# Aneks

# Zbiorysekwencji:

## Zbiór pozytywny, pierwsza gramatyka testowa:

*aabbcccccccccccccccccc*

*aaaaaaabbbbbbbcccccccc*

*aaaaaaaaabbbbbbbbbcccc*

*aaabbbcccccccccccccccc*

*abcccccccccccccccccccc*

*aaaaaaaaaabbbbbbbbbbcc*

*aaaaaabbbbbbcccccccccc*

*aaaaaaaabbbbbbbbcccccc*

*aaaaabbbbbcccccccccccc*

*aaaaaaaabbbbbbbbccccc*

*aaaabbbbccccccccccccc*

*aaaaaabbbbbbccccccccc*

*aaaaaaaaaabbbbbbbbbbc*

*aaabbbccccccccccccccc*

*aaaaabbbbbccccccccccc*

*aaaaaaabbbbbbbccccccc*

*aabbccccccccccccccccc*

*abccccccccccccccccccc*

*aaaaaaaaabbbbbbbbbccc*

*aaaaaaaabbbbbbbbcccc*

*aaaaaaabbbbbbbcccccc*

*aabbcccccccccccccccc*

*abcccccccccccccccccc*

*aaaaaabbbbbbcccccccc*

*aaabbbcccccccccccccc*

*aaaabbbbcccccccccccc*

*aaaaaaaaabbbbbbbbbcc*

*aaaaabbbbbcccccccccc*

*aaaaabbbbbccccccccc*

*abccccccccccccccccc*

*aabbccccccccccccccc*

*aaaaaaaaabbbbbbbbbc*

*aaaabbbbccccccccccc*

*aaaaaaabbbbbbbccccc*

*aaabbbccccccccccccc*

*aaaaaabbbbbbccccccc*

*aaaaaaaabbbbbbbbccc*

*aaaaaaabbbbbbbcccc*

*abcccccccccccccccc*

*aaaaabbbbbcccccccc*

*aabbcccccccccccccc*

*aaaabbbbcccccccccc*

*aaaaaaaabbbbbbbbcc*

*aaabbbcccccccccccc*

*aaaaaabbbbbbcccccc*

*aaaabbbbccccccccc*

*aaaaaaabbbbbbbccc*

*aaabbbccccccccccc*

*aaaaaabbbbbbccccc*

*aaaaaaaabbbbbbbbc*

*abccccccccccccccc*

*aaaaabbbbbccccccc*

*aabbccccccccccccc*

*aaaaaabbbbbbcccc*

*aaabbbcccccccccc*

*aabbcccccccccccc*

*aaaaabbbbbcccccc*

*aaaabbbbcccccccc*

*abcccccccccccccc*

*aaaaaaabbbbbbbcc*

*aaabbbccccccccc*

*aaaaaabbbbbbccc*

*aabbccccccccccc*

*abccccccccccccc*

*aaaaabbbbbccccc*

*aaaaaaabbbbbbbc*

*aaaabbbbccccccc*

*aaabbbcccccccc*

*aaaaaabbbbbbcc*

*abcccccccccccc*

*aabbcccccccccc*

*aaaabbbbcccccc*

*aaaaabbbbbcccc*

*aabbccccccccc*

*aaaabbbbccccc*

*aaaaaabbbbbbc*

## Zbiór negatywny, pierwsza gramatyka testowa:

*babcbcabb*

*bcbabbbcaabbbc*

*bbbacaabab*

*cabccbc*

*abbbacabaccc*

*cbaabbbbaaabbcabbb*

*ccbcc*

*abcbabb*

*aacbcacbaaaaacccac*

*cacabbaaaccabc*

*acbacaaacbcb*

*bbcba*

*bcbca*

*bccba*

*cbbbaaaabccccccacbcb*

*bbcccbaaaaccbaabaa*

*aabccbcbabaacbaac*

*bbcaabbaabccbbbcac*

*ccbcacbcacaacabbcca*

*aacab*

*bccccbbbaaabca*

*bbacaacaacaaccb*

*abbbbaaaccbcbabacbbb*

*aabbccacbcbbaac*

*cabaaabcccc*

*caaaaacbcc*

*baaaacbbabcaaabbac*

*cacccbbabcaabcacccc*

*cbabcacca*

*ccccccbbaccbcbc*

*abaabbababc*

*ccbbaccbbc*

*bbacababccaabccaac*

*cbcacbbaaab*

*abbbcacacca*

*bbcabb*

*bcccbbabacaababa*

*caacbcbabcbabac*

*aacabaabcbbbbbaaaa*

*abcaabbccbb*

*bbaaabaacabaaac*

*ccbcbcbbacccaa*

*cbccabacb*

*accccacb*

*acabacaccacbcbabcabc*

*bccbbc*

*bbccbaabcccccc*

*cbcaacaacab*

*bcaaccbabaaccbbcab*

*caacbcaccaaac*

*abbbbaabcccbcbcababc*

*bbacbbaccaa*

*cbcabbabbb*

*acccabc*

*bcbcab*

*accccccaccca*

*caaccacabbaacabbbbc*

*cabcccbbbccaacba*

*bccbbcabbccacbac*

*cbbbaacbbbcc*

*ccabc*

*aabbbaaccacabbc*

*cbabc*

*ccaacbcccacbcaaaabca*

*bcaabaca*

*bbcabcaaacccabb*

*bbcbbbbccbacaaaacac*

*ccbbccaaa*

*aacacbabacac*

*acbaccccac*

*accabcaab*

*aaacbbcba*

*acbbcbbbbbbaabacbcc*

*bcbaabbbbacbc*

*aabbbaaccabccbbbcab*

*bbbbbaaacabaa*

*babaacbbc*

*baccaabc*

*bbbcaabacabcbcbabacc*

*acbacbaaaccbac*

*cbaaccbbbacbabbcb*

*bababac*

*ccacbac*

*bcccccbbaccaacbc*

*acbabacab*

*cabcc*

*bccacabcaab*

*cacabbaab*

*accbaccbbaab*

*accbabcabbcbb*

*bcabbcac*

*abcccbbbaabcbbaa*

*cacbaccbbcbbbbbba*

*caccbcabbcba*

*cbcac*

*aacaa*

*caccbaaaaaaac*

*cabbcaabbababb*

*aaabacbbcccacccacb*

*aabaaaaa*

## Zbiór pozytywny, druga gramatyka testowa:

*bccc*

*bcccc*

*acccc*

*bccccc*

*accccc*

*bcccccc*

*acccccc*

*bccccccc*

*accccccc*

*acccccccc*

## Zbiór negatywny, druga gramatyka testowa:

*acccaacaaaccaacab*

*ccabaca*

*bcbcbcacbbacbcbba*

*baaacbbcc*

*ccbcbabababcca*

*bcbbaabacbabbaaac*

*babcab*

*bbcabbb*

*babaa*

*babbbbccaacccacacb*

*bcbbabc*

*bacbacacacaccbb*

*baccaaabacba*

*bbaccacacccbacaccb*

*aacabcbbcbbabaacbcac*

*cbcbabcbcba*

*bcbbbccaccbbb*

*aaccccbbbca*

*abccacaccbbbccb*

*cccaacacbccbacac*

*ccabccbb*

*bccbcaacccccb*

*baaaabccaba*

*aaaccbc*

*aaaacbababcaa*

*bbbcacccc*

*accab*

*cbaaaaaaacaabccccc*

*bbaaacabccabacbca*

*acabacbbb*

*cabcaacbb*

*cbacabcacbac*

*bccbaabbbaabcb*

*baccacac*

*bbccbbacabcaaccaccb*

*aaaccbcb*

*ccbcaacac*

*caabaca*

*aacaccbcaaa*

*abbca*

*aaacccabcaacaaba*

*abbcbbacbacaaba*

*bbcabbcaa*

*bcbca*

*acbaccccacac*

*bbccbcbbca*

*bbcbaabbabaacabb*

*aacaaccbcaaabcac*

*cbcaccbbccbbcbcac*

*bbbbbaababbc*

*caabacbbbabacb*

*cbaaba*

*baaccabbaaabbacac*

*bccacaccabb*

*aaaaccabc*

*acababccaaaca*

*cabcaaaccc*

*cabac*

*acccaa*

*cacaabcbacabb*

*bbbacabb*

*bcbacaabca*

*acbcbbcccaacaccab*

*acabcbacacacb*

*acaccbcbbbc*

*aaaaacc*

*cabccabbbcbcccbbcaac*

*cbbbcaacccacbbaab*

*ccacbacbbbbbcbaaacaa*

*abcbc*

*aaabbbabc*

*cabbabcaacbcbbba*

*ccbaabcca*

*aaacaabab*

*bcbcbaabaccaa*

*acbbbb*

*caabaaabbcacbca*

*cccbabbba*

*abbcaccccbaabacacc*

*acbabccbabacc*

*baccccb*

*ccbabbcbccbcaabba*

*aacbccbaaacacaababbb*

*acbbabca*

*bbccbbaa*

*bcbabbc*

*acbaaccccaaabbbccc*

*babcbcbcbcaacc*

*acbaabaabaca*

*aabbcaaccaaacaa*

*ccbacbbbbbabacbcbccb*

*bbcbcbcbcbb*

*abaac*

*bbacbacbcbbcaabb*

*bccaaccccbbc*

*bbbcacbbcc*

*abaaababbabbab*

*baabbbbbac*

*acbaccc*

*abbcbba*

## Białka, próbka pozytywna:

VNQSFGNLTTTMSSRAFQGQM

DGHVFHNNKIGGRARVAQGDL

TRNTVGVVSAKNDSRIQVGTV

ASHTYDGVEVENNGKALIGNK

TTNYVNEVDTAETGRVNIGNT

SGQEIGKVITLDESRAMVGLP

TGQKITKADMSDGGKLLVGLI

LSHTYDGVQVDVSGKALLGNS

VRNYVREIQGEENAKVRLGND

TGQKFGAMRTDNESIAMQGIV

TVSRVDSVAARGKSAVHIGHQ

GRNSAKDIRTEKRARVQLGNV

TGTAVKHAEAFNKARQWIGNI

TNNLVETVSAKDQSQVQIGTV

MDQTIGDVSTSDGSNAFVGYR

ANNTVHGLDAKGSAKVQIGSR

TGHSFKNNKLLGEAKAIYGDV

INQLITDVQTERHSTAHVGVL

VKNYARNVATREEARVRVGNE

TTNSVETVVGKGESKVLIGNE

FYQEIHDVKATTGGKGIVGAV

GSHKFTGNTVTGEADAQYGDQ

GMNSAGNVKTEDLARFQLGNV

INQRIAGVRTDGGSRAVVGVY

TINSVKTVVATGESGIQIGNT

ATNKIGDLDAQGNSRVHVGDS

TAQQIGNVTTTDESKALVGMP

KNRSFDNVKITGDARVRFDDT

TGQKIKGVRATDDSTALAGFV

RGQRITDVEMSGKGKALVGRV

AAVAIDQLIAKDKARQMGGTI

NTNRVEDIEVKGDSGVHVGDT

GQNYAKNIQSKEKARVRVGNE

GRSSAKSISTEDDARVQVGSV

TTHKYSFTEASEDSIVIQGDY

TANRADSIAAKGKSVMHIGNQ

IDQDIRHVTADTRSVAVAGVV

KNHSYDGNEANNETRAVYGNI

GHNSAEFVNLEGSAKFLVGNV

TRNEAGSVTAKGSSTVHIGNR

ISQDISDVSADNRGFVIAGVA

RPVSVEKAEARGQSRMFTGNM

GSNSTGNVTLEDAARIHVGNV

FTQDISDVTADGHSIAAAGVF

PGHSFVENTASEKAKTHYGNK

TGQTIRDVKATDNSMALTGLV

TVNHVDEINTAEPSRVHIGNT

LKQQIGNTTTENGGKSQVGIP

TNQDIGNVTIAELGYGAVGIS

MGISVESLVTQDNARIRNGNF

INQDISDVTTDDKSFAVIGAA

RSHRYVNVVARNGAKAMMGDY

VDQSFGKLSAQNNSRAFQGQM

FSHVYEENKVSGFARVLYGDQ

ITGQNTARCISGKGRTNIGHT

VEQTFGKLTATKESKAFQGQM

TGQRFGDMLIEDRSMAMQGIV

TGQDIGNIRTSDDSRALVGLP

VEQNISQIHTDGHSISVAGVA

STHVYIGNEISGNACVKLGDE

QSQTIGDTDTGEGAESVVGIF

VTNVAENIKVGQEARAHVGNV

TKNSMRTVSATNQSRLQVGNV

IDQRLGDVSTTKKSVALVGVY

HLHTYDTVIVGESGKVLLGNE

GRNSAQEVRMEGSSRFQLGNQ

SSISVAHLNAKGNARVRNGNV

VNQRISDVRTDGGSTAFVGIF

GRNSAKDVRMGGHARVHILIR

RRNLYAGNTMTGEAKVLQGNV

GKNSARNVTTEDKVRFHVGNV

TTNTADEIDAQAKSRVHIGNK

IEQDISQIAADGRSIAVAGVV

ASPQYRNFFTGGNARIQAGDE

ASHKYDEIKVEESGKALVGNK

VKQKIGKVTTEKGSQAEVGIV

INQRIEEVRTEKGSHAIVGVF

TKNSVETISAKSETRVQVGNV

SNVQIGNLDVEEMARLQNGDV

TKQNIRDVKTTGNSIALAGLI

VSNTADSVAAKGRSAVQIGNQ

SHNFARDIKAADQADVQVGDQ

PPISLDHAIARDEAIQGVGNM

TINSAETVFARGRSRVQIGNT

SGQDISDIVTSDNSQALVGMP

RGQRITGVEISEDGKAFVGKS

TINHVDEVNTTESSKVNVGNT

SGHSFGTVFTSDTASIFAGNY

MNVEIDDVSVGPGSWSLVGVS

AQHRFENVQTSDQAKAFFGNQ

MAVTIERLIAKGRAVQLTGRM

GINRIYKAKVTETAKVKVGNE

SGQDIRNIITSANSRTYVGMP

TSNTADTVDARGMSRVHIGAS

KQQRIRNVKTSANSVAHVGIF

APHVYEQIILEDNGNIQIGNK

TSHRIHDQTVTDNARVQVGHT

VHQRIKDVTTHNNSAAAVGVF

AGHSYIRTQNKGKTRALMGDE

GMNYAGNIERDGEAKVHVGDS

SANTFDVLIAQDRARQMAGSI

TGQDISKVHAGQKSFAVAGLA

SGHTFTENLAMDNSRVRYGDE

TGDKIKGLTIDGKARVEFGKF

AGTTIKYAEAMEDSRQLFGQI

GEHTYDGMYTSGTARALYGNK

ASNTIDNASVTDDAKVRVGNE

SCLDIHDIKATNHGKGVVGMI

PGHSYDGNTASGKAKAQYGNR

ITQDFRNITATSGGKGMFGVI

TGHFFLNVTVRNQGRALNGDT

ASHTYNGLTVGEGAKALIGNQ

RGNIYHNNTVSGEAFVQFGNT

HAQNIQDVKAIHDSTALAGYI

RGNVYDGVIIEDNARVHNGNQ

AAQRISDVCATEESTTLAGKF

AGVVLRDIHMSSGGQVMAGFI

AGHRYGGNTVQGSSKVHYGNR

AGTMVKYQTTYDNAKLMTGNI

VRNSVKVINSSEDANVRLGND

GRNFTKDIKSEEMARVRVGNE

ASIAIASAQVKGNARVMNGDT

TINSAGTVMGRGEARIQIGNR

TGHTYKYLEASNEARMLAGDL

TINEIVDLEARGSSDVHVGNT

PGNVYSGIHISGETRVRNGTN

INQDISNVTADNRSFAAAGVI

AGIIVDVQNVGGSSRVREGDV

TGQKIKGIKASQNSSALAGFI

SGHQIGDQSTFDNARVQNGHT

SGRTYQDIDADGEANSHIGRR

TSNRVDHVNGGEASRVQIGNT

TNQGISDVTAEYRSLAVAGVI

PGSRYVENEATDYAKAVYGNQ

VRQDIHNVKATNHGKGVIGMA

GKNYAKEIRSEEMARVLVGNV

TSISIRSIHARGNSRVMNGNN

TGTTVKYAETLDQARQFIGNI

TNNQGGNAVVAEKARFMVGNV

SSNTIDQAKVAETAKVKVVNE

EGNTVRYAVALGEARQLVGVI

QGDVFQRMKFGGKSKSHVGDS

TSHEYKCIEVEKGGKALIGNK

GGNRYEAVTITDSAMVENGDY

QGMTISIQKTGGHSRVRNGDV

VQNFAARVKTSESGKFEIGNI

RGLSIGSIVIEENARVRDGNF

VKQNIGDVSVGKKSTGFVGIH

PGHSYGVTIITGGTKLIQGDS

ASSQYDSVEVEKGGKALIGEI

PGSLYEKNEASGDVTVHYGDA

ISGQNTVGYISGQGRTSVGDI

GKNHASQVGAHEQARVRIGNE

AFHIYDGILVDKTAKALVGNK

PETNVEHGQAKNESRQVIGNI

SKNTYEVVLAKDKANIVVGNM

TVNMAGDVKAKGKSSVHIGNQ

RKHAYVDTTNSDDARVLMGNR

FCHVYHGIQVDDEARTHLGNI

HRIKIGKVTQASNAKAVIGVH

## Białka, próbka negatywna:

YLRQEEEVVARYFLPAGMTPL

DGRILSSTPNVLATQTLENTS

YRLNGLMVDTRKLQSYIDHAL

TAKSKKFPSYSVSYRFYPNRW

AVKSRGVIVIKLNDKDFVTSA

RPNAQRFGISNYCQIYPPNAN

SFPTYLGRPWKEYSRTVVMQS

LFIAGHDTNLANLGGALELNW

QILLGEFEPAEIVLEELNENA

HDFVPARYEQPRQEDLLNRWT

EIPAVIYNSPYYGFATRADLF

LVFERWRRLSDNSQWIQVSLV

SSSSPGEPLEPTDQDLDVLLK

LEETFRIGRELDVPVVISHHK

LDYEDDPSGDAQANTNACLRF

NPVQSTITEETFQTLLVSRAL

KIMEPLFRQLAKCVSSPHFQV

HWTWELEQKLPDRTVVRGKKN

VLGATIYLIGFDGTTVITRAV

AAADVWYLLPITAKLMVETEA

DGMGVSTVTAARILKGQKKDK

RHQVYEALMQDYPIYSELAQI

LRGNHECASINRIYGFYDECK

PYGYTTQSIPDKTELNQVAKS

TAAVSERGAAEALTAIRRGLR

QAFWKAVTAEFLAMLIFVLLS

ANVQCDTATRTYSVQVHGSND

LDRKTRFIVHGFIDKGEDGWL

AGVVGHSYLFAYTPVALKSLK

VQRLVNREQDEEFIFALLNHA

RNCYFTLEGVHEVFQEEGLHT

FLDHSERYEASAEFTQVWRRL

NFCVAISSNEPRCTSCHAGYG

RDLEIEVVLFHPNYNINGKKE

FRVGLSPSSVRTLVEAGHTVF

TSPATVDRTHALMRKIGQSPV

NGAPGRSSSDKHLPMIHCLGG

PLPEEETNPKGSGWLYHSDAI

LGFHWTKVHPDHPGLGEVYVL

LNYVPVYVMLPLGVVNVDNVF

VPTHVEEPAFLPDPNDGSLYT

RTTEERHKKVVQLVLKKVYEA

GSVVAHSTFSAMKANTMSNYT

ITEGSVKGLTPSKGPAQASVN

KRQPDSYFSVLNAFIDRKDSY

RFLMQQPNMVVAPSKAVNAGG

VKRLLNGKCTSGPGYVQLRSG

SHMEEETRELQSLAAAVVPSA

ENVHLIGHSLGAHVVGEAGRR

LRAQEEFGQGEWSEWSPEAMG

PGERLDLAMLAAIRRVYGLLA

FISGDKVNVAGLVLAGSADFK

AAALIVGSDPVPEIEKPIFEM

IFERDVEVFLVPTGTAANALC

TGMPGPLAHPGLVCDVAIGQS

WSPEDIDTRFLLYTNENPNNY

IDAFHWYSRTDALALGRGLEK

GHRSVGGLRASIYNAVPYESC

GTALEPALQNVAIGVTDIPAL

VVDRIYHIDRGYERIEDKLRA

KSGVMLTGSHNPPDYNGFKIV

LAELGAHGVTFHDDDLIPFGS

GRTVPLQIFRTKDRGWGVKCP

VEDKVLAHTAKAEAEIACGRA

VKTGKCLKTLPAHSDPVSAVH

CKSEKEPLVRALINDRVVPLH

REFLHVDDMAAASIHVMELAH

EPRTVEKYSNYRVVDSIEKAD

QLVSTSTNDDPLVELEPVWAM

ATNVGAGSGAMLTSYAVTVMF

VFTNPSIVKVFHGAFMDIIWL

EEILRLKEERNAIILAHNYQL

GGRTWKVARLMGDKAPPGRAW

IPIDIFQTLFKWSNQSFEYSS

TLRLVDHLLAGADTLAGLADR

VVCQLKVKIYSSNSGPTRRED

VIDELDELYGDGRSVSFHALR

GQNDLEKMTSILEAVPQVKFI

SVELTVRRLDKMASTLTEIFG

LTSLERGDVLFIDEIHRLNKA

KGQQLSMIELFSKHRDELLGR

VAGVYEVFEKLDLDPEEVDWQ

VKQWSPMHKKWVKIKSGSAEI

QRVWVNPSDGSYDLDYNLELA

SFYYRHLIYDEILQLAPAGAF

ADLLHHVSQNITAETREQDPS

SRYLNDVNSLNEGKLVLTDSG

ESWPVVGQFSSVGSLGADESK

QQYWQTVVERLPEPLAEESLS

KMDKFNIPQYTPSPTEVEAEI

RNPEQLALAKSDPALIPRLVD

LGMRYAFIGPLETMHLNAEGM

TITRLKMEGIEGAKTVAINTD

LIAEELQAIPNVTLFQGPLLN

PQTKAIIPVHYAGAPADLDAI

PHNGTDFATPIGAPVYSTGDG

GQRQSVDTHFVKGTIAADKST

QKVLAFTNLPRGLKIFFGVQV

ATTGQNGVIQAEEFSKVADVS

IETKEKNKVKLATLQLIKEKY

# Aplikacja

## SGA.m

function [grammar, distanceParam, yParam, populationHistory, lethalMutations, time, bestSolutionHistory, accuracyParam, modelZero] = SGA (hyperparameters, teachingSentencesFile, grammarFile, positiveTestFile, negativeTestFile, modelZero, stopTime)

%Główna funkcja aplikacji odpowiadająca za proces uczenia

%

%argumenty wyjściowe:

% grammar - gramatyka (nieterminale, reguły strukturalne i leksykalne oraz

% ich powiązania, czyli indeksy reguł o tym samym poprzedniku)

%

% distanceParam - zawiera parametry (max, min, mean, med, std) odległości (euklidesowej)

% poszczególnych osobników względem siebie (miara różnorodności populacji)

%

% yParam - zawiera parametry (max, min, mean, med, std) ze średniego

% dopasowania wszystkich sekwencji z próbki uczącej dla danej populacji

%

% populationHistory - zrzuty populacji robione co 100 przebiegów programu

% (dodatkowo pierwsza populacjia)

%

% lethalMutations - liczba mutacji krytycznie niekorzystnych, czyli powodujących, że któryś osobnik nie jest wyprowadzić jednego ze z próbki uczącej (p(x)=0)

%

% time - czas trwania poszczególnych cykli nauki

%

% bestSolutionHistory - najlepszy osobnik z każdej populacji

%

% accuracyParam - parametry pokazujące jak dobrym klasyfikatorem jest dana

% gramatyka (pole pod krzywą ROC)

%

% modelZero - parametry analogiczne do zawartych w accuracyParam, z

% wykorzystaniem modelu zerowego

%

%argumenty wejściowe:

%

% hyperparameters - parametry wejściowe (liczba osobników, liczba cykli,

% prawdopodobieństwa mutacji i zajścia procesu crossing over, maksymalna skala mutacji)

%

%teachingSentencesFile - plik zawierający zdania tworzące zbiór uczący

%

%grammarFile - plik zawierający reguły gramatyki

%

%positiveTestFile - plik z próbkami pozytywnymi (do walidacji)

%

%negativeTestFile - plik z próbkami negatywnymi (do walidacji)

%

%modelZero - struktura przechowująca częstotliwość występowania

%poszczególnych nieterminali względem siebie

%

%stopTime - wektor przechowujący wartości czasu(w godzinach) po którym program

%dokonuje zapisów dotychczasowych wyników

tic

rng('shuffle')

delete(gcp('nocreate'))

parpool(20, 'IdleTimeout', 120)

%inicjalizacja

[teachingSentences] = loadSentences(teachingSentencesFile);

[grammar] = loadGrammar(grammarFile);

[population] = initPopulation (grammar, hyperparameters.n);

[yParam, distanceParam, populationHistory, bestSolutionHistory, time] = preallocate (grammar,hyperparameters); %subfunkcja

lethalMutations=0;

%ocena P0

[points] = firstQualityCheck (population, grammar, teachingSentences);

[distanceParam, yParam] = calculateAllParam (population,1, distanceParam, yParam, points);

%zapisanie pierwszej populacji do archiwum populacji

populationHistory(:,:,1)=population;

%zapisanie najlepszego osobnika pierwszej populacji do archiwum najlepszych

%osobników

bestIndex = find(points==max(points), 1); %Indeks najlepszego osobnika

bestSolutionHistory(:,1) = population(bestIndex,:);

time(1)=toc;

t=1; %cycles

while t<=hyperparameters.tk

tic

%pierwsza część nowej populacji (T1) jest tworzona na podstawie

%operacji genetycznych (krzyżowanie, mutacje)

[populationT1] = reproduction(population, points, hyperparameters.CrossingOverProb);

populationT1PreMutations = populationT1;

[populationT1] = mutations(populationT1,hyperparameters.mutationProb,hyperparameters.mutationScale);

[populationT1] = scaleRulesProb (grammar,populationT1);

[pointsT1,populationT1, lethalMutations] = qualityCheck(populationT1,populationT1PreMutations, grammar, teachingSentences, lethalMutations);

%druga część nowej populacji (T2) jest tworzona poprzez skopiowanie

%lepiej przystosowanej połowy z poprzedniej populacji

[pointsT2, populationT2] = keepBetterOld (population, points, hyperparameters.n);

%łączenie części T1 i T2

population=[populationT1; populationT2];

points=[pointsT1, pointsT2];

%wyznaczanie parametrów

[distanceParam, yParam] = calculateAllParam (population,t+1, distanceParam, yParam, points);

%zapisanie co 100 populacji

if mod(t,100) == 0

[~, populationToSave] = keepBetterOld (population, points, hyperparameters.n\*2); %dzięki zastosowaniu funkcji keepBetterOld z parametrem n\*2 otrzymujemy całą populację posortowaną względem punktów

populationHistory(:,:,(t/100)+1)=populationToSave;

end

bestIndex = find(points==max(points), 1); %Indeks najlepszego osobnika

bestSolutionHistory(:,t+1) = population(bestIndex,:);

time(t+1) = toc;

%zapis po upływie czasu z wektora stopTime (argument wejściowy)

if find((sum(time)+(time(t+1)\*20))>stopTime\*3600)

stopTime((sum(time)+(time(t+1)\*20))>stopTime\*3600)=inf;

elapsedTime=sum(time)/3600;

saveResultAfterStopTime(grammar, distanceParam, yParam, hyperparameters, populationHistory, lethalMutations, time, teachingSentencesFile, grammarFile, teachingSentences, bestSolutionHistory, modelZero, elapsedTime, t);

end

t=t+1;

end

%zapis wyników po procesie nauki, przed walidacją

saveResultAfterStopTime(grammar, distanceParam, yParam, hyperparameters, populationHistory, lethalMutations, time, teachingSentencesFile, grammarFile, teachingSentences, bestSolutionHistory, modelZero, elapsedTime, t-1);

[accuracyParam, modelZero] = testTeachingAccuracy (positiveTestFile, negativeTestFile, bestSolutionHistory, hyperparameters.tk, grammar, modelZero);

%zapis wyników po walidacji

saveResult(grammar, distanceParam, yParam, hyperparameters, populationHistory, lethalMutations, time, teachingSentencesFile, grammarFile, teachingSentences, bestSolutionHistory, accuracyParam, modelZero);

end

%subfunkcja odpowiadająca za prealokację wektorów wykorzystywanych w tej funkcji

function [yParam, distanceParam, populationHistory, bestSolutionHistory, time] = preallocate (grammar,hyperparameters)

tk=hyperparameters.tk;

n=hyperparameters.n;

time=zeros(tk+1,1);

yParam.min=zeros(tk+1,1);

yParam.max=zeros(tk+1,1);

yParam.mean=zeros(tk+1,1);

yParam.std=zeros(tk+1,1);

yParam.med=zeros(tk+1,1);

distanceParam.mean = zeros(tk+1,1);

distanceParam.med = zeros(tk+1,1);

distanceParam.std = zeros(tk+1,1);

distanceParam.max = zeros(tk+1,1);

distanceParam.min = zeros(tk+1,1);

populationHistory=zeros(n,length(grammar.rules.Lex)+length(grammar.rules.NonLex),(tk/100)+1);

bestSolutionHistory = zeros(length(grammar.rules.Lex)+length(grammar.rules.NonLex),tk+1);

end

%subfunkcja odpowiadająca za zapis końcowy

function [] = saveResult(grammar, distanceParam, yParam, hyperparameters, populationHistory, lethalMutations, time, teachingSentencesFile, grammarFile, teachingSentences, bestSolutionHistory, accuracyParam, modelZero)

if exist('results', 'file') ~= 7 %sprawdzanie czy istnieją odpowiednie foldery, jeżeli nie to tworzenie ich

mkdir('results');

end

if exist(strcat('results/', grammarFile), 'file') ~= 7

mkdir(strcat('results/',grammarFile));

end

tn=1; %liczba porządkowa oznaczająca numer testu o zadanych parametrach

saved=0;

while ~saved

filename=strcat('results/',grammarFile,'/Teaching1n',mat2str(hyperparameters.n),'mp',mat2str(hyperparameters.mutationProb),'ms',mat2str(hyperparameters.mutationScale),'COp',mat2str(hyperparameters.CrossingOverProb),'tk',mat2str(hyperparameters.tk),'testNr',mat2str(tn),'.mat');

if exist(filename, 'file') == 2

tn=tn+1;

else

save(filename,...

'grammar', 'distanceParam', 'yParam', 'hyperparameters', 'populationHistory', 'lethalMutations', 'time', 'teachingSentencesFile', 'grammarFile', 'teachingSentences', 'bestSolutionHistory', 'accuracyParam', 'modelZero');

saved=1;

end

end

end

%subfunkcja odpowiadająca za zapis po upływie sprawdzanego czasu

function [] = saveResultAfterStopTime(grammar, distanceParam, yParam, hyperparameters, populationHistory, lethalMutations, time, teachingSentencesFile, grammarFile, teachingSentences, bestSolutionHistory, modelZero, elapsedTime, t)

time=time(1:t+1);

yParam.min=yParam.min(1:t+1);

yParam.max=yParam.max(1:t+1);

yParam.mean=yParam.mean(1:t+1);

yParam.std=yParam.std(1:t+1);

yParam.med=yParam.med(1:t+1);

distanceParam.mean = distanceParam.mean(1:t+1);

distanceParam.med = distanceParam.med(1:t+1);

distanceParam.std = distanceParam.std(1:t+1);

distanceParam.max = distanceParam.max(1:t+1);

distanceParam.min = distanceParam.min(1:t+1);

bestSolutionHistory = bestSolutionHistory(:,1:t+1);

if exist('results', 'file') ~= 7 %sprawdzanie czy istnieją odpowiednie foldery, jeżeli nie to tworzenie ich

mkdir('results');

end

if exist(strcat('results/', grammarFile), 'file') ~= 7

mkdir(strcat('results/',grammarFile));

end

tn=1; %liczba porządkowa oznaczająca numer testu o zadanych parametrach

saved=0;

while ~saved

filename=strcat('results/',grammarFile,'/Teaching1n',mat2str(hyperparameters.n),'mp',mat2str(hyperparameters.mutationProb),'ms',mat2str(hyperparameters.mutationScale),'COp',mat2str(hyperparameters.CrossingOverProb),'time',mat2str(elapsedTime),'testNr',mat2str(tn),'.mat');

if exist(filename, 'file') == 2

tn=tn+1;

else

save(filename,...

'grammar', 'distanceParam', 'yParam', 'hyperparameters', 'populationHistory', 'lethalMutations', 'time', 'teachingSentencesFile', 'grammarFile', 'teachingSentences', 'bestSolutionHistory', 'modelZero', 'elapsedTime');

saved=1;

end

end

end

## ScaleRulesProb.m

function [population] = scaleRulesProb (grammar,population)

%funkcja odpowiadająca za skalowanie prawdopodobieństw reguł o tym samym

%poprzedniku w taki sposób, aby ich suma wynosiła 1

[n,~] =size(population);

for i=1:n

rulesProb = population(i,:);

for j=1:length(grammar.rules.connections)

idx=cell2mat(grammar.rules.connections(j));

sameSum=sum(rulesProb(idx));

rulesProb(idx)=rulesProb(idx)/sameSum;

end

population(i,:) = rulesProb;

end

end

## reproduction.m

function [populationT1] = reproduction(population,points, reproductionProb)

%Funkcja odpowiedzialna za przeprowadzanie procesu reprodukcji i

%krzyżowania

pointsScaled = zeros(1,length(points));

bestResult=max(points);

for i=1:length(points)

pointsScaled(i)=bestResult/points(i);

end

[n,m] = size(population);

populationT = zeros(n,m);

%Tworzenie populacji Tymczasowej T

pointsSum = sum(pointsScaled);

points2 = pointsScaled/pointsSum;

points3 = cumsum(points2);

for i = 1:n/2

x=rand;

found = find(points3>x, 1);

populationT(i,:) = population(found,:);

end

%Crossing Over

[n,m] = size(population);

populationT1 = zeros(n/2, m);

for i=1:n/4

x=rand;

if x<=reproductionProb

x = randi([1 m]);

populationT1(((i-1)\*2)+1,1:x) = populationT(((i-1)\*2)+2,1:x);

populationT1(((i-1)\*2)+1,x:m) = populationT(((i-1)\*2)+1,x:m);

populationT1(((i-1)\*2)+2,1:x) = populationT(((i-1)\*2)+2,1:x);

populationT1(((i-1)\*2)+2,x:m) = populationT(((i-1)\*2)+1,x:m);

else

populationT1(((i-1)\*2)+1,:) = populationT(((i-1)\*2)+1,:);

populationT1(((i-1)\*2)+2,:) = populationT(((i-1)\*2)+2,:);

end

end

end

## mutations.m

function [population] = mutations(population,mutationProb,mutationScale)

%Funkcja odpowiadająca za przeprowadzanie mutacji na danej populacji

%

%parametry wejściowe:

%

%population - populacja na której przeprowadza mutacje

%

%mutationProb – prawdopodobieństwo zajścia mutacji

%

%mutationScale – maksymalna skalę mutacji

%

%parametry wyjściowe:

%population - populacja po przeprowadzeniu mutacji

[n,m] = size(population);

for i=1:n

for j=1:m

x=rand;

if x<mutationProb

z=0.01+rand\*(mutationScale-0.01);

y = randi([1 2]);

if y==1

population(i,j)=population(i,j)+z;

else

if population(i,j)>z

population(i,j)=population(i,j)-z;

else

population(i,j)=0;

end

end

end

end

end

end

## keepBetterOld.m

function [pointsT2, populationT2] = keepBetterOld (population, points, n)

%Fucnkja odpowiedzialna za zachowanie lepiej przystosowanej połowy

%populacji

OldPopPoints=[points.', population]; %Old population with points

OldPopPoints2=sortrows(OldPopPoints, 1);

OldPopPoints2=flipud(OldPopPoints2);

pointsT2 = OldPopPoints2(1:n/2,1).';

populationT2 = OldPopPoints2(1:n/2,2:end);

end

## initPopulation.m

function [population] = initPopulation (grammar, n)

%Funkcja odpowiedzialna za wygenerowanie populacji początkowej

%

%Parametry wyjściowe:

%population - zestaw osobników składających się z prawdopodobieństw

%przypisanych do poszczególnych reguł

%

%n - liczba osobników w populacji

population=zeros(n,length(grammar.rules.Lex)+length(grammar.rules.NonLex));

for i=1:n

for j=1:length(grammar.rules.Lex)+length(grammar.rules.NonLex)

population(i,j)=rand;

end

end

[population] = scaleRulesProb (grammar,population);

end

## firstQualityCheck.m

function [points] = firstQualityCheck (population, grammar, sentences)

%Funkcja odpowiadająca za ocenę pierwsej populacji

[n,~] =size(population);

points=zeros(1,n);

parfor i = 1 : n

for j = 1 : length(sentences)

[prob] = CYK\_Probabilistic(grammar, sentences{j}, population(i,:))

points(i)=points(i)+log(prob)

end

end

points=points/length(sentences);

end

## CYK\_Probabilistic.m

function [prob] = CYK\_Probabilistic(grammar, input, rulesProb)

%Fucnkja odpowidająca za analizę składniową zdania

%

%Dane wejściowe:

%gramamr - struktura zawierająca sprawdzaną gramatykę (reguły, symbole

%terminalne)

%

%input - sprawdzane zdanie

%

%rulesProb - zestaw prawdopodobieństw przypisany do reguł sprawdzanej gramatyki

%Inicjalizacja

P=zeros(length(input),length(input),length(grammar.nonTerminals));

%Pierwszy obieg (dla reguł leksykalnych)

for i=1:length(input)

for r=1:length(grammar.rules.lex)

if strcmp(input(i),grammar.rules.lex{r,2})

P(1,i,grammar.rules.lex{r,1})=rulesProb(r+length(grammar.rules.nonLex));

end

end

end

n=length(input);

%Testowanie reguł strukturalnych

for i=2:n

for j=1:n-i+1

for k=1:i-1

for r=1:length(grammar.rules.nonLex)

%left hand side (poprzednik w sprawdzanej regule)

lhs = grammar.rules.nonLex{r,1};

%right hand side (pierwsza część następnika w sprawdzanej regule)

rhs1 = grammar.rules.nonLex{r,2};

%right hand side2 (druga część następnika w sprawdzanej regule)

rhs2 = grammar.rules.nonLex{r,3};

if P(k,j,rhs1) && P(i-k,j+k,rhs2)

prob\_splitting = rulesProb(r)\*P(k,j,rhs1)\*P(i-k,j+k,rhs2);

P(i,j,lhs) = P(i,j,lhs) + prob\_splitting;

end

end

end

end

end

prob = P(n,1,1);

end

## calculateAllParam.m

function [distanceParam, yParam] = calculateAllParam (population,t, distanceParam, yParam, points2)

%funkcja wyznaczająca parametry statystyczne dla danej populacji

[m,n] = size(population);

distance = zeros(1, (m\*(m-1))/2);

idx=1;

for i = 1 : m-1

for j = i+1 : m

singleDistance = 0;

for r = 1 : n

singleDistance = singleDistance + (population(i,r) - population(j,r)).^2;

end

distance(idx) = singleDistance.^(1/2);

idx=idx+1;

end

end

distanceParam.mean(t) = mean(distance);

distanceParam.med(t) = median(distance);

distanceParam.std(t) = std(distance);

distanceParam.max(t) = max(distance);

distanceParam.min(t) = min(distance);

yParam.min(t)=min(points2);

yParam.max(t)=max(points2);

yParam.mean(t)=mean(points2);

yParam.med(t)=median(points2);

yParam.std(t) = std(points2);

end

## testTeachingAccuracy.m

function [accuracyParam, modelZero] = testTeachingAccuracy (positiveTestFile, negativeTestFile, bestSolutionHistory, tk, grammar, modelZero)

%Funkcja odpowiadająca za przeprowadzanie analizy uzyskanych w wyniku nauki

%gramatyk pod kątem klasyfikacji sekwencji z próbki uczącej

load('proteinTestSentences.mat');

accuracyParam.condiction = [ones(1,length(positiveSentences)),zeros(1,length(negativeSentences))];

accuracyParam.testSentencesPoints = ones((tk/10)+1, length(accuracyParam.testSentences));

testSentences=accuracyParam.testSentences;

solution = bestSolutionHistory(:,1);

currentPoints=zeros(1,length(accuracyParam.testSentences));

parfor j = 1 : length(accuracyParam.testSentences)%Pierwszy osobnik (później jest co dziesiąty)

currentPoints(j) = CYK\_Probabilistic(grammar, testSentences{j}, solution);

end

accuracyParam.testSentencesPoints(1, :) = currentPoints;

currentPoints=zeros(1,length(accuracyParam.testSentences));

for i = 1 : tk/10

solution = bestSolutionHistory(:,i\*10);

parfor j = 1 : length(accuracyParam.testSentences)

currentPoints(j) = CYK\_Probabilistic(grammar, testSentences{j}, solution);

end

accuracyParam.testSentencesPoints(i+1, :) = currentPoints;

end

auc = zeros(1, (tk/10)+1);

for i = 1:(tk/10)+1

[~,~,~,auc(i)] = perfcurve(accuracyParam.condiction,accuracyParam.testSentencesPoints(i,:),1);

end

accuracyParam.aucHistory = auc;

[modelZero] = zeroModel (modelZero, accuracyParam);

end

## qualityCheck.m

function [points,population, lethalMutations] = qualityCheck (population,populationPreMutations, grammar, sentences, lethalMutations)

%Funkcja odpowiedzialna za wyznaczanie wartości przystosowania dla

%osobników sprawdzanej populacji i sprawdzanie, czy nie doszło do mutacji

%krytycznie niekorzystnej

[n,~] =size(population);

points=zeros(1,n);

parfor i = 1 : n

for j = 1 : length(sentences)

[prob] = CYK\_Probabilistic(grammar, sentences{j}, population(i,:))

if prob == 0

points(i)=0;

lethalMutations = lethalMutations + 1;

population(i,:) = populationPreMutations(i,:);

for k = 1 : length(sentences)

[prob] = CYK\_Probabilistic(grammar, sentences{k}, population(i,:));

points(i)=points(i)+log(prob);

end

break;

else

points(i)=points(i)+log(prob);

end

end

end

points=points/length(sentences);

end

## loadSentences.m

function [sentences] = loadSentences(fileName)

%funkcja wczytująca zdania z pliku tekstowego

data=readtable(fileName);

sentences={};

data=table2array(data);

for i=1:length(data)

oneSentence=strsplit(data{i}, ' ');

oneSentence= oneSentence(3:end);

sentences{i}=strcat(oneSentence{1:end});

end

end

## loadGrammar.m

function[grammar] = loadGrammar(fileName)

%fnkcja wczytująca gramatykę z pliku tekstowego zawierającego jej reguły

fileID = fopen(fileName);

grammarRow=textscan(fileID,'%s %s %s');

A=grammarRow{1};

B=grammarRow{2};

C=grammarRow{3};

grammar.nonTerminals=A;

grammar.nonTerminals=unique(grammar.nonTerminals,'stable');

grammar.rules.Lex={};

grammar.rules.NonLex={};

for i=1:length(A)

if ~isempty(C{i})

grammar.rules.NonLex(end+1,:)={A(i),B(i),C(i)};

else

grammar.rules.Lex(end+1,:)={A(i),B(i)};

end

end

%mapowanie reguł strukturalnych (zamiana symboli nieterminalnych na liczby)

for i=1:length(grammar.rules.NonLex)

for j=1:length(grammar.rules.NonLex(1,:))

grammar.rules.NonLex{i,j}=find(ismember(grammar.nonTerminals, grammar.rules.NonLex{i,j}));

end

end

%mapowanie reguł leksyklanych

for i=1:length(grammar.rules.Lex)

grammar.rules.Lex{i,1}=find(ismember(grammar.nonTerminals, grammar.rules.Lex{i,1}));

end

%wyszukiwanie reguł o tym samym poprzedniku

allRulesLeft=vertcat(grammar.rules.NonLex{:,1},grammar.rules.Lex{:,1});

grammar.rules.connections={};

for i=1:length(grammar.nonTerminals)

sameRules=[];

for r=1:length(allRulesLeft)

if allRulesLeft(r)==i

sameRules=[sameRules, r];

end

end

grammar.rules.connections{end+1}=sameRules;

end

end