text mining

#TradingSignals #SoftwareSelection #OnlineReports #Java #Groovy #CRF #MachineLearning #SupervisedLearning #SequenceLabelPrediction #Regex #TextMining #Tokenization #Make #AWK #PowerBI

We want to monitor the market sentiment on agri-commodity futures: bullish/bearish. We will be downloading the weekly COT report published by the US CFTC and extract a few specific values from that. As data solutions architects, we are asked to set up a process that can support such activity over time and without being broken by minor, unexpected changes in the input files.

The report, unfortunately, seems to have an old-style typographic layout. Since we don't have any vendor's format specification with us, we either need to infer it with a detailed analysis and a good confidence or set up a smart extractor.

In this page we evaluate strength and weakness of two different approaches. The core component of this process is going to be a lexer, which is any deterministic tool that, given a set of rules, splits the input text into strings of characters and tags each string (based on morphology and not semantics).

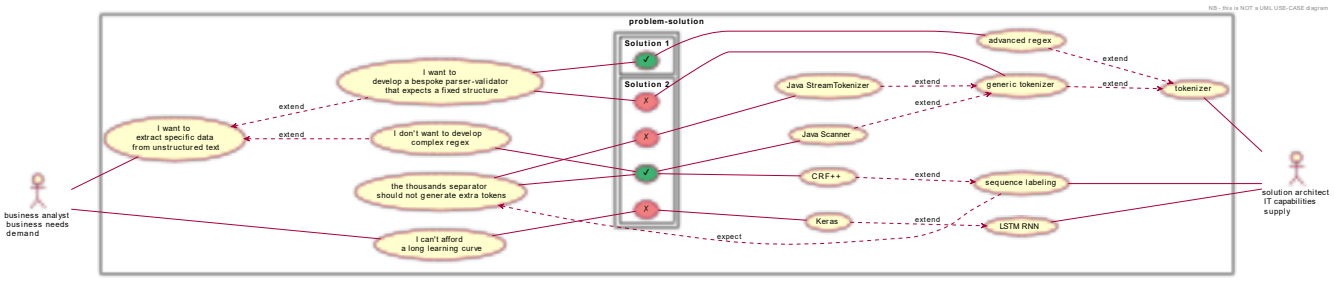
We will develop a lexer that is built ad hoc and may require both advanced knowledge of regex development and a thorough analysis of the text. Such lexers will likely generate a sequence of tokens that occur with highly regular patterns and are easier to tag depending on their relative positions.
On the other side, we will develop a more generic tokenizer, built with shallow knowledge of the input text file and with no regex development.
Such tokenizer, being more generic, will likely generate a sequence of tokens that may have a less regular patterns that may require Sequence labeling machine-learning algorithms like the CRF++.

Table of Content

building the tech stack
data source
goal
input
preliminary analysis
Lexer program
output
the machine-learning approach

building the tech stack

Below is the supply-demand-solution diagram that shows how the tools we have *shortlisted* can be combined to meet the requirements. Both reasons for *adoption* and reasons for *dismissal* are noted.



data source

weekly report COT by CFTC : https://www.cftc.gov/dea/options/ag_lof.htm - cached
Disaggregated Futures-and-Options -Combined / May 02, 2023 / Agriculture / Long Format

goal

to extract, for each product,

- **the product label** (line 1),
- the product code (line 1),
- **the report date** (line 2),
- the long and short *Positions* held by *Managed Money* traders, *All* crop years (line 11),
- the *Changes* of the two figures above w.r.t the previous week (line 16)

and to eventually show, for each product,

- **the current Bull/Bear Ratio**
- **the previous Bull/Bear Ratio**

input

```
1  WHEAT-SRW - CHICAGO BOARD OF TRADE
2  Disaggregated Commitments of Traders - Options and Futures Combined, May 02,
3  ------------------------------------------------------------------------------
4       :          :                                                 Reportable Po
5       :          :  Producer/Merchant/ :                                  :
6       :   Open   :   Processor/User    :          Swap Dealers          :
7       : Interest :   Long   :  Short   :   Long   :  Short   :Spreading :   L
8  ------------------------------------------------------------------------------
9       :          :(CONTRACTS OF 5,000 BUSHELS)
10      :          :    Positions
11 All  :   462,482:    78,394     51,330     71,087      4,461     14,141
12 Old  :   457,133:    77,419     49,598     70,363      4,321     13,838
13 Other:     5,349:       975      1,732        940        356         86
14      :          :
15      :          :    Changes in Commitments from:       April 25, 2023
16      :    32,903:     6,760     -4,310        158        918        559
17      :          :
18      :          :    Percent of Open Interest Represented by Each Category o
```

```
All  :     100.0:      17.0       11.1       15.4        1.0        3.1
Old  :     100.0:      16.9       10.8       15.4        0.9        3.0
Other:     100.0:      18.2       32.4       17.6        6.7        1.6
     :          :
     :          :    Number of Traders in Each Category
All  :       376:        80         69         24          5         21
Old  :       376:        80         68         24          5         21
Other:        75:        13         25         10          5          .
     :----------------------------------------------------------------------
     :             Percent of Open Interest Held by the Indicated Number of
     :                          By Gross Position                         By N
     :               4 or Less Traders     8 or Less Traders     4 or Less T
     :                 Long:     Short       Long      Short:      Long
     :----------------------------------------------------------------------
All  :                 12.0       12.6       20.4       21.8        8.8
Old  :                 12.1       12.6       20.5       21.9        8.9
Other:                 40.0       49.6       53.6       60.4       37.8

WHEAT-HRW - CHICAGO BOARD OF TRADE
Disaggregated Commitments of Traders - Options and Futures Combined, May 02,
------------------------------------------------------------------------------
     :          :                                                Reportable Po
     :          :  Producer/Merchant/ :                                  :
     :   Open   :   Processor/User    :          Swap Dealers           :
     : Interest :   Long   :  Short   :   Long   :  Short   :Spreading :   L


..............
```

preliminary analysis

given that we are analyzing a raw text file (not csv, not json, etc.),
gawk and a 200-char script can already provide valuable information on the file structure and uncover any high-level recurring pattern:

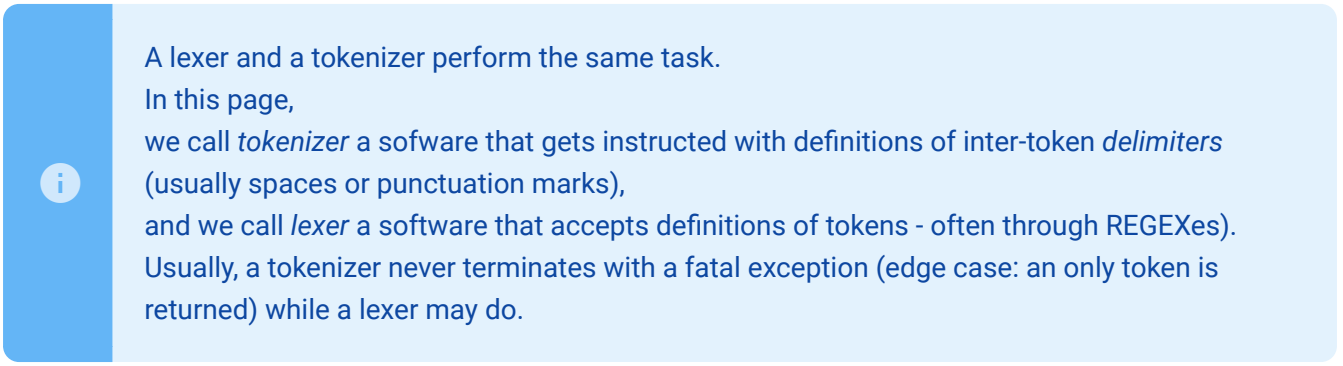
```
# gawk script
BEGIN {X=0; m=1; tab="\t"; print "SN" tab "NR" tab "NRgap" tab "input line"}

X==0 && /^[[:space:]]*$/ { X=1; next }
X==1 && /[[:space:]]+Code-[[:alnum:]]+$/ {
            print m++ tab NR tab NR-(NRprev?NRprev:NR) tab $0
            NRprev=NR; X=0; next
    }
```

the simple parsing above shows that there are 25 product sections and sections are regularly 36-line long.

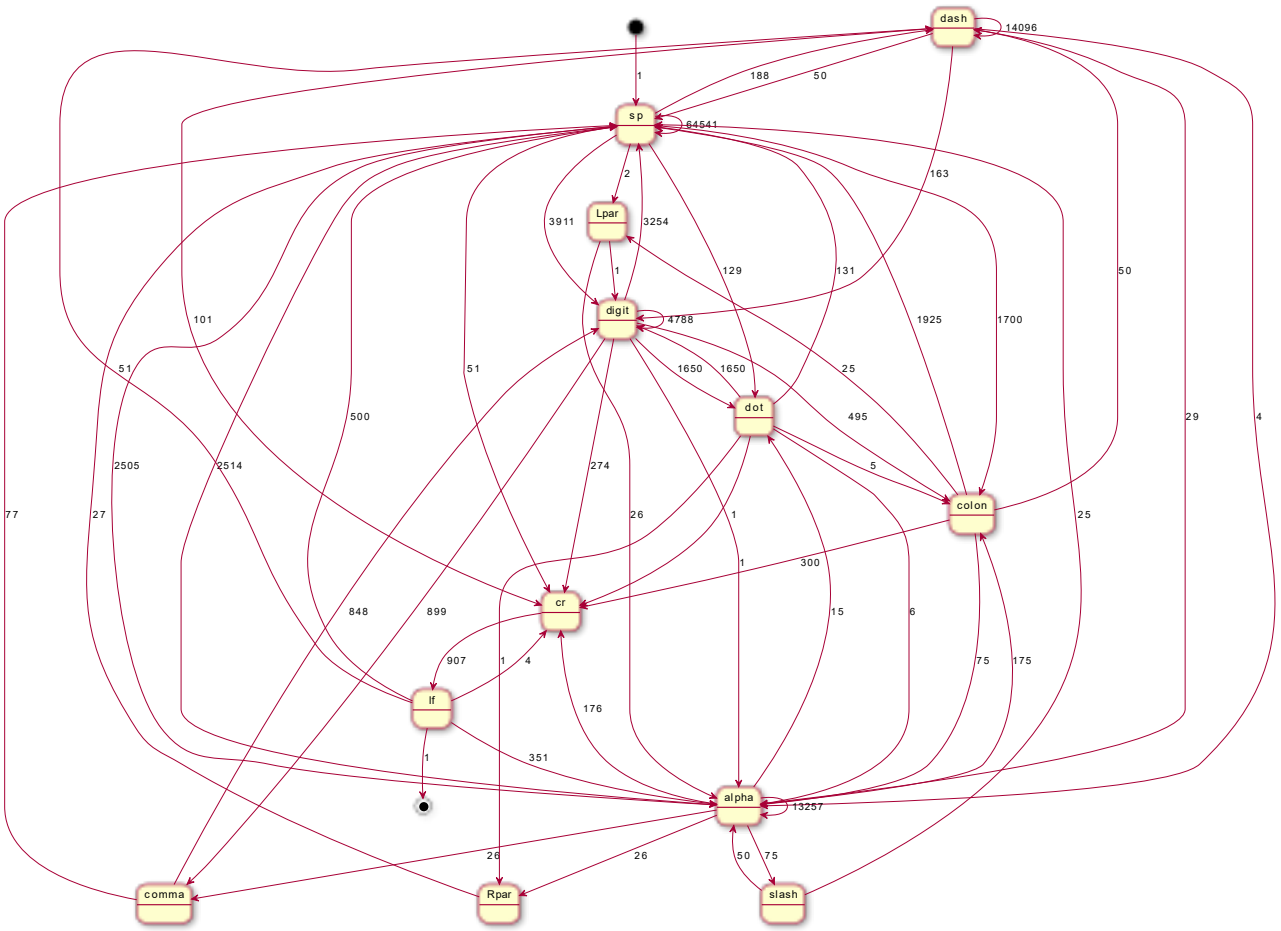
```
SN      NRin    NRgap   input line
```

```
1       2       0       WHEAT-SRW - CHICAGO BOARD OF TRADE
2       38      36      WHEAT-HRW - CHICAGO BOARD OF TRADE
3       74      36      WHEAT-HRSpring - MINNEAPOLIS GRAIN EXCHANGE
4       110     36      CORN - CHICAGO BOARD OF TRADE
5       146     36      CORN CONSECUTIVE CSO - CHICAGO BOARD OF TRADE
6       182     36      OATS - CHICAGO BOARD OF TRADE
7       218     36      ROUGH RICE - CHICAGO BOARD OF TRADE
...
22      758     36      FRZN CONCENTRATED ORANGE JUICE - ICE FUTURES U.S.
23      794     36      COCOA - ICE FUTURES U.S.
24      830     36      SUGAR NO. 11 - ICE FUTURES U.S.
25      866     36      COFFEE C - ICE FUTURES U.S.
```

Lexer program



A pure Lexer is a set of rules by which a sequence of characters is identified and labeled, independently of context. For example, given the input "380 Airbus A380" the Lexer and its rules may return: NUMBER(380) WORD(Airbus) WORD(A380).
A byte-pair sequence analysis based on utilities such as *GNU od* ("object dump") leads to the following finite-state-machine, that we will keep in mind while developing the ad-hoc, regex-based Lexer:



The Lexer developed in the Groovy language is as follows:

```
import java.util.regex.Pattern
import java.util.regex.Matcher

assert args.size()==1

String fileContents = new File(args[0]).getText('UTF-8')

List get_matches(String text,def re, String groupnames_csv) {
    List groupnames = groupnames_csv.tokenize(",")
    List ret = []
    Pattern pattern = Pattern.compile(re,Pattern.MULTILINE)
    Matcher matcher = pattern.matcher(text)
    while (matcher.find()) {
        groupnames.each { grp_nm ->
            String s = matcher.group(grp_nm)
            if(s) ret << [ a:matcher.start(), z:matcher.end(), sz:s.size(),
        }
    }
    return ret
}

```

```
skipFields=5
regexLineWithProduct = "^(?<prodlabel>.*?)(?:\\s+)(?<prodcode>Code-\\w+)\$"
regexLineWithReportDate = "^(?:Disaggregated.*?),\\s(?<reportdate>.*?)\\s+\\
regexLineWithPositionsAll =      "^(?:[\\s:]+Positions[\\s:]+\\n)All.*?:\\s*
regexLineWithPositionsChanges =  "^(?:[\\s:]+Changes in Commitments from[^\\
regex = [regexLineWithProduct,regexLineWithReportDate,regexLineWithPositions

int row=0
Closure stagger = { row += (it.grp== "prodlabel") ? 1:0; return [(it.grp):it
Closure merge = { Integer sn, List singleKeyMaps -> singleKeyMaps.inject(["r

List<Map> tokens = get_matches(fileContents,regex,"prodlabel,prodcode,report
List<Map> listWithLotNumber = tokens.collect stagger
Map<Integer,List<Map>> lots = listWithLotNumber.groupBy { it.row }
List<Map> out = lots.collect merge
println ""
delim="|"  // assuming no escaping is required
println "row,prodlabel,prodcode,reportdate,managedLong,managedShort,managedL
// enrich with signals:
out = out.collect { Map map ->
    managedShort = map.managedShort ? Float.parseFloat(map.managedShort.repl
    managedLong = map.managedLong ? Float.parseFloat(map.managedLong.replace
    managedLongChg = map.managedLongChg ? Float.parseFloat(map.managedLongCh
    managedShortChg = map.managedShortChg ? Float.parseFloat(map.managedShor
    BullBearRatio = (managedLong+managedShort==0) ? 0 : managedLong / (manag
    //  [Number of Bullish views/ (Bullish + Bearish views)] // x 100
    // computing prev-week values: if chg reads -3, prev-week value = this+3
    managedLong = managedLong - managedLongChg
    managedShort = managedShort - managedShortChg
    PrevBullBearRatio = (managedLong+managedShort==0) ? 0 : managedLong / (m
    BullBearRatioChange = "="
    if(BullBearRatio.round(2) > PrevBullBearRatio.round(2)) BullBearRatioCha
    if(BullBearRatio.round(2) < PrevBullBearRatio.round(2)) BullBearRatioCha
    return map + [BullBearRatio:BullBearRatio.round(2), PrevBullBearRatio:Pr
}
out.each { Map map -> println "row,prodlabel,prodcode,reportdate,managedLong
```

The COT report with the tokens identified and highlighted by the regex defined in the Lexer above is available at this link .
A few tokens (like "Other:") are inaccurate but the extraction of the data that is relevant to our objective is not affected.

output

below is the output of the groovy program, which contains both the regex logic and the computation of the Bull/Bear ratio and its one-week change.

```
row|prodlabel|prodcode|reportdate|managedLong|managedShort|managedLongChg|ma
```

```
1|WHEAT-SRW - CHICAGO BOARD OF TRADE|Code-001602|May 02, 2023|58,776|185,100
2|WHEAT-HRW - CHICAGO BOARD OF TRADE|Code-001612|May 02, 2023|38,707|44,171|
3|WHEAT-HRSpring - MINNEAPOLIS GRAIN EXCHANGE|Code-001626|May 02, 2023|4,844
4|CORN - CHICAGO BOARD OF TRADE|Code-002602|May 02, 2023|152,174|270,320|-38
5|CORN CONSECUTIVE CSO - CHICAGO BOARD OF TRADE|Code-00260B|May 02, 2023|0|7
6|OATS - CHICAGO BOARD OF TRADE|Code-004603|May 02, 2023|336|1,738|0|312|0.1
7|ROUGH RICE - CHICAGO BOARD OF TRADE|Code-039601|May 02, 2023|603|1,309|118
8|LEAN HOGS - CHICAGO MERCANTILE EXCHANGE|Code-054642|May 02, 2023|52,260|59
9|LIVE CATTLE - CHICAGO MERCANTILE EXCHANGE|Code-057642|May 02, 2023|124,263
10|FEEDER CATTLE - CHICAGO MERCANTILE EXCHANGE|Code-061641|May 02, 2023|21,8
11|BUTTER (CASH SETTLED) - CHICAGO MERCANTILE EXCHANGE|Code-050642|May 02, 2
12|MILK, Class III - CHICAGO MERCANTILE EXCHANGE|Code-052641|May 02, 2023|0|
13|NON FAT DRY MILK - CHICAGO MERCANTILE EXCHANGE|Code-052642|May 02, 2023|9
14|CME MILK IV - CHICAGO MERCANTILE EXCHANGE|Code-052644|May 02, 2023|318|0|
15|CHEESE (CASH-SETTLED) - CHICAGO MERCANTILE EXCHANGE|Code-063642|May 02, 2
16|SOYBEANS - CHICAGO BOARD OF TRADE|Code-005602|May 02, 2023|107,640|51,267
17|SOYBEAN OIL - CHICAGO BOARD OF TRADE|Code-007601|May 02, 2023|53,844|77,5
18|SOYBEAN MEAL - CHICAGO BOARD OF TRADE|Code-026603|May 02, 2023|83,479|22,
19|USD Malaysian Crude Palm Oil C - CHICAGO MERCANTILE EXCHANGE|Code-037021|
20|CANOLA - ICE FUTURES U.S.|Code-135731|May 02, 2023|18,867|81,022|-1,746|8
21|COTTON NO. 2 - ICE FUTURES U.S.|Code-033661|May 02, 2023|28,913|50,801|-5
22|FRZN CONCENTRATED ORANGE JUICE - ICE FUTURES U.S.|Code-040701|May 02, 202
23|COCOA - ICE FUTURES U.S.|Code-073732|May 02, 2023|111,952|62,461|-3,786|2
24|SUGAR NO. 11 - ICE FUTURES U.S.|Code-080732|May 02, 2023|260,090|52,290|-
25|COFFEE C - ICE FUTURES U.S.|Code-083731|May 02, 2023|45,710|13,704|-1,663
```

useful links about regex:

https://regex101.com/

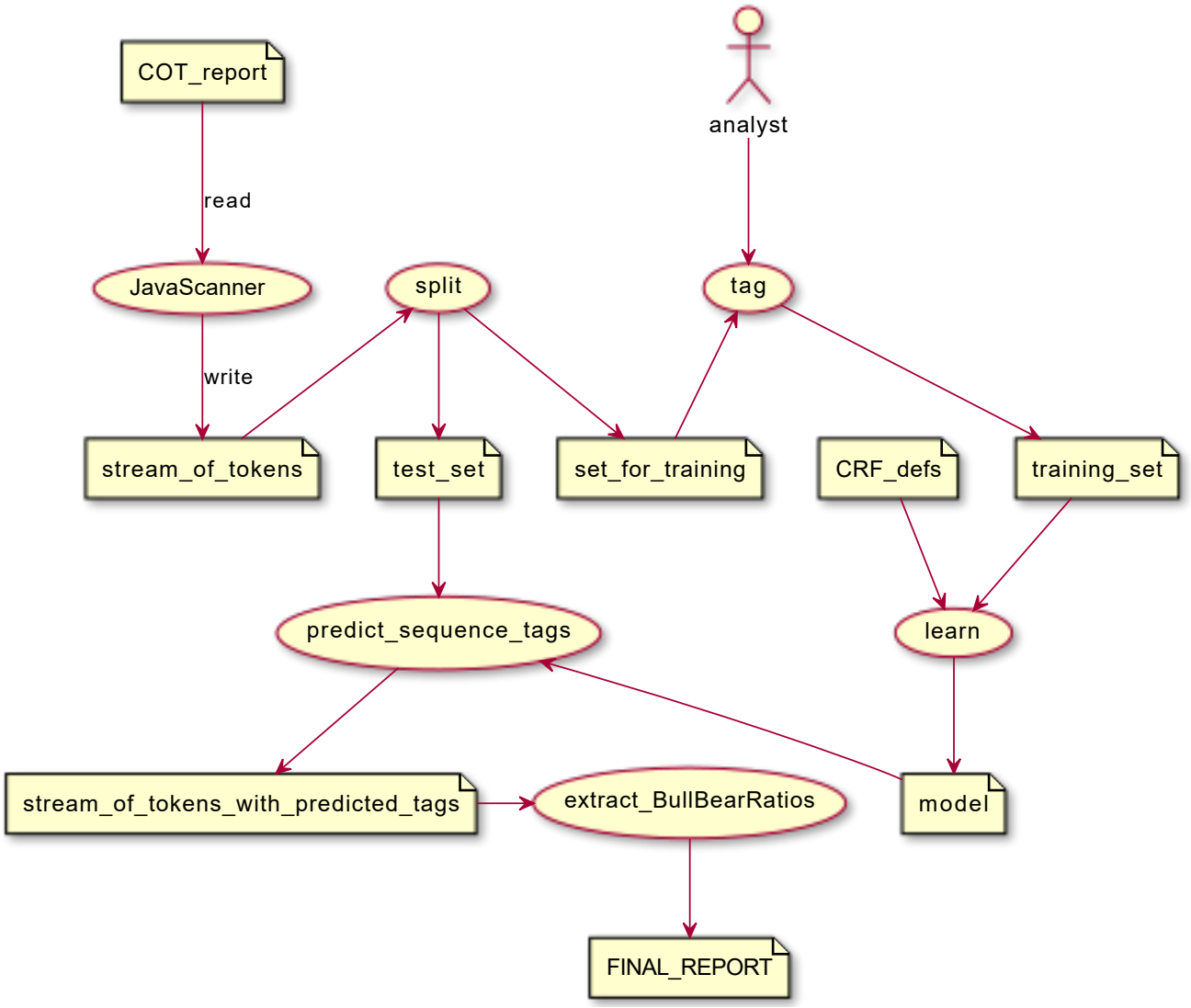
https://www.rexegg.com/regex-quickstart.html
https://www.regular-expressions.info/refquick.html
https://www.regular-expressions.info/refcapture.html
https://www.regular-expressions.info/refadv.html

http://www.rexegg.com/regex-lookarounds.html

> (?=foo) (Positive) Lookahead:
> Asserts that what immediately follows the current position in the string is foo
> (?<=foo) (Positive) Lookbehind:
> Asserts that what immediately precedes the current position in the string is foo
> (?!foo) Negative Lookahead:
> Asserts that what immediately follows the current position in the string is not foo
> (?<!foo) Negative Lookbehind:
> Asserts that what immediately precedes the current position in the string is not foo

Generic tokenizer + sequential pattern mining (ML)

We now look into an alternative approach in which, instead of leveraging on regex development skills, we will use a **generic tokenizer** that will generate a non-regular stream of tokens that we will be processed by a **sequential pattern mining** algorithm: we will manually tag a few tokens to generate a training set and we will run the CRF++ ML algorithm (**conditional random field**) on the test set to have all the tokens we need extracted from the stream.



Links:
http://www.philippe-fournier-viger.com/spmf/index.php?link=algorithms.php
https://taku910.github.io/crfpp/

Generic tokenizer (non regex-based): JavaScanner

```
// this is streamtknz.groovy

assert args.size()==1
String infilename= args[0]

class Xscanner {

    Scanner sc
```

```
    int inline_token_sn
    int lineno

    Xscanner(String line, int lineno_) {
        inline_token_sn=0
        lineno=lineno_
        sc= new Scanner(line)  //https://docs.oracle.com/javase/1.5.0/docs/a
    }
    Xscanner() {}

    String safeprint(String scnext) {
        int stkzttype = (int)scnext[0]
        return stkzttype>=32 && stkzttype<255 ? sprintf("%c",stkzttype) : "N
    }
    void xprint(List L) {
        if(L==null) { println(["lineno","LN_TK_SN","tkcode","tkval"].join("\
        assert L.size()==2
        println( ([lineno, inline_token_sn]+L).join("\t"))
    }
    void consume() {
        if(sc.hasNextDouble()) {
            xprint(["TT_NUM",  sc.nextDouble()])
        } else {
            String scnext = sc.next()
            if(scnext.size()>1) {
                xprint(["TT_WORD",  scnext])
            } else {
                int stkzttype = (int)scnext[0]
                xprint(["ASCII"+stkzttype,  safeprint(scnext)])
            }
        }
        inline_token_sn++
    }
}

new Xscanner().xprint()
int lineno
new File(infilename).eachLine("UTF-8") { line ->
    Xscanner p = new Xscanner(line,lineno++)
    while(p.sc.hasNext())
        p.consume()
    p.xprint(["TT_EOL",    "N/P"])
}
```

execution:

```
groovy streamtknz.groovy CFTC_COT_report.txt > CFTC_COT_stream_of_tokens.tsv
```

output with training labels added (excerpt):

inputLineSeqNo	LineTokenSeqNo	TokenCode	TokenText	TrainingLabel
0	0	TT_EOL	N/P	TT_EOL
1	0	TT_WORD	WHEAT-SRW	PRODUCT_NAME
1	1	ASCII45	-	ASCII45
1	2	TT_WORD	CHICAGO	TT_WORD
1	3	TT_WORD	BOARD	TT_WORD
1	4	TT_WORD	OF	TT_WORD
1	5	TT_WORD	TRADE	TT_WORD
1	6	TT_WORD	Code-001602	PRODUCT_CODE
1	7	TT_EOL	N/P	TT_EOL
...				
4	1	ASCII58	:	ASCII58
4	2	TT_WORD	Reportable	/Reportable
...				
6	7	ASCII58	:	ASCII58
6	8	TT_WORD	Managed	/Managed
6	9	TT_WORD	Money	TT_WORD
6	10	ASCII58	:	ASCII58
...				
7	11	ASCII58	:	ASCII58
7	12	TT_WORD	Long	LONG_HEADER

inputLineSeqNo	LineTokenSeqNo	TokenCode	TokenText	TrainingLabel
7	13	ASCII58	:	ASCII58
7	14	TT_WORD	Short	SHORT_HEADER
7	15	TT_WORD	:Spreading	TT_WORD
...				
11	7	TT_NUM	14141	TT_NUM
11	8	TT_NUM	58776	MMLONGVAL
11	9	TT_NUM	185100	MMSHORTVAL
11	10	TT_NUM	70898	TT_NUM
...				
35	8	TT_NUM	54.5	TT_NUM
35	9	TT_EOL	N/P	TT_EOL
36	0	TT_EOL	N/P	TT_EOL
...				
73	0	TT_WORD	WHEAT-HRSpring	PRODUCT_NAME
73	1	ASCII45	-	ASCII45
73	2	TT_WORD	MINNEAPOLIS	TT_WORD
73	3	TT_WORD	GRAIN	TT_WORD
73	4	TT_WORD	EXCHANGE	TT_WORD
73	5	TT_WORD	Code-001626	??? (expected prediction label: PRODUCT_CODE)
73	6	TT_EOL	N/P	TT_EOL

the Exchange of the first product (CBOT) was tokenized into 4 words;
the Exchange of the second product (MGE) was tokenized into 3 words;
the number of tokens generated for each commodity product may vary between 387 and 394;
hence the need of an ML algorithm trained to cope with such irregular sequences.

We keep our conditional random field definitions as naive as possible, based on the token type and its horizontal position (i.e. how many preceding tokens occur in the same line) and vertical offset (w.r.t. the beginning of the current "commodity product table"), as follows:

```
# CRF_TEMPLATE.txt
# Unigram
U_tokenclass_line_pos__prev2:%x[-2,1]/%x[-2,2]/%x[-2,3]
U_tokenclass_line_pos__prev1:%x[-1,1]/%x[-1,2]/%x[-1,3]
U_tokenclass_line_pos__curr_:%x[0,1]/%x[0,2]/%x[0,3]
U_tokenclass_line_pos__next1:%x[1,1]/%x[1,2]/%x[1,3]
U_tokenclass_line_pos__next2:%x[2,1]/%x[2,2]/%x[2,3]
```

We run this workflow:

```
# Makefile
# reminder: $@ = the target

all: CFTC_COT_stream_of_tokens.predictions-model1.extract.tsv CFTC_COT_strea

CFTC_COT_stream_of_tokens.tsv : streamtknz.groovy CFTC_COT_report.txt
    groovy streamtknz.groovy CFTC_COT_report.txt > $@

CRF_TEMPLATE = CRF_TEMPLATE.txt

model1 : CFTC_COT_stream_of_tokens.trainset.tsv  $(THE_TEMPLATE)
    @echo running LEARNER 1 :
    .\CRF\CRFpp-058\crf_learn -f 3 -c 4.0   $(CRF_TEMPLATE)    CFTC_COT_stre

model2 : CFTC_COT_stream_of_tokens.trainset.tsv  $(THE_TEMPLATE)
    @echo running LEARNER 2 :
    .\CRF\CRFpp-058\crf_learn -a MIRA -f 3  $(CRF_TEMPLATE)    CFTC_COT_stre

CFTC_COT_stream_of_tokens.predictions-model1.tsv : CFTC_COT_stream_of_tokens
    @echo running the PREDICTOR 1 :
    .\CRF\CRFpp-058\crf_test -m model1    CFTC_COT_stream_of_tokens.testset.

CFTC_COT_stream_of_tokens.predictions-model2.tsv : CFTC_COT_stream_of_tokens
    @echo running the PREDICTOR 2 :
    .\CRF\CRFpp-058\crf_test -m model2    CFTC_COT_stream_of_tokens.testset.
```

```
26
27  CFTC_COT_stream_of_tokens.predictions-model1.extract.tsv : parse_prediction.
28      @echo extracting the key PREDICTIONS - model-1:
29      awk -f parse_prediction.awk  CFTC_COT_stream_of_tokens.predictions-model
30
31  CFTC_COT_stream_of_tokens.predictions-model2.extract.tsv : parse_prediction.
32      @echo extracting the key PREDICTIONS - model-2:
33      awk -f parse_prediction.awk  CFTC_COT_stream_of_tokens.predictions-model
```

the final raw output, where we can see the Long and Short Open Interests held by the Managed Money for each commodity, is as follows:

```
1   1       PRODUCT_NAME    WHEAT-SRW       PRODUCT_CODE    Code-001602     MMLO
2   2       PRODUCT_NAME    WHEAT-HRW       PRODUCT_CODE    Code-001612     MMLO
3   3       PRODUCT_NAME    WHEAT-HRSpring  PRODUCT_CODE    Code-001626     MMLO
4   4       PRODUCT_NAME    CORN            PRODUCT_CODE    Code-002602     MMLO
5   5       PRODUCT_NAME    CORN            PRODUCT_CODE    Code-00260B     MMLO
6   6       PRODUCT_NAME    OATS            PRODUCT_CODE    Code-004603     MMLO
7   7       PRODUCT_NAME    ROUGH           PRODUCT_CODE    Code-039601     MMLO
8   8       PRODUCT_NAME    LEAN            PRODUCT_CODE    Code-054642     MMLO
9   9       PRODUCT_NAME    LIVE            PRODUCT_CODE    Code-057642     MMLO
10  10      PRODUCT_NAME    FEEDER          PRODUCT_CODE    Code-061641     MMLO
11  11      PRODUCT_NAME    BUTTER          PRODUCT_CODE    Code-050642     MMLO
12  12      PRODUCT_NAME    MILK,           PRODUCT_CODE    Code-052641     MMLO
13  13      PRODUCT_NAME    NON             PRODUCT_CODE    Code-052642     MMLO
14  14      PRODUCT_NAME    CME             PRODUCT_CODE    Code-052644     MMLO
15  15      PRODUCT_NAME    CHEESE          PRODUCT_CODE    Code-063642     MMLO
16  16      PRODUCT_NAME    SOYBEANS        PRODUCT_CODE    Code-005602     MMLO
17  17      PRODUCT_NAME    SOYBEAN         PRODUCT_CODE    Code-007601     MMLO
18  18      PRODUCT_NAME    SOYBEAN         PRODUCT_CODE    Code-026603     MMLO
19  19      PRODUCT_NAME    USD             PRODUCT_CODE    Code-037021     MMLO
20  20      PRODUCT_NAME    CANOLA          PRODUCT_CODE    Code-135731     MMLO
21  21      PRODUCT_NAME    COTTON          PRODUCT_CODE    Code-033661     MMLO
22  22      PRODUCT_NAME    FRZN            PRODUCT_CODE    Code-040701     MMLO
23  23      PRODUCT_NAME    COCOA           PRODUCT_CODE    Code-073732     MMLO
24  24      PRODUCT_NAME    SUGAR           PRODUCT_CODE    Code-080732     MMLO
25  25      PRODUCT_NAME    COFFEE          PRODUCT_CODE    Code-083731     MMLO
26  26      PRODUCT_NAME    ------------------------------------------------
```

the final output contains:
25 true positives,
1 false positive,
0 true negatives,
0 false negatives.

back to Portfolio

backlog 🔒