

---

## Verbal Autopsy - SMO

---

Egileak:  
Aitor San José, Andoni Olabarria, Leire García

Irakasgia:  
Erabakiak Hartzeko Euskarri Sistemak

3. maila

31. taldea

2022.eko apirilaren 8

# Aurkibidea

<b>1 Proiekako atazak eta arduren banaketa</b>	<b>2</b>
<b>2 Esparru teorikoa</b>	<b>3</b>
2.1 Support Vector Machine . . . . .	3
2.2 Diseinua eta atazen banaketa . . . . .	5
2.2.1 Aurreprozesamendua . . . . .	5
2.2.2 Baseline . . . . .	6
2.2.3 Ekorketa . . . . .	7
2.2.4 Sailkapena . . . . .	9
<b>3 Esparru esperimentalak</b>	<b>10</b>
3.1 Datuak eta aurre-prozesamendua . . . . .	10
3.2 Emaitza esperimentalak eta emaitza finalen diskusioa . . . . .	12
3.3 Exekuzio adibidea . . . . .	14
<b>4 Ondorioak eta etorkizunerako lana</b>	<b>15</b>
4.1 Atazaren laburpena . . . . .	15
4.2 Sendotasunak eta ahuleziak . . . . .	15
4.3 Ondorioak . . . . .	15
<b>5 Bibliografia</b>	<b>17</b>

## 1 Proiektyko atazak eta arduren banaketa

Gure taldeari esleituriko ataza Verbal Autopsy izan da, Munduko Osasun Era-kundeak bultzatutako proiektu bat. Proiektu honen helburua hirugarren munduan hildakoentzako heriotza-kausa jakitea da, askotan autopsiarik egiten ez delako, eta hildakoentzako zenbaketa eta hauen heriotza-kausa gordetzen ez direlako. Verbal Autopsy-ak hildakoaren familiakoekin edo lagunekin elkarritzketa bat izatean datza, elkarritzketa hauek transkribatu egiten dira eta ondoren hildakoaren heriotza-kausa iragarri ahal izateko erabiltzen dira datuak.

Proietkuan lan egiteko SMO algoritmoareaz baliatuz, transkribatutako elkarritzke-ta batekin eta datu gutxi batzuekin (sexua, adina eta jatorrizko herrialdea) heriotza-kausa bat iragartzea, lortu nahi da.

Proietkua Java lengoainean egingo da Eclipse garapen tresna eta WEKA paketea rabiliz, eta lau zati nagusitan banatuko da:

1. Aurreprozesamendua
2. Baseline
3. Parametro ekorketa
4. Sailkapena

Lau zati nagusi hauek, taldekideon artean banatu dira. Nahiz eta, taldekide batek zati konkretu baten ardura izan, hiru kideen artean garatuko da ataza horri dagokion kodea edo lana. Ahala ere, erantzukizuna ataza horri dagokion arduradunak jasango du guztiz.

Taldekideon ardura banaketa:

1. Leire: Parametro ekorketa
2. Aitor: Sailkapena eta Baseline
3. Andoni: Aurreprozesamendua

## 2 Esparru teorikoa

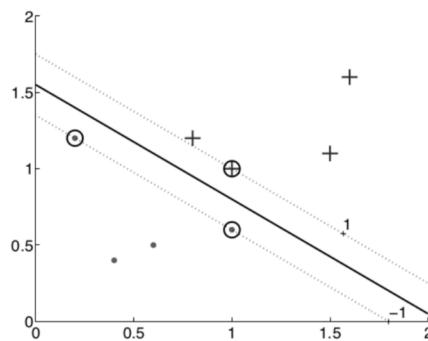
Hurrengo atalean, proiektuaren garapena ahalbidetu duten ikasketa prozesuan jorraturiko kontzeptuak eta teknikak laburbilduko dira. Diseinua azalduko da eta implementazioaren xehetasun nabarmenak aipatuko dira.

### 2.1 Support Vector Machine

SVM gainbegiratutako sailkapen-algoritmo bat da, bi klase bereizteko haien artean dagoen muga definitzen duena.

Entrenamenduaren ondoren, eredu linealaren parametroa, entrenamendu multzoaren azpimultzo baten arabera idatzi daiteke, hots, support vectors deritzenen arabera. Sailkapenean, mugatik hurbil dauden kasuekiko informazioa izateak ezaugatza erauzteko aukera ematen du: bi klaseren arteko mugatik hurbil dauden kasu zalantzazkoak edo okerrak dira. Haren zenbakiak orokortze-errorearen estimazio bat ematen digu, eta ondoren ikusiko dugunez, eredu-parametroa instantzia-multzo batean idatzi ahal izateak kernelizazioa ahalbidetzen du.

Kernel funtzioko espazioa definitzen du antzekotasunaren arabera eta honek ere, mugaren konplexutasuna ezartzen du erabilitako kernel motaren arabera. Kernel funtzioko baten kalitatea dagokion espazioaren bereizketaren araberakoa izango da. Eskabideak hiperplanoaren eskualde egokian egoteaz gain, urru egotea ere nahi dugu, hobeto orokortzeko. Hiperplanoaren inguruko instantzietara alde bakoitzean dagoen distantziari marjina deitzen zaio, eta hori maximizatu nahi dugu hobeto orokortzeko.



- 1. irudia:** Bi klaseko problema , non instantziak + eta · zeinuekin adierazi diren, lerro lodia muga adierazten duen eta lerro etenak marjinak zehazten dituzten. Zirkuluko kasuak support vectors dira[1].

Support vectors ez diren kasuek ez dute informazio gehigarririk ematen, beraz, horietako edozein azpimultzo ezabatzen bada ere, oraindik ere soluzio bera izango genuke. Ikuspegi horretatik, SVM algoritmoa hurbileneko nearest neighbor algoritmoarekin konparatu daiteke, auzoko instantziek soilik biltzen baitute klase-bereizlea. Mugatik hurbil dauden instantziez bakarrik arduratzentz da SVM, eta baztertu egiten ditu bestea [1].

Hala ere, SVMak zenbait arazo ditu, eskala handiko datuetarako entrenamendu abiadura motela eta konplexutasuna barne. VA ataza ebazteko, beste algoritmo bat, SMO, erabiltzea erabaki da sailkapenerako. SMO algoritmo berri bat da SVMren entrenamendurako, programazio koadratikoaren (QP) optimizazio-problema handi bat hausten duena, QPko problema posible txikiagoetan eta horrela laburtu egiten da denbora konputazionala [2].

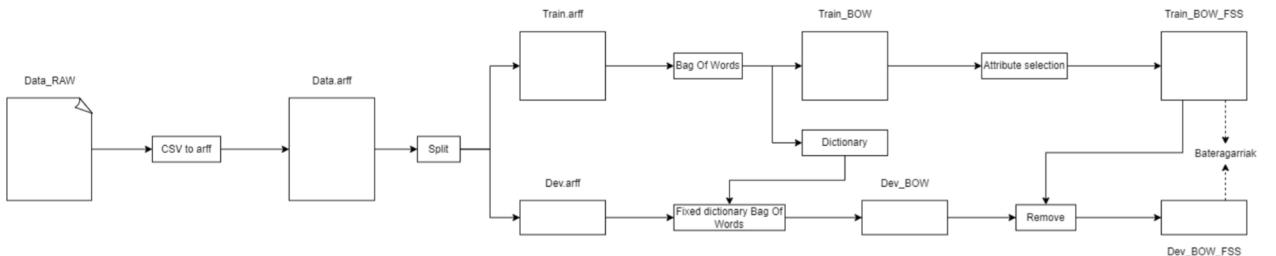
## 2.2 Diseinua eta atazen banaketa

### 2.2.1 Aurreprozesamendua

Lehenengo atal honen arduradun nagusia Andoni izan da eta batez ere Aitorren laguntzarekin garatu da. Aurreprozesamenduaren diseinuaren diagraman (2. irudian) ageri denez, atal honen fluxuan data era gordinean sartzen da programan. Lehenik eta behin .csv fitxategia .arff formatura bihurtzen da, jarraian lortutako dataset-a beste bi fitxategietan banatzen da train.arff eta dev.arff direnak. Behin bi dataset-ak banatuta daudela, bi aurreprozesamendu desberdin burutzen dira:

Lehenengo aurreprozesamendua train.arff fitxategiari egiten zaio, honi Bag Of Words eta Attribute Selection filtroak pasatzen zaizkio eta train\_BOW\_FSS.arff bilakatzen da.

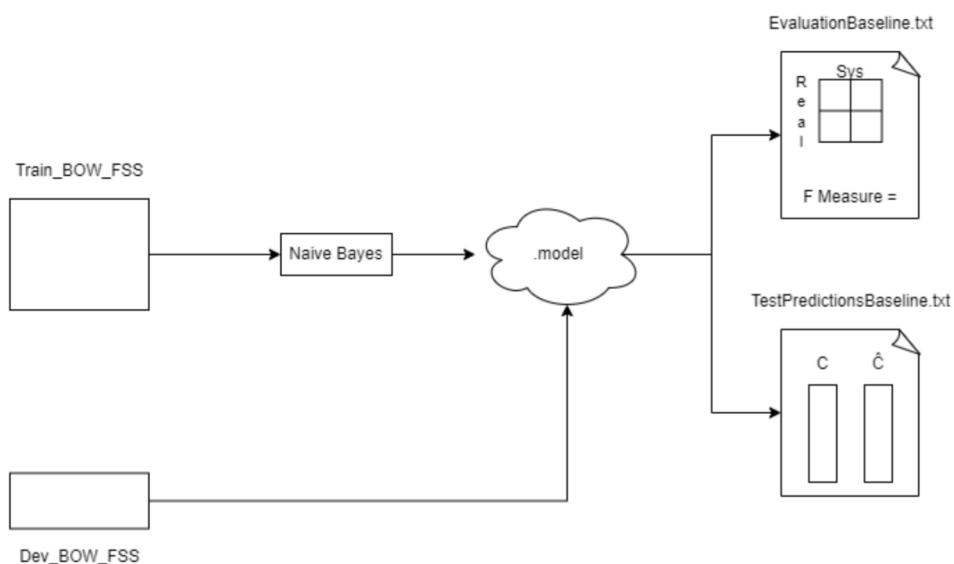
Aldiz, bigarren aurreprozesamendua dev.arff-ri egiten zaio, beharrezkoa da dev eta train headerrak bateragarriak izatea, horregatik train fitxategiari BOW aplikatu ostean lortutako dictionary fitxategi batean gordetzen da, eta hau erabiltzen da dev.arff-ri Fixed Dictionari BOW aplikatu eta devBOW.arff eskuratzeko; jarraian, train\_BOW\_FSS.arff-ren headerra izanda, remove filtroa aplikatuko zaio dev-ri horrela dev\_BOW\_FSS.arff lortzen.



**2. irudia:** Aurreprozesamenduaren diseinua.

### 2.2.2 Baseline

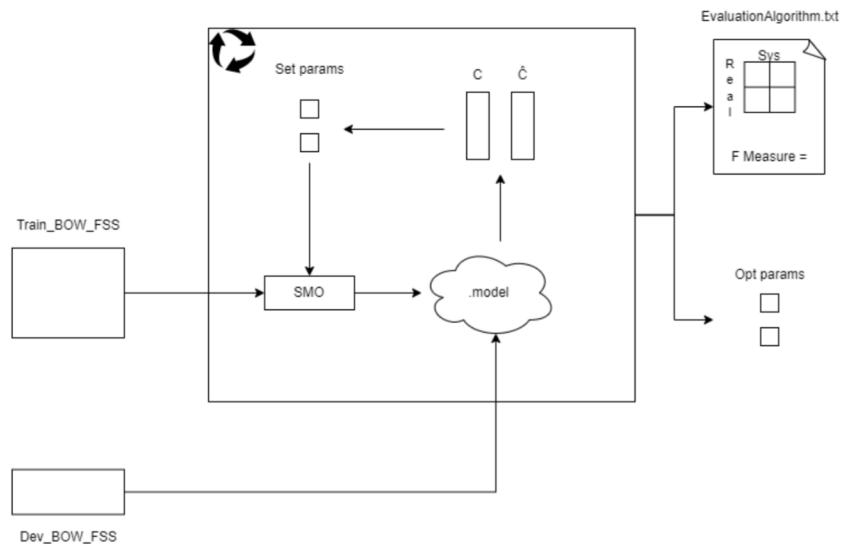
Baseline atalaren arduraduna Aitor izan da eta Leireren aholkuekin burutu da. Atza horretan aurreprozesamendutik pasa diren train\_BOW\_FSS.arff eta dev\_BOW\_FSS.arff sartzen zaizkio eta Naive Bayes sailkatzailea ebaluatzetan da adibidez, hold-out ebaluazio eskema erabiliz, lehenengo train fitxategiarekin entrenatzetan da eta dev erabiliz kalitatearen ebaluazioa eta iragarpeneak lortzen dira (3. irudia).



**3. irudia:** Baseline-aren diseinua.

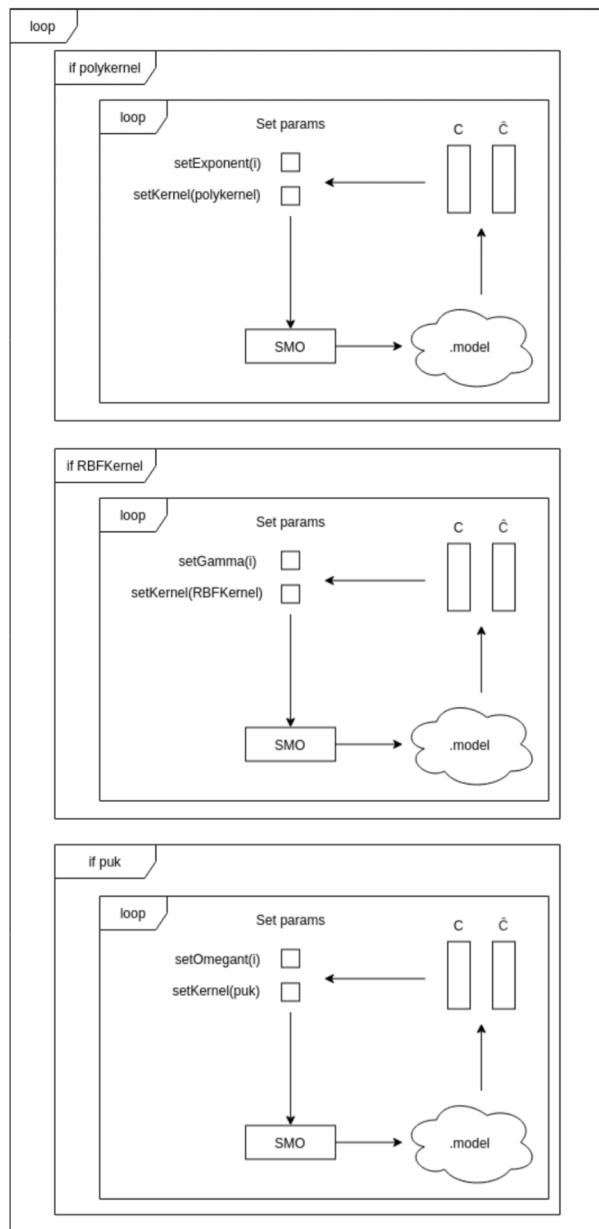
### 2.2.3 Ekorketa

Ekorketaren arduraduna Leire izan da eta Aitorren laguntzarekin implementatu da. Programa honi aurreprozesamendutik pasa diren train\_BOW\_FSS.arff eta dev\_BOW\_FSS.arff sartzen zaizkio eta iterazio bakoitzean SMO sailkatzailea era-biliz bere barnean behar dituen parametroak iteratzen dira klase minoritarioaren f-measure hoherena lortzen duten parametroak lortzeko (4. irudia).



**4. irudia:** Parametro ekorketaren diseinua.

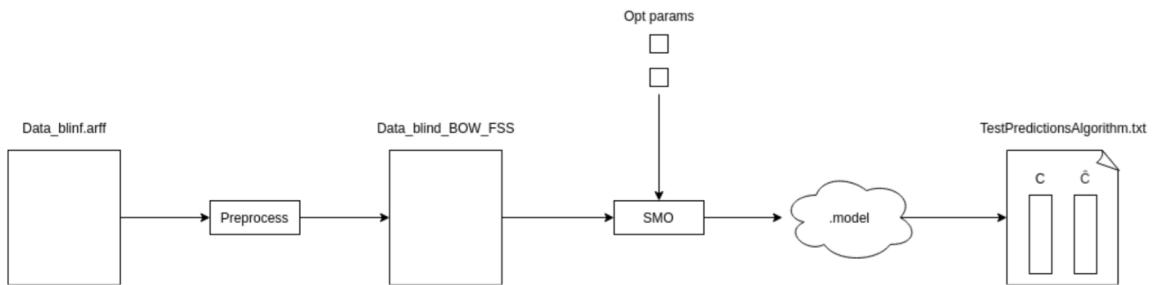
5. irudian ekorketaren iterazio bakoitzean zer burutzen den azaltzen da xehetasun handiagoarekin. 3 Kernel iteratzen dira: polykernel, RBFkernel eta puk; Kernel bakoitzak bere parametroak behar ditu, hurrenez hurren, exponentea, gamma eta omega. Parametro hauek ere iteratuko dira balio tarte jakin batean. Behin Kernel bat eta bere beharrezko parametroa definituta, modeloa eraikiko da eta estimazioak egingo dira, f-measure hoherena duen kernela eta bere parametroa itzuliz.



**5. irudia:** Parametro ekorketaren diseinua xehetasunenaz.

### 2.2.4 Sailkapena

Sailkapen atazaren arduradun nagusia Aitor izan da. Ataza honetarako oso garantzitsua da aurreprozesamendu atalean dev\_BOW\_FSS.arff train\_BOW\_FSS.arff-rekin bateragarria bihurtzeko egin den aurrprozesamendu bera egitea data盲\_ blind.arff-tik data\_ blind\_BOW\_FSS.arff bilakatzeko; eorketa egin ostean parametro optimoe-kin eraikitako eta data sorta guztiarekin entrenatutako modeloarekin bateragarria izan daitela bermatzeko. Amaitzeako pedikzioak testPredictionsAlgoritm.txt fitxate- gian modelo optimoarekin egindako iragarpeneak gordeko dira.(6. irudia).



**6. irudia:** Sailkapenaren diseinua.

### 3 Esparru esperimentalak

Atal honetan, proiektuaren zati praktikoa azaldu egingo da. Lehenik, eredu optimoa lortzeko eta hau testeatzeko balio izan duten datu-sortei buruz hitz egingo da. Geroago, datu-sorta hauei egin beharreko aldaketak jorratuko dira. Ondoren, parametro ekorketak eta baselineak eta algoritmoak izango emaitzak ikusi eta ekorketa merezi izan duen ala ez eztabaideatuko da. Azkenik, egindako programaren exekuzioaren adibide bat emango da.

#### 3.1 Datuak eta aurre-prozesamendua

Proiektua egiteko bi datu sorta desberdin eman zaizkigu, alde batetik eredu optimoa lortzeko train.csv artxiboa eta eredu ebaluatzenko testblind.csv artxiboa. Datu sortekin hainbat arazo egon dira .csv -tik .arff -ra pasatzeko hainbat karaktere berrezik eta komatxoek erroreak ematen zutelako eta hauek kentzeko **replace** funtzioa erabli egin da.

Behin data.arff izanda, hau bitan banatu egin da (train eta dev) parametro ekorketa egiteko. Hau **StratifiedRemoveFolds** filtroarekin egin da, filtro honetan zenbat fold erabiliko diren eta zein fold izango den muga adierazten da. Filtroak adierazitako fold kopurutan banatzen du datu-sorta eta instantzien zer portzentaia egongo den zati bakoitzean klasearen maiztasuna mantentzen dela bermatzen duen aldi berean. Filtro hau birritan aplikatuz 70/30 zatitan banatu da data.arff datu-sorta train.arff eta dev.arff sortuz, hurrenez hurren.

Train datu-sortari, ondoren, **StringToWordVector** aplikatuko zaio datu-sortaren atributu bati, elkarritzketaren transkripzioari, textu batetik atributu sorta batera bihurtzearen. Hala ere, hau egin baino lehen **NominalToString** filtroa aplikatu behar izan zaio atributuari nominal klasekoa zelako eta **StringToWordVector**-en izenak dioen bezala atributua string klasekoa izan behar duelako filtroa aplikatzearen. Horretaz aparte, hainbat atributuen izena aldatu behar izan da **RenameAttribute** filtroaren bitartez ez zutelako izen esanguratsurik. STWV filtroa aplikatu ostean trainBOW.arff fitxategia sortu da eta sortutako atributuen hiztegia gorde da.

Aurreko atalean sortutako trainBOW.arff-k 7000 atributu inguru ditu eta asko direnez, *InfoGain* gehien dituzten 2000 atributuekin bakarrik geratuko gara **AttributeSelection** filtroa erabiliz. Hau egikaritzeko lehenik InfoGainAttributeEval eta Ranker bat sortuko dira eta Rankerrean setNumToSelect 2000-ra hasieratuko da. Hau egin ostean Rankerra eta InfoGainAttributeEval **AttributeSelection**-era sartuko dira eta filtroa aplikatuko da trainBOWFSS.arff sortuz.

Azkenengo pausua hiztegi berri bat sortzearen **StringToWordVector**-en sortutako hiztegia iteratzea eta trainBOWFSS-rekin kointziditutako atributuak idatzi egin dira hiztegi berri batean. Hiztegi berriarekin dev.arff-ri **FixedDictionaryString**-

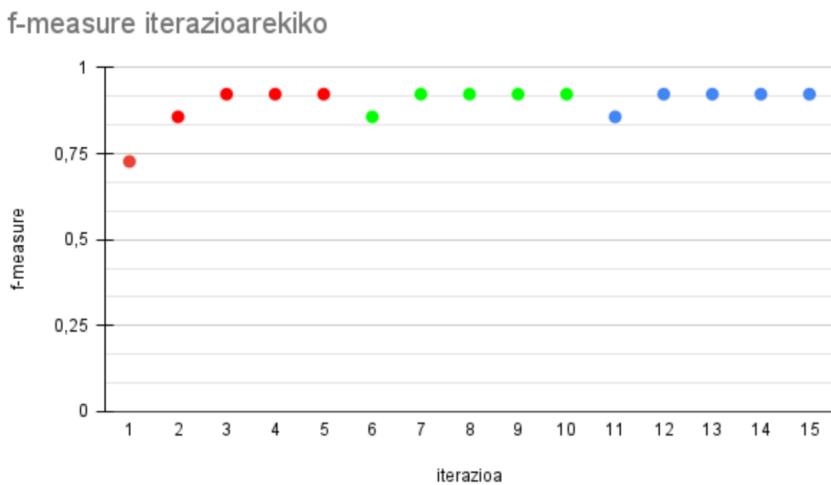
**ToWordVector** filtroa aplikatu zaio train.arff-ri aplikatutako **NominalToString** eta **RenameAttribute** aplikatu ostean eta devBOWFSS.arff sortu egin da. Azkenik, **Reorder** filtroa erabili da devBOWFSS eta trainBOWFSS-k atributuak orden berean izatearren.

Test blind csv-aren aurreprozesamendua aurrekoen antzekoa da, lehengo **replace** funtzioa erabili da karaktere bereziak eta komatxoak kentzeko, gero .arff fitxategi batera bihurtu da, **NominalToString** eta **RenameAttribute** aplikatu zaio eta azkenik, lehen sortutako hiztegi berriarekin **FixedDictionaryStringToWordVector** filtroa aplikatu zaio eta baita **Reorder** ere atributuak ordenatzeko.

### 3.2 Emaitza esperimentalak eta emaitza finalen diskusioa

Parametro ekorketa egiteko estadistika bat hobetza bilatu behar da, ataza honetan klase minoritarioaren f-measure hobetza bilatu da, horretarako ez 0 diren klasearen balioen artean maiztasun minimoa duena aukeratu da. Klase hori hartuta frogak egitean lortutako f-measurearen balioak NAN izan dira, beraz ez zen baliozkoak ataza honetarako eta NAN ematen ez duen hurrengo maiztasun txikiena duen klasea hartu da, 37 indizedun Breast Cancer.

Ekorketaren emaitzak, 15 iterazioen ostean lortutako f-measure balioak, 7. irudian laburbildu dira. Grafikoaren ardatz bertikalean f-measurek hartzen dituen balioak ageri dira, eta ardatz horizontalean ekorketa egitean zer iterazioan dagoen zehazten da, amaitzeko f-measure balioak adierazteko kolore desberdinak erabili dira kernel desberdinak adierazteko. Gorri ageri diren puntuak PolyKernel eta exponentari dagozkie, non exponentearen balio tartea [1-5] den; berdez ageri diren puntuak RBFkernelari eta gammari dagozkie, non gammaren balio tartea [1-5] den; eta amaitzeko, urdinez ageri diren puntuak Puk kernelari eta omegari dagozkie, non omegaren balio tartea [1-5] den.



**7. irudia:** Parametro ekorketaren diseinua.

7 irudian ageri diren emaitzak aztertuz ikusten da kernela PolyKernel denean, exponentearen balio tartea [1-5] doala. Itearatzen doanean, bere balioa 1 denenan f-measure balio txikiarena hartzen du eta exponentearen balioa 3 eta 5 balioen tartean konstante mantentzen da eta exponentea 2 zenenan baino f-measure hobea lortzen da. Kernela RBFkernela denean gammaren balio tartea [1-5] doa. Iteratzen doanean, bere balioa 1 denenan f-measure balio txikiarena hartzen du eta hurrengo iterazioan f-measure hobetzen da eta tartea amaitu arte konstante mantenduko da, hau da, ez da hobetuko bigarren iteraziotik aurrera eta berdin mantenduko da. Amaitzeko, Kernela Puk denean, omegaren balio tartea [1-5] da. Itearatzen doanean, bere balioa 1 denenan f-measure balio txikiarena hartzen du eta RBFkernelarekin gertatzen

den bezala f-measure ez da hobetuko bigarren iteraziotik aurrera eta berdin mantenduko da. Gainera, RBFkernela eta Puk kernelak bakoitzak, gamma eta omega parametroak erabiltzen dutela hain zuzen ere, f-measure balio berak dituzte haien parametroen iterazio guztieta.

Behin ekorketa amaituta, modelo optimoa eraikitzen da lortutako parametro optimoekin, eta modelo horri ebaluaketa ez-zintzoa eta 10-fold cross validation egiten zaizkio. Ebalueketak burutu ostean lortutako f-measure estatistikoak konparatuko dira baselineri ebaluazio bera egin ostean lortutako estatistikoekin 1.

	Ez-zintzoa		10-fold cross validation		Hold - out	
	f-measure	accuracy	f-measure	accuracy	f-measure	accuracy
Baseline	0.7101	50.16%	0.5833	41.29%	0.6666	51.43%
Model	0.9841	98.42%	0.4545	33.15%	1.0	99.04%

**1. taula:** Aztertutako bi ereduetako bakoitzaren errendimendua konparatzeko lortutako f-measure.

Bi modeloetan bururtutako bi ebaluazioen emaitzak aztertzean agerikoa da ebaluaketa ez-zintzoak beti goi-bornea ematen duela. Horretaz aparte, ikus daiteke ekortutako algoritmoak lortzen dituen emaitzak apartekoak direla 98.42% eta 99.04% -ko accuracy-a lortuz ebaluazio ez-zintzoan eta hold-out -ean. 10 fold cross-validationean, ordea, asko jaitsi egiten da bai f-measure eta bai accuracy-a algoritmoarekin; hau, 50 klase izanda datu-sorta 10 zatitan banatzean klase guztiekin erre-presentazio nabarmenenik ez dutelako gertatzen da.

Exekuzio denboraren aldetik, baseline-k 135 segundu iraun ditu eta algoritmoak, berriz, 333 segundu. Hau da, bien artean hiru minutu eta erdiko aldea dago. Exekuzio denbora eta kalitatea kontuan hartuz, argi dago algoritmoa askoz eraginkorragoa dela eta begi bistako aukera dela, exekuzio denbora apur bat handitu arren kalitatea asko handitzen delako.

Train eta dev batuta 3200 instantzia baino gehiago egonda, baieztau dezakegu lortutako emaitzak konsistenteak eta zuzenak direla.

Proiektuaren hasieran, **AttributeSelection** egiterakoan, 20 atributuarekin barrik geratzen ginen eta kalitatea oso txarra zela konturatu ginen. Ondoren 100 eta 200 atributuarekin probatuz, kalitatea nahiko igotzen zela jabetu ginen. Azkenik, 1000 eta 2000 atributuarekin probatu genuen eta 2000 atributuarekin geratu ginen optimoena iruditu zitzaigulako.

### 3.3 Exekuzio adibidea

(Aurreprozesamendua.jar):

```
java -jar path/to/Aurreprozesamendua.jar (+) path/to/data/data_og.csv (+) path/-  
to/data/data.arff (+) path/to/data/data_garbi.csv (+) path/to/data/hiztegi.txt (+)  
path/to/data/train_BoW_FSS.arff (+) path/to/data/test_BoW_FSS.arff (+) path/-  
to/data/hiztB.txt
```

(Sailkapena.jar):

```
java -jar path/to/Sailkapena.jar path/to/model/SMO_inf.model (+) path/to/da-  
ta/testBlind.csv (+) path/to/model/iragarpen.txt (+) path/to/data/hiztegiF.txt (+)  
path/to/data/test_blind.arff (+) path/to/data/garbia.csv
```

(Baseline.jar):

```
java -jar path/to/Baseline.jar (+) path/to/data/va_trainBoW_FSS.arff (+) path/-  
to/data/va_DevBoW_FSS.arff (+) path/to/model/eval.txt (+) path/to/model/i-  
ragarpenak.txt (+) path/to/data/bilketa.arff
```

(Ekorketa.jar):

```
java -jar path/to/Ekorketa.jar (+) path/to/data/va_trainBoW_FSS.arff (+) path/-  
to/data/va_DevBoW_FSS.arff (+) path/to/model/SMO.model (+) path/to/mo-  
del/ebaluazio.txt (+) path/to/data/data_raw.arff (+) path/to/data/histegiF.txt (+)  
path/to/data/osoa.arff
```

## 4 Ondorioak eta etorkizuneko lana

### 4.1 Atazaren laburpena

Verbal autopsy atazak, transkribaturiko elkarritzeten bidez lortutako informazioa baliatuz hildako pertsona baten gaixotasuna edo heriotzaren zergatia iragartzean datza.

Informazioaren tratamendu hau egiteko, SMO ikasketa algoritmoaz baliatu gara, Support Vector Machine deritzon multzoko kide dena. Hauek, datu sortak osaturiko espazioa hobeto zatitzen duten mugak topatzean oinarritzen da, bektore auxiliarren arteko tartea maximizatzu.

### 4.2 Sendotasunak eta ahuleziak

Alde batetik, laborategiko zereginetako konstanteak izan garenez taldekideok, proiektuari zegokion garapen karga nahiko arindu da, kode-lerro asko berrerabili baitira. Bestetik, programaren funtzionamendu egokia bermatu da proba anitzen ostean. Horretaz aparte, honen erabilera behe mailara egokitu da, edonork erabili ahal izateko (iruzkin asko, eta funtzionalitateen azalpen praktikoak adibideekin). Azkenik, esparru esperimentalean azaldu den bezala programak eskaini dezakeen kalitatea asko handitu dela esan genezake.

Beste aldetik, proiektuaren konplexutasuna ez dela oso handia izan esan dezakegu. Izan ere, aukeratutako algoritmoa ez da konplexuenetarikoa. Gainera, garbitutako datu sorta bat aukeratu dugu eta ez ‘open response’ izena zuen beste datu sorta zailago bat.

### 4.3 Ondorioak

Proiektu honen garapena igaro ahala, hainbat ondorio ateratzen joan gara. Hurrengo proiekturei begira hartuko ziren neurriak eta egingo ziren dinamika aldaketa, eta ikasketa prozesuan eta atazen jorratzea eman diren ahala barneratu diren kontzeptuak.

Lortutako programa, nahiz eta ataza konkretu baten betebeharraak asetzeko gratua izan, nahiko malgua dela esan genezake (testu meatzariko esparruan noski). Egokituriko datu sortari loturiko optimizazio faktoreak aldatuz gero, emaitzen kalitatea handituko zen beste datuentzat. Gainera, inolako aldaketarik egitekotan, lan hau errazteko ohitura onak jarraitu direla bermatu dezakegu, iruzkinak eta funtzionalitateen banaketa aproposa hala nola.

Proiektua berriro hasiko bagenu orain dauagun jakinduriarekin, SMO ikaske-  
ta algoritmoa erabili beharrean LibSVM algoritmoa erabiliko genuke, proiektuari  
konplexutasun maila igotzearen. Hasieratik aldatuko genukeen beste puntu bat,  
atazen lan kargaren eta hauei eskeinitako orduen antolakuntza izango zen. Egia da,  
beste ikasgaietako azterketak izan direla eta hainbat arazo izan dugula proiektuare-  
kin ekiteko, baina lan kargaren banaketa ez dela optimoa izan ere onartu genezake.  
Lan orduak era uniformean banatu genituzke, eta ez entrega data gerturatzen doan  
heinean handitzan doan kurba exponentzial baten gisa. Berdina gertatzen da, gara-  
penerako beharrezkoak ziren ezagutza eta tekniken ikerketarekin.

## 5 Bibliografia

### Erreferentziak

- [1] Alpaydin, E. (2010). Introduction to Machine Learning. MIT Press.
- [2] Yu Wan, Zhuo Wang and Tzong-Yi Lee (2020). Incorporating support vector machine with sequential minimal optimization to identify anticancer peptides.  
<https://bmcbioinformatics.biomedcentral.com/articles/10.1186/s12859-021-03965-4>