theta)
        HT = self.sigmoid(xT)
```

Str Gheorghe Titeica nr 6, Sector 2,
Bucuresti
Romania

+4 031 4055287
+4 031 4012234
+4 0721 368 127

Website:
http://www.4esoft.com

email: office@4esoft.com

```
        HT = np.clip(HT,self.eps,1-self.eps)
        H = HT - y

        # cost for linear regression
        # cCostRegression = (1/float(2))*np.power(H,2 )

        # cross-entropy cost function
        cCost1 = self._simple_cross_entropy_loss(y,HT)
        cCost2 = cCost1 / (float(m))
        # end cross-entropy cost function

        # ADD REGURALIZATION
        # default lmbd is 0 so no reg by default
        cCost = cCost2 + (self.lmbd) / (2 * m) * np.sum(Theta0.T.dot(Theta0))
        Grad = (1.0 / m) * x.T.dot(H)
        #ADD REGURALIZATION
        Grad += (self.lmbd / m) * Theta0

        if np.isnan(cCost):
            self.DebugInfo("[ERROR] NaN Cost",100)
            prev_grads = self.gradients[:,i_theta,:]
            sum_vector = np.sum(prev_grads, axis = 1)
            plt.plot(range(self.m-1),sum_vector[:-1])
            plt.show()
            cls = i_theta
            itr = self.m
            self.DebugInfo("[ERROR] Grad norm vect= {}".format(sum_vector),100)

            str_E  = "[ERROR]NaN cost Theta={} at batch no. {} ".format(cls,itr)
            str_E += "\nMaxX={:.2f} MinX={:.2f}".format(np.max(x),np.min(x))
            str_E += "\nMaxT={:.2f} MinT={:.2f}".format(np.max(Theta),np.min(Theta))
            str_E += "\nX.dot.Theta={}".format(xT)
            str_E += self.GetHyperParams()
            raise Exception(str_E)

        return Grad, cCost

    def LogisticTrain(self,xi, yi):
        ### 1 step stohastic logistic classifier training based on
        ### multi-class logistic

        # prepare gradient storage
        self.gradients = np.append(self.gradients,
```

Str Gheorghe Titeica nr 6, Sector 2,
Bucuresti
Romania

+4 031 4055287
+4 031 4012234
+4 0721 368 127

Website:
http://www.4esoft.com

email: office@4esoft.com

```
                                np.zeros(shape=(1,
                                                self._nr_thetas,
                                                self.n)), axis = 0)
        if yi.ndim>1:
            yi = np.ravel(yi)
        if self.MultiClass:
            #find right theta for each class !!!
            y_coded = np.empty(shape=(yi.size))
            cCosts = np.zeros(shape=(1,self.nr_Classes))
            for i in range(self.nr_Classes):
                y_coded.fill(0)
                if y_coded.size == 1:
                    y_coded[0] = (yi == self.Classes[i])
                else:
                    y_coded[yi==self.Classes[i]] = 1
                Grad, cCost = self.CostFunctionLogistic(i,xi,y_coded)
                ### store gradient
                self.gradients[-1,i,:] = Grad
                ### done store gradient
                self.Theta[:,i] = self.Theta[:,i] - self.alpha*Grad
                cCosts[0,i] = cCost

            self.Costs = np.append(self.Costs,cCosts,axis=0)
            J = np.empty((1,self.nr_Classes), float)
            J[0,:] = np.nanmean(self.Costs,axis=0)
            self.J_array = np.append(self.J_array, J, axis=0)
            ## done multi class
        else:
            ##
            ## now for single class
            ##
            i_c = 0
            Grad, cCost = self.CostFunctionLogistic(i_c,xi,yi)
            self.Theta[:,i_c] = self.Theta[:,i_c] - self.alpha*Grad
            self.Costs = np.append(self.Costs,cCost)
            ### store gradient
            self.gradients[-1,0,:] = Grad
            ### done store gradient
            J = np.sum(self.Costs)/self.Costs.shape[0]
            self.J_array = np.append(self.J_array, J)
            ##
            ## done single class
            ##
        return J
```

Str Gheorghe Titeica nr 6, Sector 2,
Bucuresti
Romania

+4 031 4055287
+4 031 4012234
+4 0721 368 127

Website:
http://www.4esoft.com

email: office@4esoft.com

```
def _log_loss(self,y,y_pred):
    ##
    ## Generalized cross-entropy. y input is a OneHot matrix
    ##
    J_matrix = y*np.log(y_pred)
    J =-np.sum(J_matrix)
    return J

def _log_loss_reg(self,y,y_pred,lmbd, theta, m):
    J_temp = self._log_loss(y,y_pred)
    # now apply ridge (L2) regularization
    J_temp = J_temp / m + 0.5 * lmbd * np.sum(theta*theta)
    return J_temp

def CostFunctionSoftmax(self,x, ohm_y):
    ### 1 step stohastic softmax training based on gradiend descent
    ### works BOTH for single observation and multiple observations
    ### y is not
    if self.method != 'softmax':
        raise Exception('Softmax function called from non-softmax classifier')

    if x.ndim != 2:
        raise Exception('Softmax function received x with ndim!=2')

    m = np.float64(x.shape[0])        # batch update size not all obs !!!


    Theta = np.array(self.Theta)
    xT = x.dot(Theta)
    yhat = self.softmax(xT)
    yhat = np.clip(yhat,self.eps,1-self.eps)
    # now final calc incl reguralization
    J = self._log_loss_reg(ohm_y,yhat, self.lmbd, Theta, m)
    self.LastGThe = Theta
    self.LastM = m

    TempG = ohm_y - yhat
    self.LastYERR = TempG
    self.LastYHAT = yhat
    self.LastYOHM = ohm_y
    self.LastObs = x
    Grad = (-1.0/m) * x.T.dot(TempG)
```

Str Gheorghe Titeica nr 6, Sector 2,
Bucuresti
Romania

+4 031 4055287
+4 031 4012234
+4 0721 368 127

Website:
http://www.4esoft.com

email: office@4esoft.com

```
        Grad += self.lmbd*Theta



        return Grad, J

    def SoftmaxTrain(self,xi,yi):



        cur_m = xi.shape[0]
        SparseBoolLabels = np.zeros(shape=(cur_m,self.nr_Classes))
        softmax_y = np.zeros(shape=(cur_m,1))

        for i,k in zip(range(self.nr_Classes),self.Classes):
            where_y = np.where(yi==k)
            if where_y[0].size>0:
                if len(where_y)>1:
                    row, column = where_y
                else:
                    row = where_y
                softmax_y[row] = i+1
                # now calculate 1(Yi==col) sparse boolean matrix
                SparseBoolLabels[row,i] = 1

        """
        else: ???
            self.SparseBoolLabels  = np.zeros(shape=(1,self.nr_Classes))
            self.SparseBoolLabels[self.Classes==yi] = 1
        """
        # prepare gradient storage
        self.gradients = np.append(self.gradients,
                                   np.zeros(shape=(1,
                                                   self.nr_Classes,
                                                   self.n)), axis = 0)

        if self.alpha_search_iter == 200:
            test_stop = True
        ### now the gradient descent step
        if self.softmax_alpha_search:
            if self.alpha_search_iter < self.alpha_search_epochs:
                bestAlpha = self.SearchBestAlpha(xi,SparseBoolLabels, Verbose = False)
                self.alpha = bestAlpha
                self.BestAlphas.append(bestAlpha)

        Grad, J = self.CostFunctionSoftmax(xi,SparseBoolLabels)
```

Str Gheorghe Titeica nr 6, Sector 2,
Bucuresti
Romania

+4 031 4055287
+4 031 4012234
+4 0721 368 127

Website:
http://www.4esoft.com

email: office@4esoft.com

```
        self.LastGrad = Grad
        self.LastJ = J
        ### store gradient
        self.gradients[-1,:,:] = Grad.T
        ### done store gradient
        self.Theta = self.Theta - self.alpha*Grad
        self.LastAlph = self.alpha
        self.LastThet = self.Theta

        self.Costs = np.append(self.Costs,J)
        Jmean = np.sum(self.Costs)/self.Costs.shape[0]
        self.J_array = np.append(self.J_array, Jmean)

        return J

    ##
    ##  Stohastic Gradient Check with option of selecting best weights
    ##  based on cross dataset
    ##
    def OnlineTrain(self,x_input, y_input, X_cross = None, y_cross = None):

        """ obsolete since mini batch
        ## data comes in only 1 dim
        if x_input.shape[0]!=(self.original_n-1):
            raise Exception("Check your data ! x has wrong size (expected="+str(self.n-1)+'
received='+x_input.shape[1]+')')
        """

        xi = self.prepare_x(x_input)
        self.batchsize = xi.shape[0]

        yi = np.array(y_input)

        ## now add the observation to observation matrices
        self.add_observation(xi,yi)


        ## now prepare actual Xi
        xi,yi = self.get_train_obs()
```

Str Gheorghe Titeica nr 6, Sector 2,
Bucuresti
Romania

+4 031 4055287
+4 031 4012234
+4 0721 368 127

Website:
http://www.4esoft.com

email: office@4esoft.com

```
        if (self.m == 0):
            self.DebugInfo("[DEBUG] Beginning training: "+self.GetShortHyperParams(), 10)

        #now increment nr of received examples
        self.m+=1

        if (self.m % 100) ==0:
            self.DebugInfo("[DEBUG]     Training the observation/batch nr {}".format(self.m),
10)



        if self.method == 'sigmoid': # sigmoid/logistic single or multi class
            J = self.LogisticTrain(xi,yi)
        elif self.method == 'softmax': # softmax
            J = self.SoftmaxTrain(xi,yi)

        self.alpha_array = np.append(self.alpha_array, self.alpha)

        if self.DecreasingAlpha:
            N = 1 #batch size
            if self.alpha_coef>0:
                self.alpha = self.alpha * self.alpha_coef
            else:
                self.alpha = float(self.alpha_0) / (1.0 + (self.m / N ))

        ##
        ## now cross check
        ##
        if (not (X_cross is None)) and (not (y_cross is None)):
            y_cross_pred,ydf = self.Predict(X_cross)
            y1 = np.ravel(y_cross_pred)
            y2 = np.ravel(y_cross)
            preds = y1 == y2
            my_pred = (np.sum(preds)/float(X_cross.shape[0]))*100
            if my_pred>=self.BestAccuracy:
                self.BestAccuracy = my_pred
                self.BestTheta = np.array(self.Theta)
                self.BestFeed = self.m
        else:
            self.BestTheta = np.array(self.Theta)
            self.BestFeed = self.m



        return J
```

Str Gheorghe Titeica nr 6, Sector 2,
Bucuresti
Romania

+4 031 4055287
+4 031 4012234
+4 0721 368 127

Website:
http://www.4esoft.com

email: office@4esoft.com

```
def _PredictSoftmax(self,X,Theta,real_y):
    y = None
    y_floats = None
    m=X.shape[0]
    X = self.prepare_x(X) #  add poly, intercept

    y_floats = self.softmax(X.dot(Theta))
    y_indices = np.argmax(y_floats,axis=1)
    y = np.empty((0,1))
    for i,label in zip(range(m),y_indices):
        y = np.append(y,np.array([[self.Classes[label]]]),axis=0)

    return y,y_floats

def _PredictSigmoid(self,X,Theta,real_y):
    m=X.shape[0]
    X = self.prepare_x(X) #  add poly, intercept

    if self.MultiClass:
        # predict multi class
        y_floats = self.sigmoid(X.dot(Theta))
        y_indices = np.argmax(y_floats,axis=1)
        y = np.empty((0,1))
        for i,label in zip(range(m),y_indices):
            y = np.append(y,np.array([[self.Classes[label]]]),axis=0)
    else:
        # predict single class 0/1
        y_floats = self.sigmoid (X.dot(Theta))
        y = np.round(y_floats)

    return y, y_floats

def Predict(self, X,real_y = None, Best = False, return_floats = False, return_df = True):
    if Best:
        if self.BestTheta is None:
            raise Exception("BestTheta is "+self.BestTheta.tostring())
        Theta = self.BestTheta
    else:
        Theta = self.Theta

    if self.method == 'sigmoid':
        y,y_floats = self._PredictSigmoid(X,Theta,real_y)
    elif self.method == 'softmax':
```

Str Gheorghe Titeica nr 6, Sector 2,
Bucuresti
Romania

+4 031 4055287
+4 031 4012234
+4 0721 368 127

Website:
http://www.4esoft.com

email: office@4esoft.com

```
            y,y_floats = self._PredictSoftmax(X,Theta,real_y)
        else:
            raise Exception("Unknown method: "+self.method)

        if (self.Classes == self._standard_binary_classes) and \
           (self.nr_Classes==2):

            if self.method == 'softmax':
                columns=["0","1"]
            else:
                columns=["0/1"]

        else:
            columns = self.Classes
        y_df = pd.DataFrame(y_floats,columns=columns)

        if not (real_y is None):
            y_df['Real Y'] = real_y

        y = np.ravel(y)

        if return_floats:
            return y,y_df,y_floats
        else:
            if return_df:
                return y,y_df
            else:
                return y

###
### End OnlineClassifier class
###




if __name__ == '__main__':
    raise Exception("Class only file")
```