ANEKS

SPIS TREŚCI

Aneks 1
1. Zbiory sekwencji: 2
1.1. Zbiór pozytywny, pierwszy język testowy: 2
1.2. Zbiór negatywny, pierwszy język testowy: 2
1.3. Zbiór pozytywny, drugi język testowy: 4
1.4. Zbiór negatywny, drugi język testowy: 4
1.5. Białka, próbka pozytywna: 5
1.6. Białka, próbka negatywna: 7
3. Tabele 9
3.1. Tabela 1 9
3.2. Tabela 2. 11
4. Aplikacja 13
4.1. SGA.m 13
4.2. ScaleRulesProb.m 17
4.3. reproduction.m 17
4.4. mutations.m 18
4.5. keepBetterOld.m 19
4.6. initPopulation.m 19
4.7. firstQualityCheck.m 19
4.8. CYK_Probabilistic.m 20
4.9. calculateAllParam.m 21
4.10. testTeachingAccuracy.m 21
4.11. qualityCheck.m 22
4.12. loadSentences.m 23
4.13. loadGrammar.m 23
4.14. zeroModel.m 24

1. ZBIORY SEKWENCJI:

1.1. Zbiór pozytywny, pierwszy język testowy:

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

1.2. Zbiór negatywny, pierwszy język testowy:

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
abaabbbababc
ccbbaccbbc
bbacababccaabccaac
cbcacbbaaab
abbbcacacca
bbcabb
bcccbbabacaababa
caacbcbabcbabac
aacabaabcbbbbbaaaa
abcaabbccbb
bbaaabaacabaaac
ccbcbcbbacccaa
cbccabacb
accccacb
acabacaccacbcbabcabc
bccbbc
bbccbaabccccccc
cbcaacaacab
bcaaccbabaaccbbcab
caacbcaccaaac
abbbbaabcccbcbcababc
bbacbbaccaa
cbcabbabbb
acccabc
bcbcab
acccccccaccca
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
cabbcaabbbabab
aaabacbbcccacccacb
aabaaaaa

1.3. Zbiór pozytywny, drugi język testowy:

bccc
bcccc
acccc
bccccc
accccc
bccccccc
acccccc
bcccccccc
accccccccc

1.4. Zbiór negatywny, drugi język testowy:

acccaacaaaccaacab
ccabaca
bcbcbcacbbacbcbba
baaacbbcc
ccbcbababababcca
bcbbaabacbabbaaac
babcab
bbcabbb
babaa
babbbbccaacccacacb
bcbbabc
bacbacacacaccbb
baccaaabacba
bbaccacacccbacaccb
aacabcbbcbbabaacbcac
cbcbabcbcba
bcbbbccaccbbb
aaccccbbbca
abccacaccbbbccb
cccaacacbccbacac
ccabccbb
bccbcaaccccb
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
abaaabababbbab
baabbbbbac
acbaccc
abbcbba

1.5. 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
HRIKIGKVTQASNAKAVIGVH
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

1.6. 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

3. TABELE

3.1. Tabela 1

Tabela 3.1 - zestawienie sprawdzanych parametrów podczas pierwszego eksperymentu obliczeniowego. nr zest. - numer zestawu hiperparametrów, $t_{effective}$ czas efektywnego uczenia, $t_{ROC\ 0,95}$ – czas po którym AUC ROC wynosiło 0,95, t_{spec} – czas po którym dochodziło do specjalizacji nieterminali leksykalnych, r 0% - liczba wyzerowanych reguł w cyklu w którym best(ln(Pśr)) wynosił 95% maksymalnej wartości.

nr zest.	nr	max (best(ln(Pśr)))	$t_{effective}$ [cykle]	$t_{effective}$ [min]	$t_{ROC\ 0,95}$ [cykle]	$t_{ROC\ 0,95}$ [min]	t_{spec} [cykle]	t_{spec} [min]	r 0%
1	1	-2,53	1681	12,78	220	2,31	842	7,16	149
	2	-2,40	1188	10,06	170	1,89	924	8,18	108
	3	-2,55	1201	9,33	570	4,85	819	6,76	120
2	1	-2,55	1273	10,13	120	1,38	344	3,31	125
	2	-2,40	1185	9,24	90	1,08	661	5,64	109
	3	-2,49	1660	11,96	40	0,66	998	7,71	125
3	1	-2,48	1215	9,54	110	1,32	601	5,28	126
	2	-2,49	1117	8,88	550	4,78	775	6,55	111
	3	-2,48	1436	11,35	40	0,68	948	7,99	126
4	1	-2,55	767	6,71	310	3,07	582	5,33	76
	2	-3,29	1050	8,68	80	1,03	753	6,55	106
	3	-3,57	1308	10,38	370	3,42	793	6,72	132
5	1	-2,95	2731	20,41	780	7,01	2869	21,23	184
	2	-2,53	2763	21,26	560	5,09	2162	17,75	190
	3	-2,40	2441	17,70	1110	9,03	1900	14,61	184
6	1	-3,89	2681	20,67	830	7,25	0	0,00	134
	2	-4,14	2320	18,43	110	1,26	0	0,00	119
	3	-4,28	2695	22,17	-	-	0	0,00	116
7	1	-2,55	567	4,81	70	0,93	179	1,92	122
	2	-3,88	888	7,06	140	1,55	326	3,14	138
	3	-2,55	1073	7,49	40	0,60	611	4,71	157
8	1	-2,55	740	9,65	290	4,26	395	5,72	78
	2	-2,48	1262	16,07	110	2,07	590	8,84	134
	3	-2,48	861	11,19	250	3,67	946	12,16	86
9	1	-3,71	1156	14,69	310	4,57	641	8,93	110
	2	-2,48	1458	16,59	270	3,62	887	10,86	126
	3	-2,40	1063	13,62	220	3,40	377	5,63	109
10	1	-3,55	760	10,33	80	1,44	576	8,21	86
	2	-2,40	1458	17,34	110	1,86	432	6,41	143
	3	-3,62	777	10,52	190	3,05	413	6,20	82
11	1	-2,48	1428	16,83	180	2,88	581	8,00	150

	2	-2,47	834	10,90	190	3,21	225	3,85	95
	3	-2,97	933	11,77	-	-	494	6,950403	78,00
12	1	-4,18	1533	18,45	220	3,10	1663	19,74	118
	2	-4,20	1803	22,44	370	5,43	2104	25,61	136
	3	-3,65	1686	21,30	360	5,23	1896	23,50	128
13	1	-4,21	2325	27,42	170	2,46	0	0,00	117
	2	-2,68	2559	32,39	420	6,18	2540	32,21	132
	3	-3,49	2790	33,95	360	5,34	0	0,00	157
14	1	-2,48	529	6,74	60	1,13	173	2,77	121
	2	-3,27	2356	24,58	150	2,40	282	4,25	174
	3	-3,65	321	4,20	70	1,19	251	3,47	85
15	1	-2,48	1391	28,35	110	3,05	322	8,24	174
	2	-3,39	761	17,46	40	1,22	416	10,48	76
	3	-2,55	867	21,03	220	6,01	804	19,69	93
16	1	-3,66	1483	28,99	200	4,92	373	8,83	153
	2	-3,48	913	21,51	120	3,36	371	9,93	75
	3	-2,55	817	19,06	100	2,81	503	12,72	74
17	1	-2,48	810	17,45	100	2,70	338	8,32	103
	2	-2,52	772	18,61	360	9,44	405	10,77	83
	3	-3,74	930	20,95	80	2,27	464	11,65	125
18	1	-2,40	870	18,87	640	14,32	250	6,67	88
	2	-3,81	534	12,66	60	1,72	299	7,83	69
	3	-2,48	882	17,82	50	1,33	315	7,26	102
19	1	-2,48	1944	40,54	530	13,94	1092	26,44	174
	2	-2,55	1417	30,36	130	3,53	784	19,45	147
	3	-2,55	2160	43,23	260	6,89	883	21,76	165
20	1	-2,49	1644	33,90	350	8,50	949	21,87	108
	2	-2,49	2585	49,84	1100	25,52	1701	36,77	170
	3	-2,52	2186	48,53	410	10,93	1136	28,76	153
21	1	-2,51	495	11,42	70	1,99	218	5,76	127
	2	-2,55	405	9,34	70	1,98	129	3,70	120
	3	-2,46	960	18,60	70	2,01	197	5,16	168

3.2. Tabela 2.

Tabela 3.2 - obliczone parametry statystyczne dla wartości sprawdzanych w Tabeli 1. Oznaczenia jak w Tabela 3.1.

nr zest.	liczba pozystywnych symulacji	-	max (best(ln(Pśr)))	$t_{ROC\ 0,95}$ [min]	$t_{effective}$ [min]	t_{spec} [min]	r 0%
		śrd	-2,49	10,72	3,02	7,37	125,67
1	3	std	0,07	1,48	1,31	0,60	17,21
		śrd	-2,48	10,44	1,05	5,55	119,67
2	3	std	0,06	1,13	0,30	1,80	7,54
		śrd	-2,48	9,92	2,26	6,61	121,00
3	3	std	0,00	1,04	1,80	1,11	7,07
		śrd	-2,55	6,71	3,07	5,33	76,00
4	1	std	-	-	-	-	-
		śrd	-2,46	19,48	7,07	16,18	187,00
5	2	std	0,06	1,78	1,97	1,57	3,00
		śrd	-	-	-	-	-
6	0	std	-	-	-	-	-
		śrd	-2,55	6,15	0,77	3,31	139,50
7	2	std	0,00	1,34	0,17	1,40	17,50
		śrd	-2,50	12,30	3,34	8,91	99,33
8	3	std	0,03	2,74	0,93	2,63	24,73
		śrd	-2,44	15,11	3,51	8,24	117,50
9	2	std	0,04	1,48	0,11	2,61	8,50
		śrd	-2,40	17,34	1,87	6,41	143,00
10	1	std	-	-	-	-	-
		śrd	-2,48	13,86	3,05	5,92	122,50
11	2	std	0,00	2,96	0,17	2,07	27,50
		śrd	-	-	-	-	-
12	0	std	-	-	-	-	-

		śr d	-2,68	32,39	6,18	32,21	132,00
13	1	std	-	-	-	-	-
		śr d	-2,48	6,74	1,14	2,77	121,00
14	1	std	0,00	0,00	0,00	0,00	0,00
		śr d	-2,51	24,69	4,54	13,96	133,50
15	2	std	0,04	3,66	1,48	5,72	40,50
		śr d	-2,55	19,06	2,82	12,72	74,00
16	1	std	-	-	-	-	-
		śr d	-2,50	18,03	6,07	9,55	93,00
17	2	std	0,02	0,58	3,37	1,22	10,00
		śr d	-2,44	18,34	7,83	6,97	95,00
18	2	std	0,04	0,52	6,50	0,29	7,00
		śr d	-2,53	29,58	8,12	22,55	162,00
19	3	std	0,03	5,54	4,34	2,91	11,22
		śr d	-2,50	44,09	14,99	29,13	143,67
20	3	std	0,01	7,23	7,52	6,09	26,16
		śr d	-2,51	13,12	2,00	4,87	138,33
21	3	std	0,04	3,97	0,01	0,87	21,17

4. APLIKACJA

4.1. 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.mutatio
nScale);
    [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.No
nLex),(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(hyperparameter
s.n),'mp',mat2str(hyperparameters.mutationProb),'ms',mat2str(hyperparameter
s.mutationScale),'COp',mat2str(hyperparameters.CrossingOverProb),'tk',mat2s
tr(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(hyperparameter
s.n),'mp',mat2str(hyperparameters.mutationProb),'ms',mat2str(hyperparameter
s.mutationScale),'COp',mat2str(hyperparameters.CrossingOverProb),'time',mat
2str(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
```

4.2. 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
```

4.3. 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
```

4.4. 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
```

4.5. keepBetterOld.m

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

4.6. 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
```

4.7. firstQualityCheck.m

```
function [points] = firstQualityCheck (population, grammar, sentences)
%Funkcja odpowiadająca za ocenę pierwszej 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
```

4.8. CYK_Probabilistic.m

```
function [prob] = CYK_Probabilistic(grammar, input, rulesProb)
%Funkcja odpowiadająca 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
```

4.9. 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
```

4.10. 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

positiveSentences = loadSentences(positiveTestFile);
negativeSentences = loadSentences(negativeTestFile);
accuracyParam.testSentences = [positiveSentences,negativeSentences];

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

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

4.11. 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
```

4.12. 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
```

4.13. loadGrammar.m

```
function[grammar] = loadGrammar(fileName)
%Funkcja 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
```

4.14. zeroModel.m

```
function [modelZero] = zeroModel (modelZero, accuracyParam)
%Funkcja odpowiadająca za przeprowadzanie analizy uzyskanych w wyniku nauki
%gramatyk pod kątem klasyfikacji sekwencji z próbki uczącej z
uwzględnieniem modelu zerowego

terminals=modelZero.terminals;
terminalsFreq = modelZero.terminalsFreq; %częstotliwość występowania
kolejnych terminali z tablicy terminals

sentences = accuracyParam.testSentences;

modelProb = ones(1,length(sentences));
for i=1:length(sentences)

    singleSentence = sentences{i};
    n=length(singleSentence);

    ps = fzero(@(x) n*x.^(n-1) - (n+1)*x.^n ,[0.1,1]); %prawdopodobieństwo
znaku (sign)
    pe = 1 - ps; %prawdopodobieństwo wyjścia (end)

    terminalsFreq = (terminalsFreq/sum(terminalsFreq))*ps;

    prob=pe;
    for j=1: length(terminals)
        numberOfCharOccurences =
length(strfind(singleSentence,terminals(j)));
        prob=prob*terminalsFreq(j).^numberOfCharOccurences;
    end

    modelProb(i) = prob;

end

modelZero.sentencesProb = modelProb;
```

```
 modelAUC = zeros(1,length(accuracyParam.aucHistory));
for i=1:length(accuracyParam.aucHistory);

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

end

modelZero.aucHistory=modelAUC;

end
```

5. WYNIKI EKSPERYMENTÓW OBLICZENIOWYCH

Wyniki wszystkich wykonanych w pracy eksperymentów obliczeniowych zostały załączone w formacie . mat na płycie CD.