

""

INEX 2009 - Book Structure Track - Quality Evaluation Software

Jean-Luc Meunier - XRCE - 2009

This is a complementary metric that measures the quality of the links in first intention, rather than conditionally to the title validity as in Inex08.

It computes the precision and recall of the links, by considering a submission as a list of page breaks, ordered by the page number of the breaks. When a link is valid, it computes the similarity of the proposed title with the ground truth one, using the Inex weighted Levenshtein distance.

Another interest lies in the production of one result per book, with possibly a detailed display of the valid, wrong, missed links.

In addition, it produces a measure close to the Inex link metric, by (Inex-)matching the title of valid links against the ground truth. So this last measure is similar to the "Inex links" apart that the link validity is checked before the title validity. If both are valid, the entry is valid, according to that Inex-like measure.

Note: the title comparisons are done after lowercase normalization.

EVALUATION METHOD:

- for each book:
 - load the toc-entries
 - sort them by increasing page number
 - compare the page number of the reference and submitted lists of entries following the increasing page order
 - this provides for each book a number of valid/missed/wrong links, and then a measure of precision/recall/F1
 - then we consider only the valid links, i.e. same page:
 - a first title quality is computed by turning the Inex weighted Levenshtein distance into a similarity measure:
$$\text{similarity} = 1 - \text{weightedLevenshtein}(s1, s2) / (\max(\text{weight}(s1), \text{weight}(s2)))$$

the book title accuracy is the average of the title quality of its valid links
 - we then review all valid links.
 - Each valid link that does not Inex-match the groundtruth title is counted as one miss + one error.
 - (The Inex-match puts constraints on the begin, whole and end of the title, see the specification document at <http://inex.otago.ac.nz/tracks/books/INEXBookTrackSEMeasures.pdf>)
 - (We have implemented it here in the InexTitle class)
 - We compute another precision/recall/F1, which is almost equivalent to the Inex book track official "links" measure.
 - [NOTE: it can differ slightly when several links point to the same page, because this SW does look for the best alignment of the links title against the groundtruth, and therefore, this measure can provide lower values than the official one. The observed difference is at maximum 1.2% .]
 - a second title accuracy is computed using the INEX-match function, and averaged over the valid links to produce a book measure.
 - a level accuracy measure is averaged over the valid links of a book.
 - a relative level accuracy is now (Dec.15th 2009) computed. Basically, if a whole sub-tree is moved at a wrong level, it counts for only one error.


```

import xml.dom.minidom

#let's include the code of inex_title in this module to ease the code distribution
#import inex_title

class InexException(Exception):
    pass

#Utility
def trace(*msg):
    for i in msg:
        sys.stdout.write(i.encode('utf-8'))
def traceln(*msg):
    apply(trace, msg)
    trace("\n")
    sys.stdout.flush()

#-----
---

class Submission:

    def __init__(self, dom, sName, bStrict=False):
        """
        load a submission DOM
        if bStrict is True, strict conformance to the schema is required and in
particular, page number attributes must be valued
        so if a page number is missing we complain and we set it to 0 rather than
ignoring the entry
        """
        self.sName = sName
        self.dom = dom
        self.lBook = []
        self.dBook = {}
        for ndBook in self.dom.getElementsByTagName("book"): #load all the books
            bk = Book(sName, ndBook, bStrict)
            self.dBook[bk.id] = bk
            self.lBook.append(bk)
            #trace(".")
        traceln(" loaded %6d books\n"%len(self.lBook))

    def compare(cls, sub1, sub2, bVerbose=False):
        """
        Compute the macro-precision/recall/F1 of the submission, showing also the
P/R/F1 for each book
        This is done for the Xerox and Inex-like link measures
        now we also enumerate non-matching titles and check the level accuracy

        returns the raw data
        """
        nOk, nErr, nMiss, nOkTitle, nOkLevel, nOkRelLevel = 0, 0, 0, 0, 0, 0
        cntComparedBook, cntMissingBook, cntMissingBookEntries, cntAdditionalBooks = 0,
0, 0, 0

        lBook1 = sub1.lBook
        trace("Comparison: ")
        if len(lBook1) != len(sub2.lBook):

```

```

        traceln("\t *** WARNING ***: %d books vs %d books"%(len(lBook1),
len(sub2.lBook)))
    else:
        traceln("\t%d books"%len(lBook1))

    if not bVerbose:
        traceln()
        traceln(cls.formatTitle()+" book id\n")

    ltFileOkErrMissOkTokL = []
    ltAll = []
    for i, book1 in enumerate(lBook1):
        num = i+1
        if bVerbose:
            traceln()
            traceln("==== %2d Book: %s ====="%(num, book1.id))
        try:
            book2 = sub2.dBook[book1.id] #the books are identified by their id
        except KeyError:
            if bVerbose: trace( " = = = ** book %s missing from submission"%
book1.id)
            nOkBook, nErrBook, nMissBook, nOkTitleBook, nOkLevelBook,
fSumSimilTitleBook, nOkRelLevelBook = 0, 0, book1.getNbEntry(), 0, 0, 0.0, 0
            if bVerbose: traceln(" inducing %d missed entries "%(nMissBook))
            cntMissingBook += 1
            cntMissingBookEntries += nMissBook
        else:
            nOkBook, nErrBook, nMissBook, nOkTitleBook, nOkLevelBook,
fSumSimilTitleBook, nOkRelLevelBook = Book.compare(book1, book2, bVerbose)
            book2.bProcessed = True
            cntComparedBook += 1
        nOk += nOkBook
        nErr += nErrBook
        nMiss += nMissBook
        nOkTitle += nOkTitleBook #this uses the INEX-match function
        nOkLevel += nOkLevelBook
        nOkRelLevel += nOkRelLevelBook

        #----- Measure per book
        #--- Xerox-measure
        fPxx, fRxx, fFxx = computePRF(nOkBook, nErrBook, nMissBook)
        # title and level accuracy
        if nOkBook != 0:
            fAccTitleXrx = fSumSimilTitleBook / nOkBook #average title
similarity in percentage
            fAccTitleInex = float(nOkTitleBook) / nOkBook
            fAccLevel = float(nOkLevelBook) / nOkBook
            fAccRelLevel = float(nOkRelLevelBook) / nOkBook
        else: #an absence of response
            fAccTitleXrx = None
            fAccTitleInex = None
            fAccLevel = None
            fAccRelLevel = None

        #--- kind of Inex08-link measure
        # we consider here that a non-matching title removes a valid entry and
create both an error and a miss
        nMismatch = nOkBook - nOkTitleBook
        fPinx, fRinx, fFinx = computePRF(nOkTitleBook, nErrBook+nMismatch,

```

```

nMissBook+nMismatch) #Inex-like-link-measure
    if bVerbose:
        tracen()
        tracen(cls.formatTitle()+" book id\n")
#Precision, Recall, F, Accuracy, Accuracy for the XRCE and the INEX
measures
    tPRFAA = (fPxrx, fRxrx, fFrxr, fAccTitleXrx, fAccRelLevel, fPinx, fRinx,
fFinx, fAccTitleInex, fAccLevel)
    #Count of the valid/wrong/missed links, of the relative-level errors,
title errors, level errors
    tCount = (nOkBook, nErrBook, nMissBook, nOkBook-nOkRelLevelBook, nOkBook-
nOkTitleBook, nOkBook-nOkLevelBook)
    tracen("%s %s"%(cls.formatResult(str(num), tPRFAA, tCount), book1.id))
    ltAll.append( (num, tPRFAA, tCount, book1.id) ) #keep the result in
memory

    #Some additional book??
    for book2 in sub2.lBook:
        try:
            book2.bProcessed
        except AttributeError:
            if bVerbose: tracen( " = = = ** additional book %s (not in
groundtruth) - IGNORED"% book2.id)
            cntAdditionalBooks += 1

    tracen("\nDone : %d comparisons for %d books in %s and %d in %s"%
(cntComparedBook

, len(sub1.lBook), sub1.sName

, len(sub2.lBook), sub2.sName))
    if cntMissingBook : tracen("\n\t%d books missing from submission (%d misses
in total) "%(cntMissingBook, cntMissingBookEntries))
    if cntAdditionalBooks: tracen("\n\t%d additional books in submission
(ignored) "%cntAdditionalBooks)

#==== RESULTS !
if bVerbose:
    tracen("=====")
    tracen(cls.formatTitle()+" book id\n")
#Here we compute the average and standard deviation of the first 10 columns of
values
nbFloat = 10
nbInt = 6
lfSX = [0.0] * nbFloat + [0]*nbInt #to compute the average over the float and
the sum over the int
lfSX2 = [0.0] * nbFloat #to compute the standard deviation
liCNT = [0] * (nbFloat+nbInt)
for (num, tPRFAA, tCount, bookid) in ltAll:
    if bVerbose: tracen("%s\t%s"%(cls.formatResult(str(num), tPRFAA,
tCount), bookid))
    t = tPRFAA+tCount
    assert nbFloat+nbInt == len(t)
    for i in range(nbFloat+nbInt):
        f = t[i]
        if f != None:
            lfSX [i] = lfSX [i] + f
            if i < nbFloat: lfSX2[i] = lfSX2[i] + f*f
            liCNT[i] += 1
#compute the average of all those values

```

```

lfAvg = [None] * nbFloat
lfSDv = [None] * nbFloat
for i in range(nbFloat):
    if liCNT[i] > 0:
        cnt = liCNT[i]
        lfAvg[i] = lfSX[i] / cnt
        aux = lfSX2[i] / cnt - lfAvg[i]*lfAvg[i]
        if cnt > 1: aux = aux * cnt / (cnt-1)
        lfSDv[i] = math.sqrt( aux )
liSum = lfSX[-nbInt:]
traceln("-"*80)
traceln(cls.formatResult("AVG", tuple(lfAvg), tuple(liSum)))
traceln()
traceln(cls.formatResult("sdev", tuple(lfSDv)))

traceln()
traceln(cls.formatTitle())

#return nOk, nErr, nMiss
return ltAll, tuple(["all"]+lfAvg+liSum)
compare = classmethod(compare)

#Displaying things properly in columns... boring code...
def formatTitle(cls, bLong=True):
    """
    the labels of the report column(s)
    """
    s = " "*6+"Xrx-measure Links"+" "*2+"Title"+" "*2+"r-lvl"+" |"+" "*5+"~Inex08
Links"+" "*2+"Title Level\n"
    s += " Doc "
    s += "%6s%6s%6s" % ("Prec", "Rec.", "F1") + " %6s%" "acc." + " %6s%" "acc." + " | "
    s += "%6s%6s%6s" % ("Prec", "Rec.", "F1")
    s += " "
    s += "%6s%6s " % ("acc.", "acc.")
    if bLong: s += "|%7s%7s%7s%7s %7s%7s" % ("Ok", "Err", "Miss", "PbRLvl",
"PbITtl", "PbLvl")
    return s
formatTitle = classmethod(formatTitle)

def formatResult(cls, docnum
                 , (fPxx, fRxx, fFxx, fAccTitleXrx , fAccRelLevel
                 , fPinx, fRinx, fFinx, fAccTitleInex, fAccLevel)
                 , tCnt=None):
    s = "%4s "%docnum
    s += cls.formatPRF(fPxx, fRxx, fFxx)
    s += " "
    s += "%6s%"cls.formatFloatPercent(fAccTitleXrx)
    s += " "
    s += "%6s%"cls.formatFloatPercent(fAccRelLevel)
    s += " | "
    s += cls.formatPRF(fPinx, fRinx, fFinx)

    s += " "
    s += "%6s%"cls.formatFloatPercent(fAccTitleInex)
    s += "%6s%"cls.formatFloatPercent(fAccLevel)

    if tCnt:
        nOk, nErr, nMiss, nPbRelLevel, nPbInexTitle, nPbLevel = tCnt

```

```

        s += " |"
        s += "%7s%7s%7s%7s"%(nOk, nErr, nMiss, nPbRelLevel)
        s += " %7s%7s"%(nPbInexTitle, nPbLevel)
    return s
formatResult = classmethod(formatResult)

def formatPRF(cls, fP, fR, fF):
    s = ""
    for f in [fP, fR, fF]:
        s += "%6s"%cls.formatFloatPercent(f)
    return s
formatPRF = classmethod(formatPRF)

def formatFloatPercent(selfcls, f, sFmt="%.1f"):
    if f == None:
        return "N/A"
    else:
        return sFmt%(100*f)
formatFloatPercent = classmethod(formatFloatPercent)

#-----
---
```

```

class Book:
    def __init__(self, sName, ndBook, bStrict=False):
        """
        load a book from its root node
        """
        self.node = ndBook
        self.id = getNodeText(self.node.getElementsByTagName("bookid")
[0]).strip().lower()

        #load the toc entries
        try:
            self.lTocEntry = TocEntry.deepWalk(self.node.childNodes, bStrict)
        except InexException:
            #let's go on
            traceIn(" in book %s "%self.id)

        #sort by page number
        self.lTocEntry.sort(TocEntry.cmpLink)

    def getNbEntry(self):
        return len(self.lTocEntry)

    def compare(cls, bk1, bk2, bVerbose):
        """
        compare a reference book data to another one (bk1 is the groundtruth)
        return the number of ok, wrong, missed entries
            + the number of Inex-matching titles among the correct links
            + the number of valid levels among the correct links
            + the sum the title similarity over the correct links
            + the number of valid "relative levels"
        """
        ltTitleMismatches = []
        l1 = bk1.lTocEntry
        l2 = bk2.lTocEntry

        n1, n2 = len(l1), len(l2)

```

```

nOk, nErr, nMiss, nOkTitle, nOkLevel = 0, 0, 0, 0, 0
fSumSimilTitle = 0.0 #sum of the title similarity for valid links
nbUnion = 0
ltAlignment = [] #list of alignment between the groundtruth and the submission

i1, i2 = 0, 0
if n1 and n2:
    e1, e2 = l1[i1], l2[i2]
    while 1:
        d = TocEntry.cmpLink(e1, e2)
        try:
            if d == 0: #valid link, let's check the level and title
                nOk += 1
                if TocEntry.inexMatchTitle(e1, e2):
                    nOkTitle += 1
                else:
                    if bVerbose: traceln( " title error: %s <-->
'%s'"% (e1, e2.title) )

                    if TocEntry.cmpLevel(e1, e2) == 0:
                        nOkLevel += 1
                    fSumSimilTitle += TocEntry.similTitle(e1, e2)
                    ltAlignment.append( (e1, e2) ) #there are aligned!
                    i1 = i1 + 1
                    i2 = i2 + 1
                    e1 = l1[i1]
                    e2 = l2[i2]
            else:
                if d < 0:
                    nMiss += 1
                    if bVerbose: traceln( "\tMISS : %s"% e1 )
                    i1 = i1 + 1; e1 = l1[i1]
                else:
                    nErr += 1
                    if bVerbose: traceln( "\tERROR: %s"%e2 )
                    i2 = i2 + 1; e2 = l2[i2]
        except IndexError:
            break

while i1 < n1:
    if bVerbose: traceln( "\tMISS : %s"% l1[i1] )
    i1 += 1
    nMiss += 1
while i2 < n2:
    if bVerbose: traceln( "\tERROR: %s"% l2[i2] )
    i2 += 1
    nErr += 1
assert len(ltAlignment) == nOk

#Now deal with the notion of relative levels
#we look for each alignments, starting by top level nodes from the groundtruth
#when the submission is not at proper level, we fix the level of the whole
subtree and count one error
nOkRelLevel = 0
ltAlignment.sort(key=lambda (te1,te2): te1.level) #sort by increasing level
for (te1,te2) in ltAlignment:
    if te1.level == te2.level:
        nOkRelLevel += 1
    else:

```

```

        #ok, let's fix the level of the whole subtree!
        #trace("*** ")
        te2.setSubTreeLevel(te1.level)
        #traceln(str((te1.level, te1.title, te2.level, te2.title)))

    return nOk, nErr, nMiss, nOkTitle, nOkLevel, fSumSimilTitle, nOkRelLevel
compare = classmethod(compare)

#-----
---

class TocEntry:
    DELTA = 0
    def __init__(self, node, lvl, bStrict=True):
        self.title = node.getAttribute("title")
        self.level = lvl
        try:
            self.page = int(node.getAttribute("page"))+TocEntry.DELTA
        except ValueError, e:
            if bStrict:
                #traceln("page defaulted to 0 ", node.toxml())
                self.page = 0
            else:
                traceln()
                traceln("\n** WARNING **: ignoring entry with missing page
number :\n", node.toxml() )
                raise InexException("invalid page")
        self.lteChild = []

    def addChild(self, te):
        """
        record one more child to this TocEntry
        """
        self.lteChild.append(te)

    def setSubTreeLevel(self, level):
        """
        recursively set a new level
        """
        self.level = level
        for te in self.lteChild: te.setSubTreeLevel(level+1)

    def cmpLink(cls, te1, te2):
        """
        compare the page number referred to by two Toc links
        return -1, 0 or +1 as any compare operator in Python
        """
        # if abs(te1.page - te2.page) <= 2:
        #     return 0
        # else:
        #     return cmp(te1.page, te2.page)
        return cmp(te1.page, te2.page)
    cmpLink = classmethod(cmpLink)

    def cmpLevel(cls, te1, te2):
        """
        compare the level f the two toc entries
        return -1, 0 or +1 as any compare operator in Python
        """

```

```

        return cmp(te1.level, te2.level)
    cmpLevel = classmethod(cmpLevel)

    def inexMatchTitle(cls, te1, te2):
        """
        compare the title a la Inex08
        """
        return InexTitle.match(te1.title.lower(), te2.title.lower())
    inexMatchTitle = classmethod(inexMatchTitle)

    def similTitle(cls, te1, te2):
        """
        This is a XRCE addition to the metrics specified by the Inex book track.
        Evaluate the similarity between the two strings, defined as 1 - the ratio of
the
        Inex-Levenstein distance to the highest Inex-weight of the two strings.
        If both strings are empty, the similarity is 1.
        return a float in [0, 1]
        """
        return InexTitle.similarity(te1.title, te2.title) #similarity does the .lower()
    similTitle = classmethod(similTitle)

    def deepWalk(cls, lNode, bStrict, teParent=None, lvl=0):
        """
        visit recursively the listed nodes, looking for toc entries
        we also update the child list of each TocEntry
        """
        lTE = list()
        for node in lNode:
            if node.nodeName == "toc-entry":
                teNode = TocEntry(node, lvl+1, bStrict)
                if teParent: teParent.addChild(teNode) #tree downward links
                lTE.append(teNode)
                lTE.extend(cls.deepWalk(node.childNodes, bStrict, teNode, lvl+1))
        return lTE
    deepWalk = classmethod(deepWalk)

    def __str__(self):
        return "page=%4d title='%s'"%(self.page, self.title)

#----- Utilities
-----

def getNodeText(node):
    """
    return the concatenated text of children text nodes of @node
    """
    nodelist = node.childNodes
    rc = ""
    for node in nodelist:
        if node.nodeType == node.TEXT_NODE:
            rc = rc + node.data
    return rc

def computePRF(nOk, nErr, nMiss):
    """

```

```

    compute the precision/recall/F1 measure given the number of correct, wrong, missed
items
return a tuple of 3 values, being either a float in [0, 1] or None (if not defined)
"""
try:
    fP = nOk / float(nOk+nErr)
except ZeroDivisionError:
    #fP = None
    fP = 0.0
try:
    fR = nOk / float(nOk+nMiss)
except ZeroDivisionError:
    #fR = None
    fR = 0.0
if fP != None and fR != None:
    try:
        fF = 2 * fP * fR / (fP+fR)
    except ZeroDivisionError:
        fF = 0.0
else:
    fF = None
return fP, fR, fF

```

```

#----- Inex Book Track 2008 Title Matching Concepts
-----

```

```

class InexTitle:

```

```

    """

```

```

    a class to deal with the Inex 2008 title matching

```

```

    http://inex.otago.ac.nz/tracks/books/INEXBookTrackSEMeasures.pdf

```

```

    EXCERPT FROM "INEX Book Track - Results evaluation method description" 2008

```

```

    -----

```

```

    Titles - Titles match if the distance1 between two title strings is less than 20%
of the shorter string and if

```

```

    the distance of first and last five characters (or less if the string is shorter)
is less than 60%. This method

```

```

    of measurement has been introduced to reduce the impact of the OCR errors. This
method has been

```

```

    empirically verified to produce best results against OCR errors minimizing false
positives.

```

```

    1 Distance is calculated using a modified Levenshtein algorithm. The modification
consists of the following: The

```

```

    cost of alphanumeric substitution, deletion and insertion is 10. The cost of non-
alphanumeric substitution, deletion

```

```

    and insertion remains 1. Distance between two strings A and B in percent is
calculated as

```

```

    D= LevenshteinDist * 10 / Min(A.len, B.len)    [%]

```

```

    -----

```

```

    """

```

```

    fPRCNT_TITLE          = 20

```

```

    fPRCNT_HEADTAIL      = 60

```

```

    iHEADTAIL            = 5  #a constant: first and last characters

```

```

    iCOSTALNUM           = 10

```

```

iCOSTOTHER          = 1

fPOSITIVE_INFINITY = 9e99

def match(cls, sTitle1, sTitle2):
    """
    return a boolean, given the two string parameter
    this is the official Inex match function.
    """

    n1, n2 = len(sTitle1), len(sTitle2)

    sHead1, sTail1 = sTitle1[:cls.iHEADTAIL], sTitle1[-cls.iHEADTAIL:] #note: these
string may be shorter than iHEADTAIL
    sHead2, sTail2 = sTitle2[:cls.iHEADTAIL], sTitle2[-cls.iHEADTAIL:] #note: these
string may be shorter than iHEADTAIL

    return cls.distance(sHead1, sHead2) < cls.fPRCNT_HEADTAIL \
        and cls.distance(sTail1, sTail2) < cls.fPRCNT_HEADTAIL \
        and cls.distance(sTitle1, sTitle2) < cls.fPRCNT_TITLE
match = classmethod(match)

def distance(cls, s1, s2):
    """
    return the distance between the two string as a float in [0.0,
positive_infinite]
    as 10 times the weighted levenshtein distance divided by the length of the
shortest string
    """
    if s1 and s2:
        d = 10 * cls.weighted_levenshtein(s1, s2) / float(min(len(s1), len(s2)))
    else:
        return cls.fPOSITIVE_INFINITY
    return d
distance = classmethod(distance)

def similarity(cls, s1, s2):
    """
    This is a XRCE addition to the metrics specified by the Inex book track.
    Evaluate the similarity between the two strings, once converted to lowercase,
    defined as 1 - the ratio of the Levenstein distance defined in Inex Book track
    to the highest weight of the two strings.
    If both strings are empty, the similarity is 1.
    return a float in [0, 1]
    """
    s1, s2 = s1.lower(), s2.lower()
    w1 = sum(map(cls.weight, s1)) #sum of the weights of each character in the
string
    w2 = sum(map(cls.weight, s2)) #sum of the weights of each character in the
string

    mxw = max(w1, w2)
    if mxw!=0:
        return 1.0 - cls.weighted_levenshtein(s1, s2) / float(mxw)
    else:
        return 1.0
similarity = classmethod(similarity)

def weight(cls, c):
    if c.isalnum():

```

```

        return cls.iCOSTALNUM
    else:
        return cls.iCOSTOTHER
weight = classmethod(weight)

def weighted_levenshtein(cls, s1, s2):
    """
    code taken from
    http://en.wikibooks.org/wiki/Algorithm\_implementation/Strings/Levenshtein\_distance
    and modified to account for the costs chosen for Inex
    """

    if len(s1) < len(s2):
        return cls.weighted_levenshtein(s2, s1)
    #if not s1:
    #    return len(s2)

    #previous_row = xrange(len(s2) + 1)
    previous_row, sumw2 = [0], 0
    for c2 in s2:
        sumw2 += cls.weight(c2)
        previous_row.append(sumw2)
    sumw1 = 0
    for i, c1 in enumerate(s1):
        w1 = cls.weight(c1)
        sumw1 += w1
        current_row = [sumw1]
        for j, c2 in enumerate(s2):
            w2 = cls.weight(c2)
            insertions = previous_row[j + 1] + w1
            deletions = current_row[j] + w2
            substitutions = previous_row[j] + (c1 != c2)*max(w1,w2) #if one of
them is alphanumeric, the substitution as alphanumeric
            current_row.append(min(insertions, deletions, substitutions))
        previous_row = current_row

    return previous_row[-1]
weighted_levenshtein = classmethod(weighted_levenshtein)

def selfTest(cls):
    global cnt #convenience, not so clean, but was in a separate module initially
    traceln("SELF TESTS")
    cnt = 0
    def test(fun, s1, s2, refret):
        global cnt
        cnt += 1
        ret = fun(s1, s2)
        trace(".")
        if refret != ret:
            traceln("\nERROR: %s(%s, %s) == %s != %s"%(str(fun), `s1`, `s2`,
`ret`, `refret`))
            return False
        else:
            return True

    def symmetric_test(fun, s1, s2, refret):
        global cnt

```

```

    bRet = True
    for (u, v) in [ (s1, s2), (s2, s1) ]:
        bRet = bRet and test(fun, u, v, refret)
    return bRet

#-----
def reverseString(s):
    l = list(s)
    l.reverse()
    return string.join(l, '')
assert reverseString("") == ""
assert reverseString("a") == "a"
assert reverseString("ab") == "ba"
assert reverseString("abcdef") == "fedcba"

#-----
def test_weighted_levenshtein(s1, s2, refret):
    b1 = symmetric_test(cls.weighted_levenshtein, s1
, s2
, refret)
    b2 = symmetric_test(cls.weighted_levenshtein, reverseString(s1)
reverseString(s2)
, refret)
    return b1 and b2

assert test_weighted_levenshtein("a", "a", 0)
assert test_weighted_levenshtein("1", "1", 0)
assert test_weighted_levenshtein(" ", " ", 0)

assert test_weighted_levenshtein("a", "b", cls.iCOSTALNUM)
assert test_weighted_levenshtein("1", "2", cls.iCOSTALNUM)
assert test_weighted_levenshtein(" ", ".", cls.iCOSTOTHER)
assert test_weighted_levenshtein("a", "2", cls.iCOSTALNUM)
assert test_weighted_levenshtein(".", "a", cls.iCOSTALNUM)
assert test_weighted_levenshtein("a", ".", cls.iCOSTALNUM)

assert test_weighted_levenshtein("a", "a2", cls.iCOSTALNUM)
assert test_weighted_levenshtein("a", "a.", cls.iCOSTOTHER)
assert test_weighted_levenshtein("1", "12", cls.iCOSTALNUM)
assert test_weighted_levenshtein("1", "1.", cls.iCOSTOTHER)
assert test_weighted_levenshtein(".", ".2", cls.iCOSTALNUM)
assert test_weighted_levenshtein(".", "..", cls.iCOSTOTHER)

assert test_weighted_levenshtein("a.", "a2", cls.iCOSTALNUM)
assert test_weighted_levenshtein("a;", "a.", cls.iCOSTOTHER)
assert test_weighted_levenshtein("13", "12", cls.iCOSTALNUM)
assert test_weighted_levenshtein("1;", "1.", cls.iCOSTOTHER)
assert test_weighted_levenshtein(".a", ".2", cls.iCOSTALNUM)
assert test_weighted_levenshtein(".;", "..", cls.iCOSTOTHER)

assert test_weighted_levenshtein("a.z", "a2zz", cls.iCOSTALNUM)
assert test_weighted_levenshtein("a;zz", "a.z", cls.iCOSTOTHER)
assert test_weighted_levenshtein("13zz", "12zz", cls.iCOSTALNUM)
assert test_weighted_levenshtein("1;zz", "1.z", cls.iCOSTOTHER)
assert test_weighted_levenshtein(".azz", ".2zz", cls.iCOSTALNUM)
assert test_weighted_levenshtein(".;zz", "..zz", cls.iCOSTOTHER)

assert test_weighted_levenshtein(";z", "z;", cls.iCOSTOTHER*4)
assert test_weighted_levenshtein(";zz", "zz;", cls.iCOSTOTHER*4)
assert test_weighted_levenshtein(":::::::::z", "z:::::::::",
cls.iCOSTALNUM*2)

```

```

        assert test_weighted_levenshtein(":::::::::zz", "zz:::::::::",
cls.iCOSTOTHER*24)
        println()

#-----
def test_distance(s1, s2, refret):
    b1 = symmetric_test(cls.distance, s1, s2, refret)
    b2 = symmetric_test(cls.distance, reverseString(s1), reverseString(s2), refret)
    return b1 and b2

assert test_distance("", "a", cls.fPOSITIVE_INFINITY)
assert test_distance("a", "a", 0.0)
assert test_distance("a", "b", 100.0)
assert test_distance("a", "ab", 100.0)
assert test_distance("a", "abc", 200.0)
assert test_distance("a", "abcd", 300.0)
assert test_distance("abcde", "bcde", 25.0)
assert test_distance("abcdef", "bcdef", 20.0)
assert test_distance("abcdefg", "bcdefg", 100.0/6) #~16.667
assert test_distance("turlututu chapeau pointu", "turlututu chapeau pointu",
0.0)
assert test_distance("turlututu chapeau", "turlututu chapeau pointu", 610.0/17)
#35.9
assert test_distance("chapeau", "turlututu chapeau pointu", 1520.0/7) #217.1
assert test_distance("chapter I - the birth of the hero", "chapter I - the
birth of the hero", 0)
assert test_distance("the birth of the hero", "chapter I - the birth of the
hero", 840/21) # 40
        println()

#-----
def test_match(s1, s2, refret):
    b1 = symmetric_test(cls.match, s1, s2, refret)
    b2 = symmetric_test(cls.match, reverseString(s1), reverseString(s2), refret)
    return b1 and b2

assert test_match("abcdef", "abcdef", True)
assert test_match("abcdef", "bcdef", False) #200/5.0=40 100/5=20 0/5=0
assert test_match("abcdef ", "bcdef ", True) #200/5.0=40 100/6=16.7 0/5=0
assert test_match("abcdef other parts being equal", "bcdef other parts being
equal", True)
assert test_match("abcdefg", "bcdefg", True) #100/6=16.67
        println()

#-----
def test_similarity(s1, s2, refret):
    b1 = test(cls.similarity, s1, s2, refret)
    b2 = test(cls.similarity, s2, s1, refret)
    return b1 and b2

assert test_similarity("", "", 1.0)
assert test_similarity("", "b", 0.0)
assert test_similarity("a", "b", 0.0)

```

```

assert test_similarity("abc", "uvwxyz", 0.0)
assert test_similarity("a", "1", 0.0)
assert test_similarity("a", "a", 1.0)
assert test_similarity("1", "1", 1.0)
assert test_similarity(".", ".", 1.0)
assert test_similarity("abc", "abc", 1.0)
assert test_similarity("lbc", "abc", 1.0 - 1.0/3.0)
assert test_similarity("lbc", ".bc", 1.0 - 1.0/3.0)
o = float(cls.iCOSTOTHER)
a = float(cls.iCOSTALNUM)
assert test_similarity(";bc", ".bc", 1.0 - o/(o+a+a))
assert test_similarity("a;c", ".c", 1.0 - (a+o)/(a+o+a))

#-----
traceln("Done - %d tests - OK"%cnt)
selfTest = classmethod(selfTest)

#----- MAIN -----
if __name__ == "__main__":
    usage = "usage: %prog [options] <groundtruth-file> <submission-file>"
    parser = optparse.OptionParser(usage=usage, version="$Revision: 1.11 $")
    parser.add_option("-v", "--verbose", dest="bVerbose", action="store_true"
        , help="show a detailed error report per book")
    parser.add_option("--csv", dest="bCSV", action="store_true"
        , help="store the numerical data in a CSV file")
    parser.add_option("--doc", dest="doc", action="store_true"
        , help="Display the SW documentation and exit")
    parser.add_option("--selftest", dest="selftest", action="store_true"
        , help="Self tests and exit")
    (options, args) = parser.parse_args()

    if options.selftest:          #AUTOTEST
        InexTitle.selfTest()
    elif options.doc:            #SHOW DOC
        traceln(__doc__)
    else:                         #COMPARE SUBMISSIONS
        if len(args) != 2:
            parser.error("Incorrect number of arguments")

        traceln("Loading GROUNDTRUTH file: ", args[0])
        sbmgt = Submission(xml.dom.minidom.parse(args[0]), "groundtruth", False) #Loose
mode: e.g. we ignore entries without a page number
        #For GREYC results :-( :-)
        #TocEntry.DELTA = -1
        traceln("Loading submission file: ", args[1])
        sbm = Submission(xml.dom.minidom.parse(args[1]), "submission", True) #Strict
mode: e.g. missing page numbers are counted as an error

        ltAll, tSummary = Submission.compare(sbmgt, sbm, options.bVerbose)

        if options.bCSV:
            sCSVFile = os.path.normpath(time.strftime("InexLinkCompare_%Y%m%d_%H_%M_
%S.csv", time.localtime()))
            traceln("\n\t*** Storing CSV data in "+sCSVFile+" ***")
            f = open(sCSVFile, "wb")
            csvWriter = csv.writer(f)
            csvWriter.writerow(["csv date" , time.ctime()])
            for name, filename in [ ("groundtruth" , args[0]), ("submission" ,

```

```

args[1]) ]:
    o = os.stat(filename)
    csvWriter.writerow([name, filename])
    csvWriter.writerow(["size", o[stat.ST_SIZE]])
    csvWriter.writerow(["date", time.ctime(o[stat.ST_MTIME])])
    csvWriter.writerow(["SW", sys.argv[0]])
    csvWriter.writerow(["SW", "$Revision: 1.11 $"])
    lRowTitle = ["doc", "P-link", "R-link", "F-link", "Ttl-acc", "RelLvl-acc",
        "P~InexLink", "R~InexLink", "F~InexLink",
    "InexTtl-acc", "Lvl-acc",
        "OK-link", "ERR-link", "MISS-link", "PbRelLvl",
    "PbInexTtl", "PbLvl",
        "bookid"]
    csvWriter.writerow(lRowTitle)
    for (i, tPRFAA, tCount, bookid) in ltAll: csvWriter.writerow((i,
+tPRFAA+tCount+(bookid,))
    csvWriter.writerow(tSummary)
    csvWriter.writerow(lRowTitle)
    f.close()

```