Universidad del País Vasco
Euskal Herriko Unibertsitatea
BILBOKO INGENIARITZA ESKOLA
ESCUELA DE INGENIERÍA DE BILBAO

ERABAKIAK HARTZEKO EUSKARRI SISTEMAK
Hirugarren maila, 31. taldea
2. Lauhilekoa

Text Mining Proiektua

`https://github.com/EmmaManna/WEKA_Proiektua`

Xabier Dermit, Jon Gondra eta Emma Manna

2021.eko Apirilaren 21a

Eduki Aurkibidea

Irudi Aurkibidea

Taula Aurkibidea

1 Testu Meatzaritza

1.1 Sarrera

Testu meatzaritzak testuen makina bidezko analisia hartzen du bere baitan. Teknika desberdinak erabiltzen ditu, informazio erauzketatik hizkuntza naturalaren prozesamendura arte (NLP), eta datu meatzaritza, ikasketa mekanikoa eta estatistikako algoritmo eta metodoen bitartez konektatzen ditu. Beraz, kasu honetan, datuak modu orokorrean prozesatu beharrean, testu dokumentuetan jartzen da analisiaren arreta.

Testu meatzaritza definitzerakoan ikerketa arlo desberdinekin erlazionatu daiteke:

- Informazio erauzketa: Lehenengo hurbilpenak testu meatzaritza informazio erauzketan oinarritzen dela suposatzen du. Hau da, testuetatik datuak erauztea.
- Testuetako datu meatzaritza: Testu meatzaritza, datu meatzaritza bezala, testuetan patroi erabilgarriak aurkitzeko xedearekin ikasketa mekanikoko eta estatistikako algoritmo eta metodo desberdinen erabilera bezala definitu daiteke ere. Horretarako testuen aurre-prozesamendua gauzatzea beharrezkoa da. [2]

1.2 Adimen Artifizialaren Alorra

Testu-meatzaritza adimen artifizialeko teknologia bat da (AI), hizkuntza naturalaren prozesamendua (NPL) erabiltzen duena dokumentuen eta datu-baseen testu askea (egituratu gabea) analisirako edo ikaskuntza automatikoko algoritmoak bultzatzeko egokiak diren datu normalizatu eta egituratu bihurtzeko.

Hizkuntza naturala ulertzeak testu bat (edo beste sarrera bat, adibidez, mintzamena) "irakurtzen"laguntzen die makinei, eta gizakiak ingelesa, gaztelania edo txinera bezalako hizkuntza naturala ulertzeko duen gaitasuna simulatzen du.

Teknologia gisa, lengoaia naturalaren prozesamendua adin nagusitasunera iritsi da azken hamar urteotan, Siri eta Alexa bezalako produktuekin eta Googleren ahots bidezko bilaketarekin, erabiltzaileen eskaerak ulertzeko eta erantzuteko LBP erabiltzen baitute. Halaber, testu-meatzaritzako aplikazio sofistikatuak garatu dira hainbat arlotan, hala nola ikerketa medikoan, arriskuen kudeaketan, bezeroarentzako arretan, aseguruetan (iruzurrak detektatzea) eta testuinguruaren publizitatean.

Hizkuntza naturala prozesatzeko gaur egungo sistemek testu batean oinarritutako datu kopuru mugagabeak azter ditzakete, nekerik gabe eta modu koherente eta inpartzialean. Testuinguru konplexuen barruko kontzeptuak uler ditzakete eta hizkuntzaren anbiguotasunak deszifratu, gertaerak eta funtsezko erlazioak ateratzeko edo laburpenak emateko.[7]

1.3 Aplikazio Errealak Enpresan

Datuen analisia gorabidean dago, eta hori guztia "datuen uholdea"izenekoari esker. Hainbeste datu daude eta hainbeste datu berri sortzen dira etengabe, analisia nahitaezkoa dela, ezin baitira guztiak gorde.[3] Gizateria bost informazio exabyte ekoiztetik, 2005ean, 48 ordutan ekoiztera pasa da. 2020an, 35 informazio-ZettaByte sortu ziren (1 ZettaByte 1.000 exabyteren baliokidea da eta hauek 1.000 milioi Terabyteren baliokidea). Baina enpresek benetan erabiltzen dutenaren %93 ezkutuan egongo da, ustiatu gabe, eta iaz, berriz, %80, International Data Corporation (IDC) erakundearen arabera. Erabiltzen diren datuak enpresen esku dagoen informazio-multzoaren izebergaren punta dira. Itzalean dirauten datu hauek, eta, beraz, existituko ez balira bezala dira, *Dark Data* deitzen dira. [4] Testu-meatzaritza egituratu gabeko datuak aztertzeko eta prozesatzeko modu garrantzitsuenetako bat da, munduko datuen ia %80 osatzen baitute. Hor sartzen dira jokoan testu-meatzaritzako aplikazioak, testu-meatzaritzako tresnak eta testu-meatzaritzako teknikak.

Testu-meatzaritzako teknikak eta tresnak berehala sartzen ari dira industrian, mundu akademikoan eta

osasunean, enpresetan eta gizarte-baliabideen plataformetan. Horrek testu-meatzaritzako aplikazio batzuk sortzen ditu. Hauek dira gaur egun mundu osoan erabiltzen diren testu-meatzaritzako aplikazioetako batzuk:

- Arriskuen kudeaketa: Enpresa-sektorean porrot egiteko arrazoi nagusietako bat arriskuen analisi egokirik edo nahikorik ez izatea da. SAS Text Miner bezalako testu-meatzaritzako teknologietan oinarritutako arriskuak kudeatzeko softwarea onartu eta integratzeak enpresei lagun diezaieke enpresa-merkatuaren egungo joera guztiak eguneratzen eta arriskuak arintzeko gaitasunak indartzen.
- Bezeroarentzako arreta-zerbitzua: Testu-meatzaritzako teknikak, bereziki NLP, gero eta garrantzi handiagoa hartzen ari dira bezeroaren arretaren arloan. Enpresak testua aztertzeko softwarean inbertitzen ari dira, bezeroaren esperientzia globala hobetzeko, hainbat iturritako testu-datuak eskuratuz (inkestak, bezeroen iruzkinak, bezeroen deiak, etab.). Testuen analisiaren helburua da enpresaren erantzun-denbora murriztea eta bezeroen kexak azkar eta eraginkortasunez konpontzen laguntzea.
- Iruzurrak detektatzea: Testuen analitikak, testu-meatzaritzako teknikek lagunduta, aukera paregabea ematen du testu formatuko datu gehienak biltzen dituzten eremuetarako. Aseguru eta finantza konpainiak aukera hau aprobetxatzen ari dira. Testu-analisien emaitzak eta datu egituratuak konbinatuz, enpresa horiek erreklamazioak azkar prozesatzeko eta iruzurrak detektatzeko eta prebenitzeko gai dira.
- Enpresa-inteligentzia: Erakundeak eta enpresak testu-meatzaritzako teknikak baliatzen hasi dira beren enpresa-adimenaren parte gisa. Bezeroen portaeraren eta joeren ikuspegi sakona emateaz gain, testu-meatzaritzako teknikek ere laguntzen diete enpresei arerioen indarguneak eta ahulguneak aztertzen, eta horrek abantaila ematen die merkatuan. Cogito Intelligence Platform eta IBM *text analytics* testu-meatzaritzako tresnek marketin-estrategien errendimenduari, bezeroen eta merkatuaren azken joerei eta abarri buruzko informazioa ematen dute.
- Sare sozialen azterketa: Baliabide sozialen plataformen errendimendua aztertzeko soilik diseinatutako testu-meatzaritzako tresna asko daude. Horiek berriak, blogak, posta elektronikoak, etab. oinarri hartuta, linean sortutako testuak arakatzen eta interpretatzen laguntzen dute. Gainera, testu-meatzaritzako tresnek eraginkortasunez azter ditzakete argitalpen kopurua, 'Gustatzen zait' -ak eta bere markaren jarraitzaileak sare sozialetan, eta, horri esker, beren markarekin eta lineako edukiarekin elkarreragiten duten pertsonen erreakzioa uler dezakete. Azterketak 'zer dagoen modan eta zer ez' ulertzea ahalbidetuko dio. [5]

1.4 Erronkak Datu Numerikoekin alderatuta

Datu-meatzaritza egituratutako datuez arduratzen den bitartean (datu oso formateatuak, adibidez, datu-baseetakoak edo ERP sistemetakoak), testu-meatzaritza egituratu gabeko testu-datuez arduratzen da, hau da, aldez aurretik definitu eta inola ere antolatu gabe dauden testuez, sare sozialetakoak bezala.

2 Aurre-Prozesamendua

2.1 Datu Errepresentazioak

Datuen analisia egiteko 4 datu formatu tratatu ditugu:

- Raw: Datu gordinak, lehen mailako datu gisa ere ezagutzen direnak, iturri batetik jasotako datuak dira (adibidez, zenbakiak, tresnen irakurketak, zifrak, etab.). Gure kasuan iturri hori Twitter izan da eta datuak testu motan gordetako tweet-ak.
- BoW: 'Bag of Words' eredua hizkuntza naturala prozesatzeko eta informazioa berreskuratzeko (*Information Retrieval* IR) erabiltzen den irudikapen sinplifikatua da. Eredu honetan, testu bat (esaldi edo dokumentu bat bezala) bere hitzen poltsa (multzo anitzekoa) bezala irudikatzen da, gramatika eta haren hitzen ordena kontuan hartu gabe, baina aniztasunari eutsiz.

- TF·IDF: 'Term Frequency times Inverse Document Frequency' akronimoa da, 'Terminoaren maiztasuna bider dokumentuaren alderantzizko maiztasuna' bezala itzul dezakeguna. Termino edo esaldi bat dokumentu jakin baten barruan zer maiztasunekin agertzen den neurtzen du, eta termino hori dokumentu-bilduma oso baten barruan aipatzen duten dokumentu-kopuruarekin alderatzen du.
- Sparse/NonSparse: Aljebra lineal numerikoan, *sparse matrix* edo matrize sakabanatua matrize handi bat da, non bere elementu gehienak zero diren. Wekan, instantzia sakabanatu batek zero ez diren atributuen balioetarako bakarrik behar du biltegiratzea. Helburua aurrez zehaztutako balio asko dituzten datu-multzoen biltegiratze-baldintzak murriztea da. Kontran, sakabanatuak ez diren instantziak balio guztiak gordetzen ditu, zero direnak barne.

2.2 Adibideak

Gure dokumentuak tweet-ak direnez raw, BoW eta TF·IDF-ren adibideak horiekin egingo dira:

- Raw: Erabilitako datu mota ikusten da: tweet-aren id-a, tweet-a arrazista edo ez den eta azkenik, tweet-a (ikusi Taula 1).
- BoW: Atributu bakoitza zein den jakiteko ikusi Taula 2.
  - {0 yes,17 1,20 1,21 1,22 1,23 1,24 1,25 1,26 1,27 1,28 1,29 1,30 1}
  - {1 1,2 1,3 1,4 1,5 1,6 1,7 1,8 1,9 1,10 1,11 1,12 1,13 1,14 1,15 1,16 1,17 1,18 1,19 1}
- TF·IDF: Atributu bakoitza zein den jakiteko ikusi Taula 2.
  - {0 yes,20 0.480453,21 0.480453,22 0.480453,23 0.480453,24 0.480453,25 0.7615,26 0.480453,27 0.480453,28 0.480453,29 0.480453,30 0.7615}
  - {1 0.480453,2 0.480453,3 0.480453,4 0.480453,5 0.480453,6 0.480453,7 0.480453,8 0.480453,9 0.480453,10 0.480453,11 0.480453,12 0.480453,13 0.480453,14 0.480453,15 0.480453,16 0.480453,18 0.480453,19 0.7615}

id	label	tweet
8	no	the next school year is the year for exams. can't think about that #school #exams #hate #imagine #actorslife #revolutionschool #girl
499	yes	are you #black & feel like the are stomping on you? #retweet #tampa #miami

Taula 1: Datu Gordin moduko 2 Doc.

2.3 Datu Sorta

Ataza garatzeko erabili den datu sorta *kaggle*-etik aterata dago, `https://www.kaggle.com/arkhoshghalb/twitter-sentiment-analysis-hatred-speech`. Bertan proposatutako ataza Tweetetako esaeretan gorrotoa detektatzea da. Sinpleagoa egiteko, gorrotoa dagoela esaten da sentimendu arrazistak edo sexistak badaude. Beraz, Tweet arrazistak zein sexistak detektatzea eskatzen da.

Bi datu sorta ematen dira, alde batetik entrenamendu multzo gainbegiratua eta bestetik test multzo ez gainbegiratua, *blind*. Multzo gainbegiratuan klaseak bi balio posible har ditzake, '0' tweeta ez bada arrazista/sexista eta '1' arrazista/sexista izatekotan. Gainera, Tweeterreko erabiltzaile guztien *username*a '@user'-ekin ordezkatu dira.

@attribute	num/nom
klasea	no,yes
#actorslife	numeric
#exams	numeric
#girl	numeric
#hate	numeric
#imagine	numeric
#revolutionschool	numeric
can	numeric
exams	numeric
for	numeric
is	numeric
next	numeric
school	numeric
t	numeric
that	numeric
the	numeric
think	numeric
year	numeric
#black	numeric
#miami	numeric
#retweet	numeric
#tampa	numeric
&amp	numeric
are	numeric
feel	numeric
like	numeric
on	numeric
stomping	numeric
you	numeric

Taula 2: Atributu zerrenda

Datu sorta hau aukeratzearen arrazoi nagusiena gure atazarekin zuen egokitzapena izan zen. *Kaggle*-en proposatutako ataza eta guri egokitutakoa funtsean berdinak dira, beraz datu sorta oso egokia iruditu zitzaigun guk beharrezkoak genituen baldintza guztiak betetzen zituelako. Horretaz gain, interesgarria iruditu zaigu nola erabili daitekeen datu meatzaritza egunero erabiltzen dugun sare sozial batean pertsonen sentimenduak aztertzeko. Hala eta guztiz ere, garapenean zehar, emaitzak ikustean zehazki, konturatu gara datu sortak eragozpen bat daukala, 'yes' klasea duten instantzia gutxi daude entrenamendu multzoan, arrazistak edo sexistak diren adibide gutxi daude, alegia. Horrek, aurrerago demostratuko den moduan, eragin handia izango du gure eredu iragarlearen jardueran.

Horretaz gain, beste erabiltzaileek *Kaggle*-eko ataza ebazteko egindakoa ere aztertu da. Aurreprozesamenduari dagokionez, erabiltzaile gehienek datuen garbiketa nahiko antzekoa gauzatu dute: karaktere bereziak kendu, puntuazio markak eta hitz motzak (*me, my, it* adibidez) ezabatu dira, besteak beste. Errepresentazio espaziorako, proba gehienak BoW eta TF·IDF erabiliz egin dira, hala ere, Word2Vec eta Doc2Vec errepresentazioak ere agertzen dira. Azkenik, gehien erabiltzen diren sailkatzaileak *RandomForest*, *Naïve Bayes* eta *XGBoost* dira (*Logistic Regression*, SVM edota *Decision Tree* ere aurkitu ditzakegu).

Hurbilpen horiek ikusita, gure ataza bideratzeko ideia orokor bat izatea lortu da. Haatik, probak sailkatzaile desberdinekin egiten diren arren, askok ez dute atributu kopurua kontutan hartzen, ezta atributu erredundanteen eragina, hobekuntza posible bat izan zitekeena bai sailkatzailearen jarduera aldetik eta baita kostu konputazional aldetik. Gainera, orokorrean erabiltzen diren sailkatzaileak entrenatzeko denbora eta kostu konputazional baxukoak dira guri esleitutako *Multilayer Perceptron*-arekin konparatuz.

Azkenik, lortutako emaitzak konparatuz, antzeko ondorioak lortzen dira. Orokorrean, TF·IDF errepresentazioak BoW baino jarduera pixka bat hobeagoa lortzen du, baina ezberdintasun oso nabarmenik egon gabe. Gainera, eredu sinpleak jarduera nahiko ona lortzen dute, klase minoritarioarekiko 'yes' *F-Measure* nahiko balio baxuak lortzen diren arren (0.5-0.6 inguruan), guri gertatzen zaigun moduan (ikusi Taula 3 eta Taula 4).

2.4 Xehetasunak

Programa 3 ataza nagusietan banatzen da eta ataza horien barruan zenbait programa daude:

- Testuen errepresentazio bektoriala: GetRaw, TranformRaw eta MakeCompatible.
  - GetRaw: Datuen dokumentua *ad-hoc* ARFF formatura bihurtzen du programa honek. Honen bitartez, train zein test datuei formatu egokia ematen die, beharrezkoak ez diren atributuak kenduz eta testuari arazoak eman ditzaketen karaktereak kenduz.

    Transformazioa *ad-hoc* egiten denez, gure datu multzoan aldaketa zehatz batzuk egin behar izan ditugu gero datuak arazorik gabe prozesatu ahal izateko. Alde batetik, datuak Tweet-ak direnez arazoak eman ditzaketen karaktere ugari aurkituko ditugu, emotikonoak adibidez. Hori dela eta, ASCII kodean ez dauden karaktere guztiak ezabatu egingo dira. Era berean, komatxoak ere filtratuko dira kontrabarra jarriz. Bestalde, egindako beste aldaketa nabarmenena klase atributuaren izena aldatzea izan da. Arrazoia sinplea da, *StringToWordVector* filtroa aplikatzean arazoak ematen zituen. Datu sortaren klasearen izena edo identifikatzailea 'label' zen hasieran, filtroak Tweeten hitz bakoitza atributu bihurtzen du, eta kasua ematen zen non hitz bat *label* zela, eta ezin dira egon identifikatzaile berdina duten bi atributu. Horrela, eta datu sorta ingelesez dagoela jakinda, 'klasea' identifikatzailearekin ordezkatu da arazo gehiagorik ez izateko.
  - TranformRaw: Datu gordinak erabiltzaileak nahi duen espazio bektorialera aldatzeko eta espazio horren hiztegia lortzeko pentsatuta dago programa hau. Konturatu ginen Wekaren iragazkiek egoera onenean funtziona zezaten hobe zela "Options"aukera erabiltzea, ezarrita dituen funtzioen bidez aukerak pasatzea baino. Horrela, gure programak Wekak egiten dituen urratsak zintzo jarraituko dituela ziurtatzen dugu, akatsak saihestuz.
  - MakeCompatible: Datu gordinak lortu eta gero eta atributu hautapena egin ondoren, test multzoa espazio honi egokitzeko beharra dago. Prozesu hau egiteko zenbait berezitasun aurkitu genituen: lehenik, train eta test *header*-ak berdindu behar dira, test multzoko *class* atributua *string* motatik *nominal* motara aldatuz eta beharreko balioak emanez {no,yes}; bigarrenik, guk gordetako hiztegia eta Wekak itzulitako hiztegia desberdinak ziren *Attribute Selection* egin eta gero, noski, atributuak ezabatu zirelako, orduan, hiztegiaren egokitzapena egiten da; azkenik, train instantziak egokitzeko aukera berdinak eskaintzen dira eta prozesu hau egin ondoren emaitzako atributuak egokitu behar ziren klasea azken tokira berrantolatuz.
- Sailkatzailea: FSS, GetBaselineModel, ParamOptimization eta GetModel.
  - FSS: Entrenamendu multzoko atributu egokienak hautatu behar dira, erredundanteak diren atributu gehiegi izatea kaltegarria izan daitekeelako. Alde batetik, doikuntza eza edo *underfitting* eta bestetik bariantza altuak eta sesgoak estatistiketan eman daitezke. Hori dela eta, informazio galera txikiena emango lukeen atributuen hautapena gauzatzen da. Horretaz gain, test multzoa ere egokitzen da.
  - GetBaselineModel: *Logistic Regression* erabiliz markatu da behe-bornea, kalitatea estimatzeko hiru ebaluazio metodo erabili dira: Ez-zintzoa, 10-*fold cross validation* eta 100 partiketa ezberdinekin egindako *hold-out* baten batezbesteko erantzunak.

  - ParamOptimization: Parametro ekorketa teknika erabiliz, esleitutako algoritmoarentzako parametro optimoenak kalkulatzen eta gordetzen dira.
  - GetModel: Kalkulatutako parametroekin esleitutako algoritmoaren modeloa eraikitzen da, eta honen kalitatearen ebaluazio bat egiten da. Kalitatea estimatzeko hiru ebaluazio metodo erabili dira: Ez-zintzoa, 10-*fold cross validation* eta 100 partiketa ezberdinekin egindako *hold-out* baten batezbesteko erantzunak.

- Iragarpena: Predictions.
  - Predictions: Iragarpenen aldetik informazio sentikorrena eskaini dugu. Instantzien klasifikazio zehaztua non iragarpena eta bere balioaren alderapena gordetzen den, ebaluazioari buruzko zehaztasunaren banakapena sortzen du klase bakoitzerako, informazioa berreskuratzeari buruzko hainbat estatistika txertatuz, hala nola *true/false positive rate*, *precision/recall/F-Measure* eta lortutako nahasmen matrizea. Gainera, garatutako interfaze grafikoan testu soila sartzeko aukera ere ematen da, test fitxategiaz aparte, instantzia bakarraren iragarpena egiteko.

Proiektu hau egiteko hiruko taldeak osatu direnez, lana honela bereizi dugu: Emma eta Jon lehenengo eta azken atazaz eta FSS eta *baseline*-a lortzeaz arduratu dira; izan ere, Xabierrek, bigarren atazaz arduratu denak, gure taldeari esleitutako algoritmoari buruzko ezagutza handiagoa zuen (*Multilayer Perceptron*).

2.5 Atributu Hautapena

Atributu Hautapena ikaskuntza automatikoaren kontzeptu nagusietako bat da, eta horrek eragin handia du ereduaren errendimenduan. Ikaskuntza automatikoko ereduak entrenatzeko erabiltzen diren datuen ezaugarriek eragin handia dute lor daitekeen errendimenduan. Garrantzirik gabeko ezaugarriek edo partzialki garrantzitsuak direnek eragin negatiboa izan dezakete ereduaren errendimenduan. Atributu hautapenean automatikoki edo eskuz hautatzen dira iragartzeko aldagaian edo interesatuta gauden emaitzan gehien laguntzen duten ezaugarriak. Zein dira atributu hautapenaren abantailak datuak modelatu aurretik?

- Gehiegizko doikuntza murrizten du: datu erredundante gutxiagok aukera gutxiago ematen dituzte zaratan oinarritutako erabakiak hartzeko.
- Zehaztasuna hobetzen du: datu engainagarri gutxiagok esan nahi du modelatzearen zehaztasunak hobetu egiten duela.
- Entrenamendu-denbora murrizten du: datu-puntu gutxiagok algoritmoaren konplexutasuna murrizten dute, eta algoritmoak azkarrago entrenatzen dira.[6]

3 Esparru Teorikoa

Atal honetan bi eredu iragarle konparatuko dira, bata *baseline*a *Logistic Regression* ereduarekin garatu dena eta bestea konplexuagoa *Multilayer Perceptron*. Horrela, optimoena zein den aztertuko da kalitatearen estimazioa gauzatuz.

3.1 Baseline

Hasteko entrenatzen errazago eta azkarragoa den eredu bat garatu da, *Logistic Regression*. Eredu honek behe borne bat emango digu, eta honi esker *Multilayer Perceptron*-ekin konparazioa gauzatuko da, jakiteko zein den egokiagoa erabiltzen gure kasurako.

3.1.1 Esparru Teorikoa

Erregresio logistikoa ulertzeko lehenengo erregresio lineala ulertu behar dugu. Erregresio lineala erraz erabil daiteke zenbakizko atributuak dituzten domeinuetan sailkatzeko. Izan ere, edozein erregresio-teknika erabil dezakegu, lineala edo ez-lineala, sailkapenerako. Trikimailua klase bakoitzerako erregresio bat egitean datza, irteera 1-ekoa ezarriz klaseari dagozkion entrenamendu-instantzietarako eta 0 klasekoak ez direnetarako. Emaitza adierazpen lineal bat da klasearentzat. Jarraian, klase ezezaguneko proba baten adibidea emanda, adierazpen lineal bakoitzaren balioa kalkulatzen da eta handiena aukeratzen da. Eskema honi, batzuetan, erantzun anitzeko erregresio lineala deitzen zaio.

Erantzun anitzeko erregresio lineala ikusteko modu bat klase bakoitzerako zenbaki-funtzio batera hurbiltzen dela irudikatzea da. Kide izateko funtzioa 1 da mota horretako kasuetarako, eta 0 gainerako kasuetarako. Instantzia berri bat emanda, klase bakoitzekoa den kalkulatuko dugu, eta handiena aukeratuko dugu.

Erantzun anitzeko erregresio linealak emaitza onak ematen ditu praktikan. Hala ere, bi eragozpen ditu. Lehenik eta behin, sortzen dituen pertenentzia-balioak ez dira berez probabilitateak, 0 eta 1 bitarteko tartetik kanpo gera baitaitezke. Bigarrenik, karratu txikienengatiko erregresioak esan nahi du erroreak ez direla soilik estatistikoki independenteak, baizik eta desbideratze estandar berarekin ere banatu ohi direla; suposizio hori nabarmenki urratzen da metodoa sailkapen-arazoei aplikatzen zaienean, behaketek 0 eta 1 balioak bakarrik hartzen baitituzte.

Erlazionatutako estatistika-teknika batek, erregresio logistiko izenekoak, ez ditu arazo horiek jasaten. 0 eta 1 balioak zuzenean hurbildu beharrean, helburua gainditzen denean legez kanpoko probabilitate-balioak lortzeko arriskua izanik, erregresio logistikoak eredu lineal bat eraikitzen du, objektibo aldagai eraldatuan oinarrituta.[8]

Erregresio logistikoak funtzio logistikoa erabiltzen du funtzio linealaren ordez. Funtzio logistikoa (4), funtzio sigmoidea ere deitua, estatistikariek garatu zuten ekologian populazioaren hazkundearen propietateak deskribatzeko, azkar handituz eta ingurunearen karga ahalmen maximora iritsiz. S formako kurba bat da, edozein balio errealeko zenbaki har dezakeena eta 0 eta 1 arteko balioa eman diezaiokeena, baina inoiz ez zehazki muga horietan.[9]

$$1/(1+e^{-value}) \tag{1}$$

Erregresio logistikoaren ekuazioa erregresio linealaren ekuaziotik abiatuta lor daiteke. Hona hemen erregresio logistikoaren ekuazioak lortzeko urrats matematikoak:

- Badakigu zuzenaren ekuazioa honela idatz daitekeela:

$$y = b_0 + b_1x_1 + b_2x_2 + b_3x_3 + ... + b_nx_n \tag{2}$$

- Erregresio logistikoan, y 0 eta 1 artean egon daiteke; beraz, horretarako, aurreko ekuazioa honela zatituko dugu (1-y):

$$y/(1-y);\ 0\ y\text{=}0\text{-rentzako, eta infinito } y\text{=}1\text{-entzat} \tag{3}$$

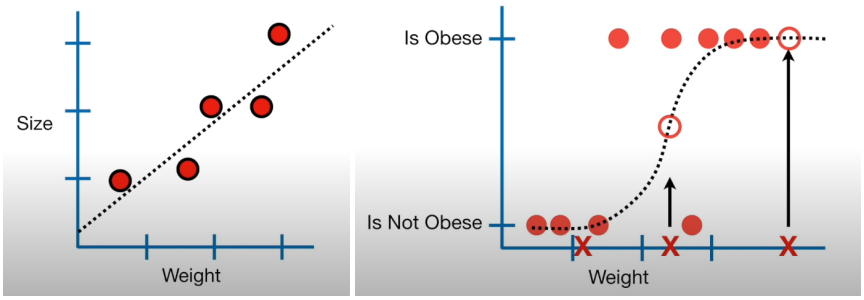
- Baina maila bat behar dugu - [Infinitua] eta + [infinitua] artean, orduan ekuaziotik logaritmoa hartzea honela bihurtuko da:

$$\log\frac{y}{1-y} = b_0 + b_1x_1 + b_2x_2 + b_3x_3 + ... + b_nx_n \tag{4}$$

[10]

Azaldutako hau hobeto ulertzeko hurrengo adibideak ditugu: Imajinatu arratoiak sailkatuko ditugula eta Fig. 1a erregresio linealaren errepresentazio bat aurkezten digu non, adibidez, arratoien tamaina kalkulatu

dezakegu pisua emanda. Fig. 1b, aldiz, erregresio logistikoa modu grafiko batean azaltzen laguntzen digu. Ikus dezakegu, erregresio lineala ez bezala, erregresio logistikoak *True/False* iragarpen bat egiten duela. Erregresio linealak tamaina bezalako iragarpenak egiten dituelako, datu jarraiak. Erregresio logistikoak 0-tik 1-era doan 'S' formako funtzio logistikoa erabiltzen du, orduan, funtzioak adierazten digu arratoiak gizen edo ez dauden haien pisua kontuan hartuz. Oso pisu handia duen arratoi bat pisatzen badugu, probabilitate oso handi bat egongo da gizen egoteko (eskuineko geziak hori adierazten du). Pisu ertaineko arratoi bat pisatzen badugu, %50-ko probabilitatea egongo da gizen egoteko (erdiko gezia). Azkenik, probabilitate oso txikia egongo da gizen egoteko arratoi arin bat pisatzen badugu (ezkerreko gezia). Orduan, klasifikatzeko, arratoi baten gizen egoteko probabilitatea >%50 bada gizen bezala klasifikatuko dugu, bezala 'ez gizen' bezala klasifikatuko dugu.



(a) *Linear Regression* (b) *Logistic Regression*

Fig. 1: *Logistic Regression*: (a) erregresio linealaren adibidea eta (b) erregresio logistikoaren adibidea.

3.1.2 Esparru Praktikoa

Eredu iragarlearen kalitatea estimatzeko ebaluazio metodo desberdinak erabili dira. Alde batetik, kalitatearen estimazioa metodo ez-zintzoa, 100 *hold-out* eta 10-*fold Cross Validation* erabiliz gauzatu da, datu errealistagoak lortzeko asmoarekin. Bestetik, datu multzoaren errepresentaziorako bi formatu desberdin probatu dira, BoW eta TD·IDF (ikusi Taulak 3 eta 4). Emaitzak aztertuz, bi errepresentazioetan kalitate oso antzekoa lortzen dela ikusi daiteke. Nabarmena da, "yes"klasearentzat *Recall* eta *F-Measure* nahiko baxuak lortzen direla, beraz, *Recall* behatuta ereduak errealitatearekiko asmatzen dituen instantzia kopurua ez da oso altua, arrazista edo sexistak diren Tweet-ak ez ditu ondo detektatzen, alegia.

		Logistic Regression		
	Klasea	**Precision**	**Recall**	**F-Measure**
Ez Zintzoa	no	0.959	0.994	0.977
	yes	0.855	0.442	0.583
	W. Avg	0.952	0.956	0.949
	Accuracy		%95.5635	
100 Hold-Out	no	0.958	0.995	0.976
	yes	0.87	0.419	0.566
	W. Avg	0.953	0.955	0.947
	Accuracy		%95.4844	
10 - fCV	no	0.959	0.994	0.976
	yes	0.835	0.436	0.573
	W.Avg	0.95	0.954	0.946
	Accuracy		%95.4415	

Taula 3: *Logistic Regression* jarduera BoW.

		Logistic Regression		
	Klasea	**Precision**	**Recall**	**F-Measure**
Ez Zintzoa	no	0.96	0.994	0.977
	yes	0.853	0.449	0.588
	W. Avg	0.952	0.956	0.949
	Accuracy	%95.5916		
100 Hold-Out	no	0.958	0.996	0.977
	yes	0.877	0.425	0.573
	W. Avg	0.953	0.955	0.948
	Accuracy	%95.547		
10 - fCV	no	0.959	0.993	0.976
	yes	0.834	0.436	0.573
	W.Avg	0.95	0.954	0.948
	Accuracy	%95.4383		

Taula 4: *Logistic Regression* jarduera TD·IDF.

3.2 Esleitutako Algoritmoa

Esleitu zaigun algoritmoa *Multilayer Perceptron* izan da. Algoritmo hau Sare Neuronalen familiako algoritmo mota bat da eta *Perceptron* (edo *Simple Perceptron*) modeloaren bertsio konplexuago bat da. Modelo hau entrenatu ostean, aurretik azaldu den *Baseline* modeloarekin konparatuko da.

3.2.1 Esparru Teorikoa

Multilayer Perceptron ulertzeko, lehenengo Sare Neuronalak nola funtzionatzen duten ezagutu behar dugu. Sare Neuronal batek neurona artifizialen multzo elkar lotua da, modelo matematikoak erabiltzen dituena informazioa prozesatzeko, eta nerbio-sistemak ikasten duen moduan oinarritzen dena. Neuronak, sare baten neurtze unitate minimoa dira, eta bakoitzak operazio matematiko sinple bat burutzen du (batuketak eta biderketak). Neurona batek kalkulatzen dituen operazioak funtzio linealak direnez, neurona bakoitzaren ostean aktibazio funtzioa deitzen den funtzio matematikoa aplikatzen da, linealtasun honekin bukatzeko. Neurona asko konbinatuta, sare neuronal mota desberdinak sortu daitezke.

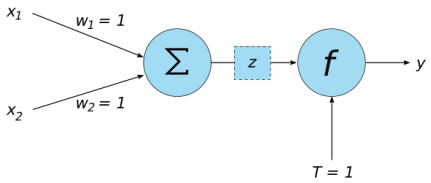


Fig. 2: Neurona baten eskema.

Sare hauek 2 (gutxienez) kapa edo geruza dituzte [11]. Lehenengoa, *input layer* da, kapa hau sarrerako datuak prozesatzen ditu eta bigarren geruzari -*output layer*- pasatzen dizkio emaitza finala kalkulatzeko. Sarearen bi kapa hauen artean, *hidden layer* edo ezkutuko geruza bat edo asko egon daitezke izan beharrez gero. *Perceptron*a prozesatze elementurik sinpleena da. Sarrera atributu batzuk ditu, ingurunetik etor daitezkeenak edota beste *perceptron* batzuen irteerak izan daitezkeenak. [1] Sarrerako atributuek pisu bat izango dute, pisu honek emaitza kalkulatzerako orduan atributu bakoitzak zenbat garrantzia duen adierazten du.

*Simple Perceptron*aren kasuan, sarrera eta irteera geruzekin nahikoa da, algoritmo hau linealki banangarriak diren problemetarako diseinatuta dagoelako. Algoritmo honen pisuak kalkulatzeko, hasiera batean pisuei zorizko balioak atzituko zaizkie, eta iterazioak egiten diren bitartean, balio hauek optimizatuko dira. Algoritmo hau datu sorta gainbegiratuak erabiltzen dituenez, iterazio bakoitzean pisuen balioak zuzenduko dira emaitza optimoa lortu arte. Klaseak linealki banangarriak direnez, *perceptron*ak sarrera espazioa bitan banatu dezake, zati positiboa eta zati negatiboa alegia. Honi esker, *perceptron*ak bi klaseak banatu ditzake irteeraren zeinua

aztertuz. s(·) *threshold function* bezala definituz

$$s(a) = \begin{cases} 1 & \text{if } a > 0 \\ 0 & \text{Otherwise} \end{cases}$$

aukeratu dezakegu

$$choose \begin{cases} C_1 & \text{if } s(w^T x) > 0 \\ C_2 & \text{Otherwise} \end{cases}$$

non a atributuen balioak eta w haien pisuak diren. [1]

Pisu hauen kalkuluarekin, azken finean algoritmoak funtzio matematiko bat definitzen ari du, plano (edo espazio) bat banatzen duen funtzioa; ekuazio lineal bat hain zuzen ere *perceptron*ak sarrera eta irteera bakarra duenean. Sarrera bat baino gehiago izatekotan, ekuazio lineala hyperplano bat bihurtzen da. [1] Kasu honetan, algoritmoak kalkulatzen duen hyperplanoaren ekuaziori (ikusi 5 ekuazioa) erabaki muga deitzen zaio, eta instantziak klasifikatzeko erabiltzen da.[12]

$$w_0 a_0 + w_1 a_1 + w_2 a_2 + ... + w_k a_k = 0 \tag{5}$$

Linealki banangarria den adibide bana AND eta OR ate logikoak izan daitezke (ikusi 3 irudia). Funtzio boolearretan sarrerak bitarrak dira, eta irteera 1 izango da funtzioa *True* bada eta 0 *False* bada. Beraz, bi klaseen sailkapen ataza bat bezala ikus daiteke eta *perceptron*a erabiliz ebatzi daiteke. Beste funtzio batzuk, XOR bezala (ikusi 4 irudia), ezin dira modu horretan ebatzi, ez direlako linealki banangarriak. Hau normala da, bi sarrera bitarrekin lau kasu posible egon daitezkeelako, eta beraz, sarrera bitarreko problema batzuk ez dira linealki banangarriak. [1]

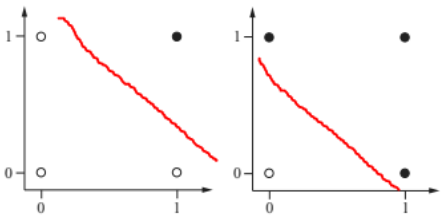


Fig. 3: Linealki banangarriak diren bi problemen adibidea: AND eta OR ate logikoen errepresentazio grafikoa.

Problema batzuetan, erabiliko diren datuak ez dira linealki banangarriak (proiektu honen kasuan adibidez), kasu hauetarako *Multilayer Perceptron* erabiliko da *Simple Perceptron* ordez. Konplexuagoa den algoritmo honek, sarrera eta irteera geruzen artean ezkutuko geruzak dauzka. Praktikan, normalean ez da *hidden layer* bat baino gehiago erabiltzen, ezkutuko geruza ugari dituen sareak aztertzea nahiko zaila izaten baita; baina kasu batzuetan ezkutuko unitate ugari badaude geruzan gehiago erabiltzea komenigarriagoa izan daiteke. [1]

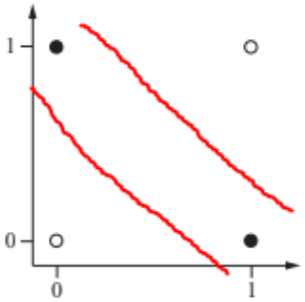


Fig. 4: Linealki banangarria ez den problema baten adibidea adibidea: XOR ate logikoaren errepresentazio grafikoa.

Multilayer Perceptron sarearen konfigurazioa *Simple Perceptron* sarearena baino konplexuagoa denez (ikusi 6 irudia), ezin dezake atributuen pisua sinpleagoa den algoritmo honen moduan kalkulatu. Kasu honetan, atributu bakoitzaren pisua kalkulatzeko, *backpropagation* deitzen den teknika erabiltzen da.

*Perceptron*ak hyperplano bat definitzen du, eta sare neuronalak hyperplano hori inplementatzeko modu bat besterik ez da. Datu instantzia bat emanda, pisuen kalkulua *offline* egin daiteke eta ondoren kargatzen direnean *perceptron*ak irteerako balioak kalkulatzeko erabiltzen ditu. Normalean, *online* ikasketa erabiltzen da baina instantziak banaka ematen direnean instantzia bakoitzaren ostean sarea eguneratzea nahi izaten da, astiro egokitzeko.

Online ikasketan aldiz, errore funtzioa ez da datu multzo osoarekiko kalkulatzen, baizik eta banakako instantzietan oinarrituta.[1] *Backpropagation* teknikak, *Simple Perceptron*-ek bezala, hasiera batean pisuak zoriz hasieratzen ditu, eta pisu hauekin sarearen kalkulo guztiak egin ondoren iragarpen bat lortzen du. Informazioa gainbegiratua denez, iragarpena ondo edo gaizki egin den jakin daiteke, eta instantzia txarto klasifikatu bada, pisuak eguneratuko dira. Kasu honetan sareak neurona asko dituenez, neurona bakoitzaren errorea kalkulatu behar da neurona horren pisua eguneratu ahal izateko.

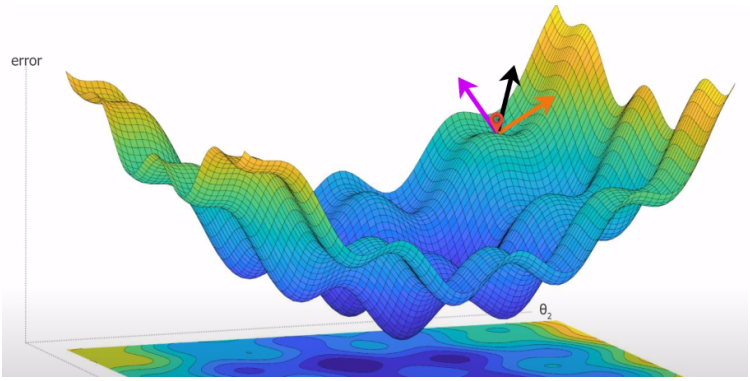
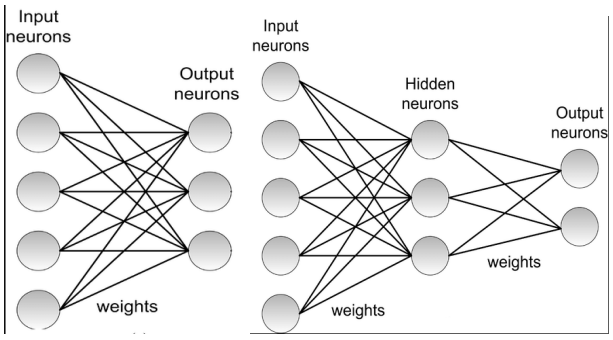


Fig. 5: *Gradientearen jaitsiera* algoritmoaren errepresentazio grafikoa.

Kalkulu hau egiteko, *gradientearen jaitsiera* (ikusi 5 irudia) algoritmoa erabiltzen da, non errorea minimizatzeko parametroen errore funtzioen deribatu partzialak erabiltzen diren. Deribatu hauek, atributu bakoitzaren errore funtzioaren malda errepresentatzen dute, eta iterazio asko eginez eta malda jarraituz errore funtzioaren minimora ailegatzen gara. *Learning rate* atributuak, iterazio bakoitzean errore funtzioaren zehar zenbat 'mugitzen' garen definitzen du, horregatik oso garrantzitsua da sarearen entrenamendurako. Informazio guztia prozesatu den ostean, errore txikiena lortzen duen pisu konbinazioarekin geratzen da, eta sarea eguneratzen du.[13]



(a) *Simple Perceptron* (b) *Multilayer Perceptron*

Fig. 6: *Perceptron*: *Simple Perceptron* (a) eta *Multilayer Perceptron* (b) eskemen arteko konparazioa.

3.2.2 Esparru Praktikoa

Eredu iragarlearen kalitatea estimatzeko ebaluazio metodo desberdinak erabili dira. Alde batetik, kalitatearen estimazioa metodo ez-zintzoa, 100 *hold-out* eta 10-*fold Cross Validation* erabiliz gauzatu da, datu errealistagoak lortzeko asmoarekin. Bestetik, datu multzoaren errepresentaziorako bi formatu desberdin probatu dira, BoW eta TD·IDF (ikusi Taulak 11 eta 12). *Baseline*arekin gertatzen den modu berdinean, bi errepresentazio espazioetan kalitate oso antzekoa lortzen dela ikusi daiteke. Nabarmena da, berriro ere, "yes"klasearentzat *Recall* eta *F-Measure* nahiko baxuak lortzen direla, beraz, ereduak errealitatearekiko benetan arrazistak diren Tweet-ak asmatzeko duen gaitasuna ez da oso ona. Nahiz eta eredu sinpleagoarekin emaitza ia berdinak lortu, hobekuntza bat espero zen *Multilayer Perceptron* eredua erabilita, askoz konplexuagoa baita.

Azkenik, bi ereduak bi fase ezberdinetatik igaroko dira. Alde batetik, sailkatzailearen inferentzia fasea daukagu, Esparru Praktikoa atalean azaltzen dena. Fase honetan, eredua entrenatu egiten da datu multzoarekin eta bere kalitatearen estimazioa egiten da, bermeak lortzeko hain zuzen. Horrela, eredua entrenatzean klasea zein izango den asmatzen saiatuko da ikasitakoa kontutan hartuz eta okertzen baldin bada zuzenketa bat egingo du. Inferentzia gauzatzeko gainbegiratutako datu multzoa erabili behar da, bitan banatzen dena, train eta dev hain zuzen, eta train multzoa entrenamendurako erabiltzen den bitartean, dev multzoa kalitatearen estimaziorako erabiliko da.

Bestalde, iragarpen fasean eredu iragarlea jada entrenatuta daukagu entrenamendu multzo gainbegiratu osoarekin. Fase honen helburua ereduak instantzia ez gainbegiratu baten klasea asmatzea da, hau da iragarpen bat egitea.

4 Esparru Esperimentala

Dokumentazioaren atal honetan, teorikoki azaldu diren ezagutzak praktikara nola moldatu diren azaltzen da.

4.1 Funtzionamendua

Proiektu hau 3 atal handietan banatuta dago: datuen aurre-prozesamendua, sailkatzaileen entrenamendua eta iragarpenak. Hurrengo atalean aurkezten diren fluxu diagramen bitartez, lanaren atalak azalduko dira.

7 .irudian programa osoaren fluxu diagrama bat aurkezten da, fluxu honetan ikusi daiteken bezala, train.csv eta test.csv fitxategiak hartuta, 8. irudiko GetRaw programari pasatzen zaizkio eta honek datuak .arff formatuan itzuliko dizkigu (train.arff eta test.arff). Emaitza hauek izanda, FSS programara sartuko ditugu, baina aurretik, lortu den train.arff TransformRaw programatik pasatuko dugu, atributu espazioa errepresentazio bektorial batera transformatzen duena.FSS programak trainInfoGain.arff eta testInforGain.arff fitxategiak bueltatuko dizkigu. Bi fitxategi hauek amaiera batean iragarpenak (Predictions programa) egiteko gordeko ditugu, baina iragarpen hauek egiteko, modelo bat sortu behar dugu.

FSS bueltatu digun trainInfoGain.arff fitxategia hartuta, GetBaselineModel, ParamOptimization eta GetModel programak exekutatuko ditugu. Lehenengo honek, kalitatearen behe bornea lortzeko *baseline* modeloa sortu du, hau da, *Logistic regression* modelo bat. Predictions programarekin eta lehen lortu ditugun trainInfoGain.arff eta testInfoGain.arff fitxategiekin, iragarpenak lortuko ditugu.

ParamOptimization programak, train fitxategia emanda parametro ekorketa egingo du, eta *hidden layers* eta *learning rate* parametro optimoak lortuko ditu gure *Multilayer Perceptron*-erako. Behin parametro optimoak lortu ditugula, GetModel programa exekutatuko dugu modeloa eta modeloaren kalitatearen estimazioa lortzeko. Modelo honekin, Predictions programa exekutatuko dugu (trainInfoGain.arff eta testInfoGain.arff fitxategiekin batera) iragarpenak lortzeko.

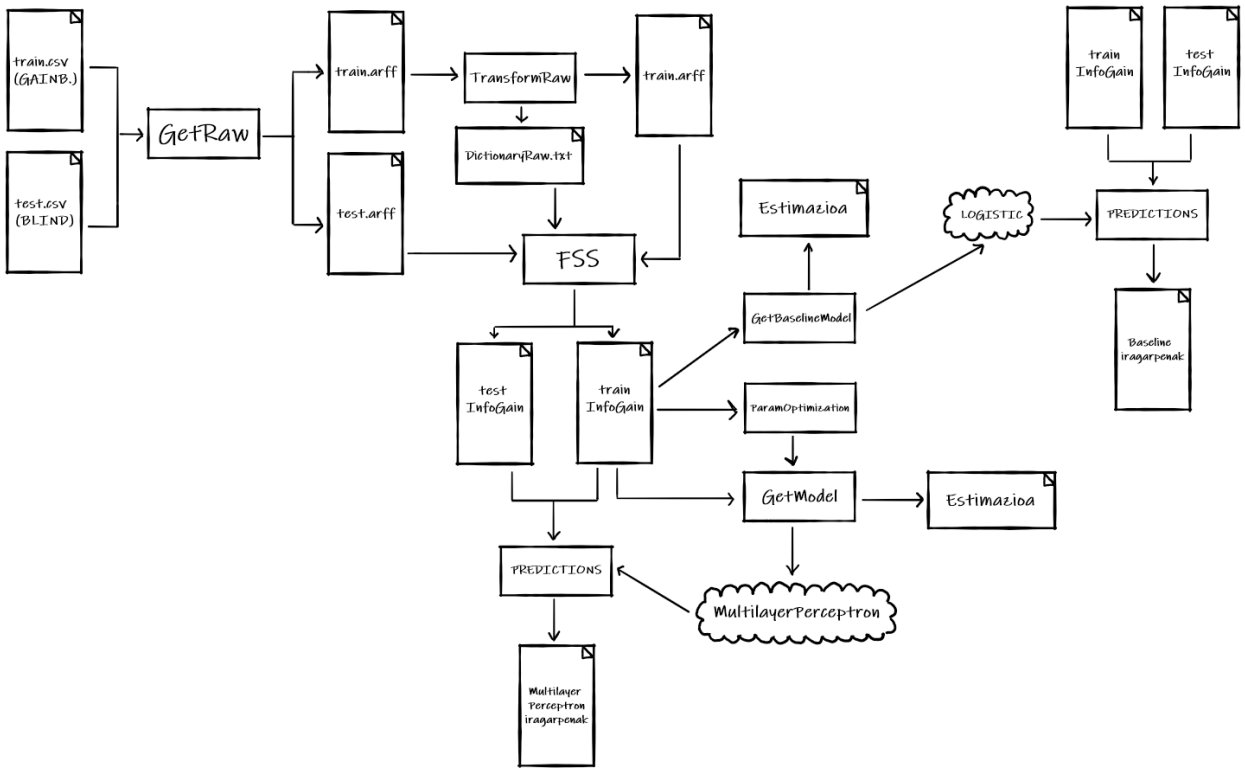


Fig. 7: Ataza guztien programaren fluxua.

4.1.1 Pre-prozesamendua

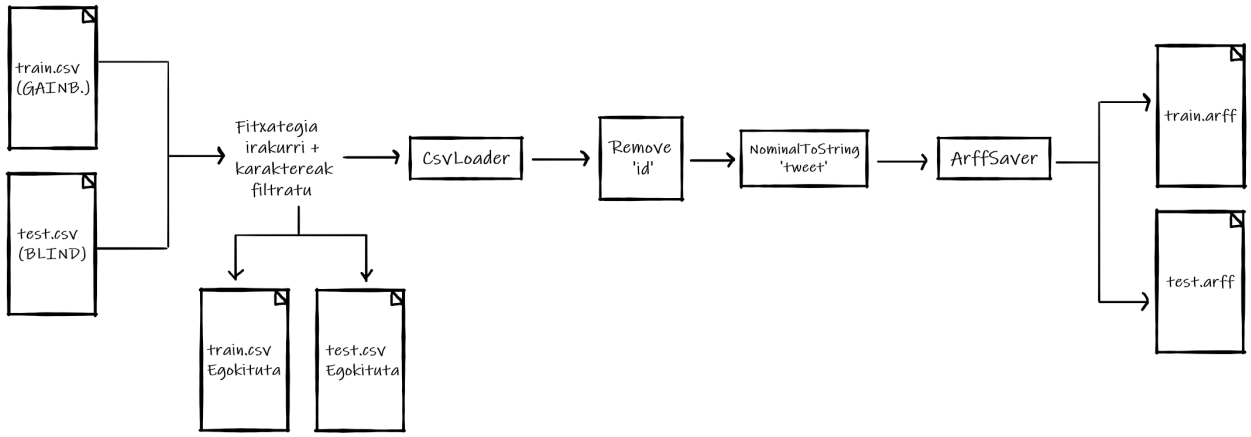


Fig. 8: GetRaw programaren fluxua.

Aurreko atalean azaldu den bezala, GetRaw programari (8. irudian agertzen dena) train eta test .csv fitxategiak pasatuko zaizkio eta honek train.arff eta test.arff fitxategiak bueltatuko ditu, baina atal honetan, programaren funtzionamendua sakonago azalduko da.

Fitxategi hauek testua dutenez atributu bezala (tweet desberdinak), testu hau egokitu behar da .arff fitxategiak sortu baino lehen. Horretarako, lehenengo instantzia bakoitzaren klasea balio numeriko batetik balio nominal batera pasatzen da (0 bada *no*-rekin ordezkatuko da eta 1 bada *yes*-ekin ordezkatuko da). Ondoren, komatxoak egokituko dira eta bukatzeko karaktere bereziak (*emoji*ak etab) ezabatuko dira.

Behin .csv fitxategiak egokitu direla, *CsvLoader* objektua erabiliz fitxategiak kargatuko dira, eta atributen egokitzapenarekin hasiko gara. Lehenengo, *id* atributua beharrezkoa ez denez (tweet-en identifikatzailea

da) *Remove* filtroa erabiliz ezabatuko da, eta ondoren, *tweet* atributua nominaletik *String*-era pasatuko da *NominalToString* filtroa erabiliz.

Programarekin bukatzeko, *ArffSaver* objektua erabiliz, egokituak izan diren bi fitxategi hauek .arff formatuan gordeko dira.

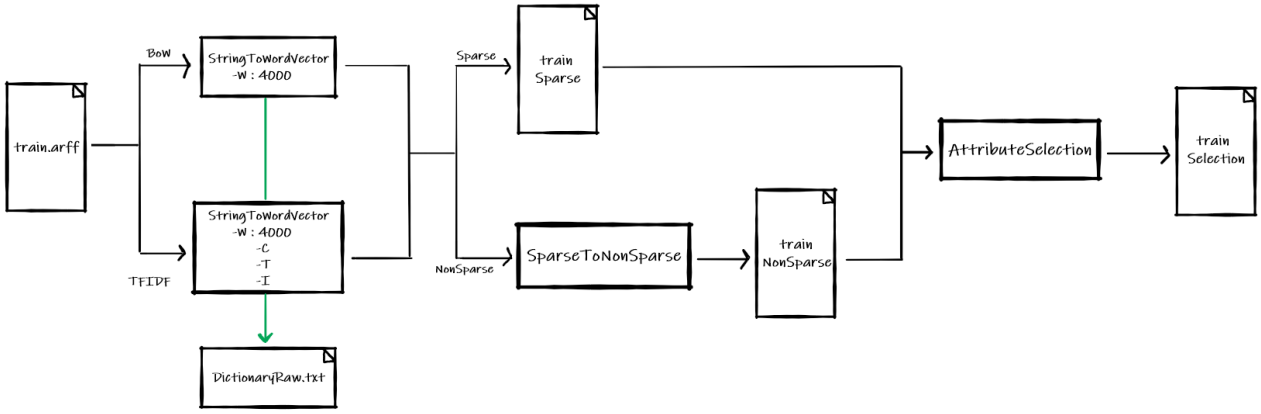


Fig. 9: TransformRaw programaren fluxua.

TransformRaw programaren kasuan, atributu espazioa errepresentazio bektorial batera transformatzen duen programa da. Malgutasun handia izateko, errepresentazio desberdinekin probatzen da (BoW eta TF·IDF adibidez). Atal honetan programa honen fluxu diagrama aztertzen da.

Exekutagarri honek, .arff fitxategi bat sarrera bezala dauka, eta erabiltzaileak erabakiko du zein errepresentazio erabiliko den (BoW edo TF·IDF). Aukeratutako filtroa aplikatuko da eta emaitzak gorde baino lehen, nola gordeko diren aukeratu definitu behar da (Sparse edo NonSparse). Bigarren filtro hau aplikatu ostean, *AttributeSelection* filtroa aplikatzen da atributu garrantzitsuenak aukeratzeko eta datuak gordeko dira programari pasatu zaion *path*ean.

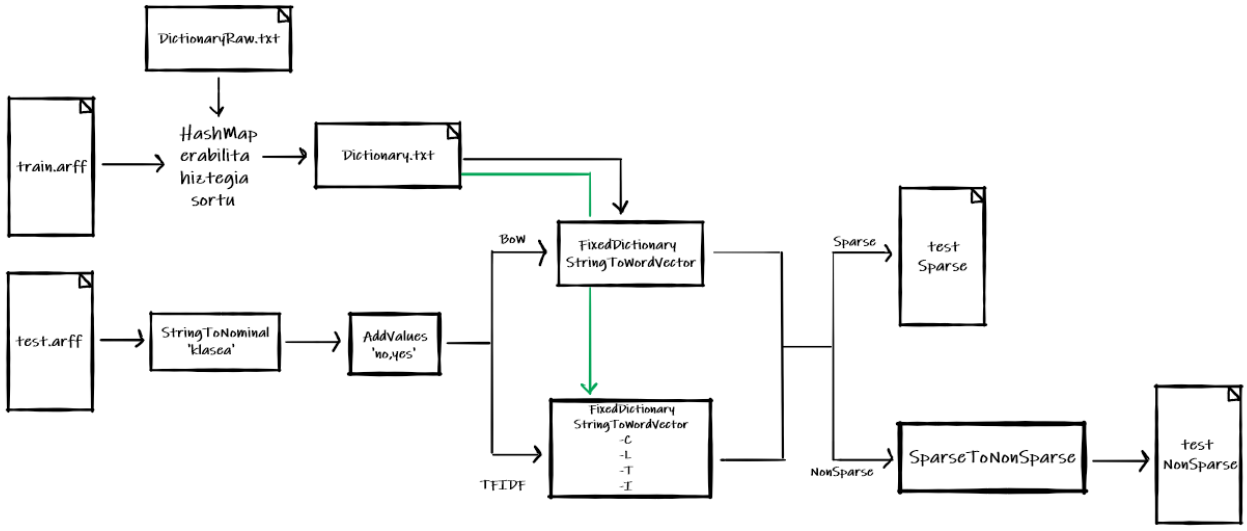


Fig. 10: MakeCompatible programaren fluxua.

Datuen aurre-prozesamenduarekin bukatzeko, MakeCompatible programaren funtzionamendua aztertuko da. Programa honek, ebaluazio multzo bat (test.arff adibidez) entrenamendu multzoan (train.arff) ezarritako espazioan errepresentatzeko erabiltzen den softwarea da (BoW nahiz TF·IDF eta Sparse nahiz NonSparse).

Programa honi train.arff eta test.arff fitxategiak pasatuko dizkiogu. Ebaluazio multzoaren kasuan (test.arff),

StringToNominal filtroa aplikatzen zaio, eta ondoren, klase bakoitzeko instantzia kopurua proportzionala izateko, *AddValue* filtroa erabiliz instantziak gehitzen dira. Entrenamendu multzorako, 9.irudiko TransformRaw programatik ateratzen den hiztegiarekin eta *HashMap* baten laguntzarekin hiztegi bat sortzen da. Ebaluazio multzoa nola errepresentatuko den erabakitzen den ondoren (BoW edo TF·IDF), train.arff multzoarekin lortutako hiztegia erabiliko da eta datuak gordeko dira (Sparse edo NonSparse moduan).

4.1.2 Sailkatzailearen Inferentzia

Sailkatzailearen inferentziaren atalaren barruan, lau programa ditugu: FSS, GetBaselineModel, ParamOptimization eta GetModel.

Hauetako lehenengo exekutagarriaren kasuan, bi zeregin ditu, alde batetik entrenamendu multzoko atributuak hautatu, eta bestetik ebaluazio multzoa train multzoarekiko bateragarria bihurtzea.

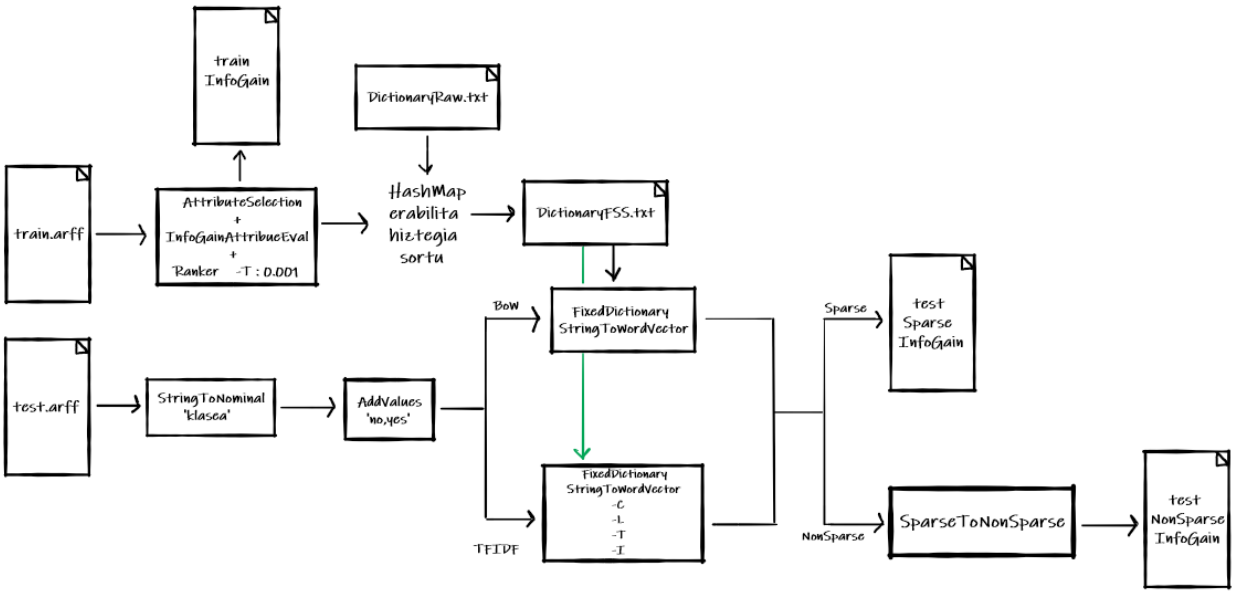


Fig. 11: FSS programaren fluxua.

10. irudiko programaren kasuan bezala, hasieran ebaluazio multzoari *StringToNominal* eta *AddValues* filtroak aplikatuko zaizkio. Entrenamendu multzoaren kasuan, hasteko *AttributeSelection* filtroa aplikatuko zaio egokienak diren atributuak aukeratzeko eta *trainInfoGain.arff* bezala gordetzeko. Ondoren, MakeCompatible programan bezala TransformRaw exekutagarriak itzultzen duen hiztegia eta *HashMap* bat erabilita, hiztegi bat sortuko du. Hiztegi honekin ebaluazio multzoko datuak errepresentatuko dira (BoW edo TF·IDF moduan) eta Sparse edo NonSparse bezala gordeko dira.

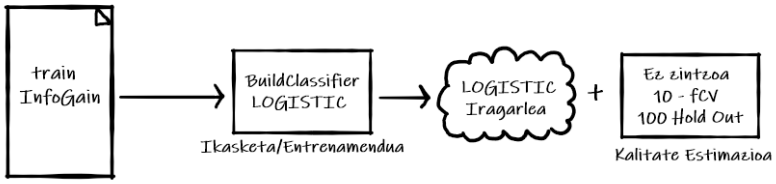


Fig. 12: GetBaselineModel programaren fluxua.

GetBaselineModel exekutagarriak *baseline* modeloa sortuko du, *Logistic Regression* gure kasuan, eta modelo honen kalitate estimazioa kalkulatuko eta gordeko du.

12.irudian ikus daiteken moduan, programa honi FSS exekutagarriak bueltatzen duen trainInfoGain entrenamendu multzoa pasatuko diogu eta datu hauekin modeloa sortuko du. Behin modeloa sortu dela, 3

ebaluazio metodo desberdin kalkulatuko dira: ebaluazio ez-zintzoa, 10-*fold cross validation* eta *Hold-Out* 100 aldiz. Lortutako emaitzak programari parametro bezala pasatu zaion helbidean gordeko dira.

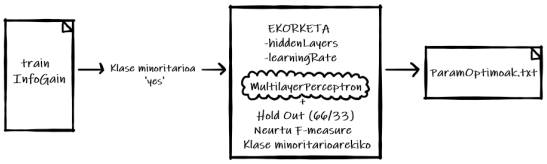


Fig. 13: ParamOptimization programaren fluxua.

Gure sare neuronala sortu baino lehen, modelo honetarako optimoenak diren parametroak eta hauen konfigurazio optimoa topatu behar dugu. Hau lortzeko, ParamOptimization programa sortu da, parametro ekorketa teknika erabiltzen duena.

Programa honi, GetBaselineModeli bezala, trainInfoGain entrenamendu multzoa pasatuko diogu eta honek gure modelorako parametro optimoen balioak gordeko ditu guk ezartzen dugun helbidean.

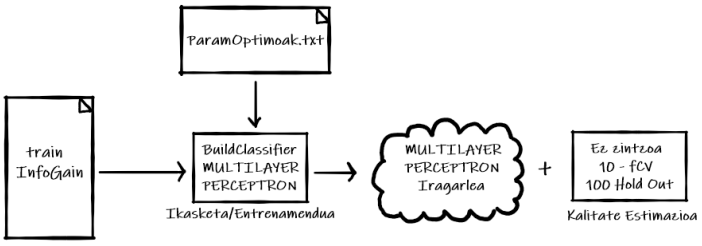


Fig. 14: GetModel programaren fluxua.

Sailkatzailearen inferentziaren atalarekin bukatzeko, GetModel programaren funtzionamendua azalduko da. Programa honek *MultilayerPerceptron* baten modeloa eraikiko du eta bere kalitatearen estimazioa lortuko du.

ParamOptimization programak kalkulatzen dituen atributuen balio optimoak exekutagarri honi pasatuko dizkiogu, eta hauekin modeloa sortuko du. Behin modeloa sortu dela, GetModelBaseline programak erabiltzen dituen ebaluazio sistema berdinak erabilita, modeloaren kalitatearen estimazioa lortu du.

4.1.3 Iragarpenak

Programaren funtzionamenduaren azalpenekin bukatzeko, proiektuaren hirugarren eta azken atala azalduko da.

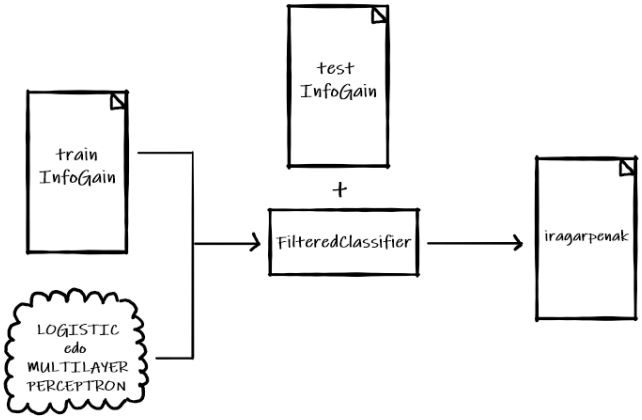


Fig. 15: Predictions programaren fluxua.

Exekutagarri honen helburua dokumentu bat (ala sorta bat) sailkatzea da, emandako eredu iragarle baten bitartez. Sarrera parametro bezala trainInfoGain entrenamendu multzoa eta iragarpenak egiteko modeloa izango ditu, eta bi hauekin iragarpenak lortuko ditu. Iragarpen hauek hirugarren sarrera parametro bezala pasatzen zaion helbidean gordeko ditu. Iragarpenak atalean azalduko den moduan, *FilteredClassifier* filtroa ez da erabiliko gure eredu iragarlearekin batera denboraren kostua asko igotzen delako.

4.2 Emaitza Nabarmenenak

4.2.1 Aurre-prozesamendua

Datu multzoak BoW eta TD·IDF errepresentazioak erabilita karakterizatu dira. Datu multzoen tamaina kontutan hartuta, bi errepresentazio horietan lortutako atributu kopurua oso altua da. Horrela izanik, eredu iragarlea entrenatzean *underfitting* eta datu zaratatsuak ekiditeko informazio galera ahalik eta txikiena suposatuko duen atributu espazioaren murrizketa gauzatu da.

Atributu espazio hoberena esperimentalki hautatu da, *Feature Subset Selection* edo *FSS* teknikak erabiliz, atributu erredundanteak kentzeko. Honi esker, sailkapen prozesua azkartzea lortu da. Atributu espazioa murrizteko WEKA-ko *AttributeSelection* filtro gainbegiratua erabili da entrenamendu multzoan, irizpidea informazio irabazia maximizatzea izanik *InfoGainAttributeEval* aplikatuz.

Gainera, atributu azpimultzoa bilatzeko *Ranker* erabili da, atributuak ordenatzen dituenak banakako ebaluazioen arabera. *Ranker*-ak bi parametro ezartzeko aukera ematen du, *numToSelect* eta *threshold*.

- *numToSelect*: Mantendu nahi den atributu kopurua adierazten du, defektuz -1 dago jarrita eta denak mantendu nahi direla esan nahi du. Parametro hau defektuz dagoen moduan mantentzea erabaki da, ahalik eta atributu gehien mantentzeko eta bigarren parametroaren bitartez soilik murrizteko espazioa.
- *threshold*: Atributu bat ez baztertzeko izan behar duen balio minimoa *InfoGainAttributeEval*-ekiko adierazten du. Defektuz -1.7976931348623157E308 ezartzen da, Javako *Long integer* txikiena. Balio optimoena esperimentalki kalkulatu da (ikusi Taula 5), eta, atributu kopurua kontutan hartuta, 0.001 izatea erabaki da, atributu gehiegi erabiltzean konputazio denbora asko hazten baita eredua entrenatzerakoan. (Ikusi Taula 14 eta Taula 15)

-T	AttributeSelection	Hasierako Kopurua
0.1	0	1661
0.01	4	1661
0.001	146	1661
0.0001	1040	1661

Taula 5: *Threshold*, -T, parametroarekin egindako probak.

4.2.2 Ereduaren Parametroak

Multilayer Perceptron eredua entrenatuko da iragarpenak egiteko. Algoritmo honetan parametro desberdinak erabiltzea posiblea da, askoren artean *learning rate*, *hidden layer* kopurua edo *epoch* kopurua. Hori horrela izanik, parametro ekorketa egitea behar beharrezkoa da eredua optimoena egiten dituzten parametroak lortzeko. Parametro ekorketa gauzatzeko *hold-out* ebaluazio eskema erabili da, train multzoko datuen %33-a dev multzo bezala erabiltzeko baztertuz. Ondoren, parametroak hautatzeko irizpide bezala klase minoritoarioarekiko, "yes", *F-Measure* hautatu da. Horrek ereduak arrazista/sexistak diren Tweet-ak ondo sailkatzeko duen zehaztasun optimoena emango digu, gure datu multzorako benetan adierazgarria izango dena.

- Parametro Ekorketa: Lehen aipatu bezala, klase minoritarioarekiko *F-Measure* optimoena lortzen dituzten parametroak ekortuko dira. Gure kasuan, kostu konputazionala eta eskuragarri zegoen denbora mugatua zela kontutan izanda, hurrengo bi parametroen ekorketa gauzatzea erabaki da:

– *Learning rate*: Esleitutako algoritmoaren esparru teorikoan azaltzen den modua, *Multilayer Perceptron* algoritmoak *backpropagation* teknika erabiltzen du atributuen pisuak finkatzeko, eta Weka-k eskaintzen dituen parametro desberdinen artean, *Learning rate*-a kalkulo honekin zer-ikusia duen bakarra da. Parametro hau 0 eta 1 artean finkatuta dago, eta defektuz 0.1 izaten da programazio pakete askotan (Weka barne), horregatik, hurrengo *Learning rate* parametroaren konfigurazio desberdinak probatu dira (ikusi Taula 6).

Learning rate balioa
0.1
0.2
0.3
0.4

Taula 6: Aztertu diren *Learning rate* desberdinen balioak.

– *Hidden layers*: Parametro hau algoritmoaren garrantzitsuena dela esan daiteke, *Multilayer Perceptron* bat *Simple Perceptron* batetik desberdintzen duena delako, ezkutuko geruza definitzen baitu. Ez dago metodo zehatz bat tarteko geruza honen topologia definitzeko, baina gehienetan ez da beharrezkoa *input* kopurua baino neurona gehiago dituen kapa bat erabiltzea. Hau dela eta, aztertu diren konfigurazio geruzak zoriz aukeratuak izan dira, azaldutako jarraibidea kontuan izanik(ikusi Taula 7).

Hidden Layers balioa
50,25,12
100
150,100,50
90,45

Taula 7: Aztertu diren *Hidden layers* desberdinen balioak.

Bi parametro hauekin parametro ekorketa eginda, 16 konbinazio posible sortzen dira, eta haien artean hauek dira datu sorta bakoitzerako optimoak izan direnak (Ikusi Taula 8). Espero zen moduan errepresentazio bereko Sparse eta NonSparse moduan emaitza berdinak lortu dira, baina BoW eta TF·IDF artean aldaketa bat ikusten da bi parametroen balio optimoetan. Azkenik, TF·IDF errepresentazioan klase minoritarioarekiko *F-Measure* hobeagoa lortzen dela ikus daiteke, nahiz eta ezberdintasuna oso txikia izan.

	BowSparse	BowNonSparse	TFSparse	TFNonSparse
Hidden layer konfigurazio optimoa	150,100,50	150,100,50	90,45	90,45
Learning rate konfigurazio optimoa	0.1	0.1	0.2	0.2
Lortutako *F-Measure*	0.5712	0.5712	0.5951	0.5951

Taula 8: Parametro ekorketa: *hidden layers* eta *learning rate* optimoak datu multzo bakoitzerako.

- *Experimental Setup*: Atazan zehar bi datu errepresentazio mota erabili dira, BoW eta TF·IDF. Esperimentazio sakonago baten bitartez errepresentazio mota egokiena aukeratzea lortu daiteke. Horretarako, parametro ekorketa bi espazioekin gauzatu da, eta klase minoritarioarekiko *F-Measure* hoberena 0.586 eman digu, *Logistic Regression* TF·IDF-koa hain zuzen (Ikusi Taulak 9 eta 10). Hori dela eta, TF·IDF errepresentazio hobea da datuen aldea mespretxagarria izan arren.

  Parametro ekorketa *InfoGain* erabiliz egin dugu. *AttributeSelection* erabili lehenengo metodo bezala baina kostu konputazional oso altua zuenez aldatu behar izan genuen (ikusi Taula 14) eta *InfoGain* aplikatu denbora aldetik bideragarria ez zelako. Klase minoritarioarekiko *F-Measure* onena lortzeko: BoW, 146 atributu erabili dira; TF·IDF, 151 atributu erabili dira.

Gainera, bi atributu espazioetan lortutako parametroen balio optimoak ikusita badakigu bi errepresentazio moten artean, berriro ere, Sparse edo NonSparse erabiltzeak berdin duela eta emaitzak berdinak itzuliko dituela. Biek *Hidden Layers* eta *Learning rate* desberdinak ematen dituzte (Ikusi Taula 8).

Azkenik, *baseline*ak bi espazio horietan lortutako klase minoritarioarekiko *F-Measure*a *Multilayer Perceptron*ekin konparatu dira. Errepresentazio espazioaren aukeraketaren emaitz berdinak ditugu, *Logistic Regression* jarduera hobeagoa dauka, baina, berriro ere, aldea mespretxagarria da (ikusi Taulak 9 eta 10).

		Logistic Regression			Multilayer Perceptron		
	Klasea	**Precision**	**Recall**	**F-Measure**	**Precision**	**Recall**	**F-Measure**
Hold out (%66 Split)	no	0.957	0.993	0.975	0,959	0,996	0,977
	yes	0.836	0.446	0.581	0,884	0,432	0,581
	W. Avg	0.949	0.953	0.946	0,954	0,956	0,949

Taula 9: *F-Measure* klase minoritarioarekiko *Logistic Regression* eta *Multilayer Perceptron*en parametro optimoekin erabiliz. BoW errepresentazioa.

		Logistic Regression			Multilayer Perceptron		
	Klasea	**Precision**	**Recall**	**F-Measure**	**Precision**	**Recall**	**F-Measure**
Hold out (%66 Split)	no	0.958	0.993	0.975	0,960	0,991	0,976
	yes	0.841	0.449	0.586	0,803	0,459	0,584
	W. Avg	0.949	0.953	0.946	0,949	0,954	0,948

Taula 10: *F-Measure* klase minoritarioarekiko *Logistic Regression* eta *Multilayer Perceptron*en parametro optimoekin erabiliz. TFIDF errepresentazioa.

4.2.3 Itxarondako kalitatea

Parametro ekorketaren bitartez *Multilayer Perceptron* ereduaren atributuentzat parametro optimoak lortu dira. Hurrengo pausua, beraz, ereduak eskainiko duen kalitatearen estimazioa lortzea da. Estimazioa gauzatzeko hiru ebaluazio eskema desberdin erabili dira, ez-zintzoa, 100 *hold-out* eta 10-*fold Cross Validation*. Gainera, *baseline*arekin egin den modu berdinean, BoW (ikusi Taula 11) eta TF·IDF (ikusi Taula 12) errepresentazio espazioekin egin da esperimentazioa, gerora gure atazarako egokiagoa izango dena aukeratzen lagunduko diguna.

Ereduek emandako emaitzak koherenteak dira, behe-bornea goi-bornea baino txikiagoa da, baina diferentzia mespretxagarria da. Eta gainera, klase minoritarioaren *Recall*-ean ohartzen bagara ikusi dezakegu nola, alde batetik, eredu iragarleek ez dutela errealitatearekiko 'yes' diren instantziak ondo sailkatzen, eta bestetik, hiru ebaluazio eskemekiko emaitzak ez direla asko bereizten (adb. *Recall*-a 'yes' klase baliorako 100 *Hold-Out*-ekin eta BoW errepresentazioarekin 0.419 da *Logistic Regression* erabiliz eta 0.432 *Multilayer Perceptron* erabiliz) (Ikusi Taulak 3 eta 11). *Multilayer Perceptron* erabiltzea ez du batere merezi itxarondako kalitatea ez delako egokia entrenatzeko behar duen denbora eta emaitzak konparatuz (Ikusi Taula 15).

4.2.4 Iragarpenak

Atazako azken zatian eredu iragarleari test multzo ez gainbegiratu bat emanda klaseen iragarpenak egiten dira. Askotan, sailkatzailea aplikatu aurretik filtratze atal bat egoten da. Horretarako, Wekako *FilteredClassifier* erabili daiteke, filtroak egiten duen gauza bakarra sailkatzailea exekutatzea da beste filtro batzuetatik pasatu den datu multzoarekin. Honi esker, train eta test multzoak bateragarriak baldin ez badira, 'bateragarri' bihurtuko ditu atributuen pisuak erabilita.

Haatik, filtro hau funtzionatzeko sailkatzailea berriro entrenatzea beharrezkoa da. Gure kasuan, *Logistic Regression* erabilita ez dago arazo handirik pare bat minutu behar dituelako soilik, baina *Multilayer Perceptron* eredua erabili nahi izatekotan sailkatzailea eraikitzeko denbora altuegia da (ikusi Taula 15) soilik iragarpenak

		Multilayer Perceptron		
	Klasea	**Precision**	**Recall**	**F-Measure**
Ez Zintzoa	no	0.961	0.997	0.978
	yes	0.913	0.461	0.613
	W. Avg	0.957	0.959	0.953
	Accuracy	%95.9108		
100 Hold-Out	no	0.959	0.996	0.977
	yes	0.884	0.432	0.581
	W. Avg	0.954	0.956	0.949
	Accuracy	%95.62		
10 - fCV	no	0,957	0,991	0,973
	yes	0,765	0,408	0,532
	W.Avg	0,943	0,950	0,942
	Accuracy	%94.969		

Taula 11: *Multilayer Perceptron* jarduera BoW.

		Multilayer Perceptron		
	Klasea	**Precision**	**Recall**	**F-Measure**
Ez Zintzoa	no	0.962	0.991	0.976
	yes	0.794	0.479	0.597
	W. Avg	0.95	0.955	0.949
	Accuracy	%95.4727		
100 Hold-Out	no	0.96	0.991	0.976
	yes	0.803	0.459	0.584
	W. Avg	0.949	0.954	0.948
	Accuracy	%95.4114		
10 - fCV	no	0,957	0,992	0,974
	yes	0,795	0,404	0,535
	W.Avg	0,945	0,951	0,943
	Accuracy	%95.0848		

Taula 12: *Multilayer Perceptron* jarduera TD·IDF.

egiteko gero. Bezero bati gure proiektua saldu nahi izatekotan, bideraezina izango litzateke orduak behar izatea iragarpenak egiteko, eta hori dela eta filtroa ez erabiltzea erabaki da. Azkenik, gure atazaren ebazpena kontutan hartuta, beti bateragarriak izango diren train eta test multzoak sortzeaz ziurtatu gara iragarpenak egiterakoan arazorik ez izateko.

4.3 Konparazioa

4.3.1 Atributu Hautapena

Atazarako erabiltzen den entrenamendu multzoak instantzia kopuru oso altua dauka, 31962 hain zuzen. Instantzia bakoitzeko Tweet bat dago, 20-40 hitz artekoa izan daitekeena gutxi gorabehera. Beraz, hitzak ez errepikatzekotan, gehienez 958 860 hitz aurkitu daitezkeela estimatu daiteke. Gutxi gorabehera, 429503 hitz daudela neurtu da *online*[1] programa baten bidez, eta horietatik 97215 hitz desberdin daudela ikusi da. Datu multzoaren errepresentazio espazioa aldatzean, hau da, BoW zein TD·IDF espazioetara aldatzean, hitz horietako bakoitza atributu bat bilakatzen da, atributu espazio izugarri handia sortuz. Horrela izanik, atributu espazio hain handia duen datu multzoak erabiltzeak kostu konputazionala gehiegi igoko luke, exekuzio denbora ez errealistak lortuz.

Hasteko, errepresentazio espazioa aldatzean datuetatik lortzen den atributu kopurua murriztea erabaki da. Horretarako, *StringToWordVector* filtroak -W (*number of words to keep*) parametroa ezartzeko aukera ematen du. Parametroak erabiliko diren hitz kopurua zehazten du, baina metodo ez gainbegiratu bat erabiltzen du, ausaz

[1] `https://www.anycount.com/free-online-word-count-tool/`

aukeratzen ditu hitz horiek, alegia. Defektuz 1000 hitz hartzen ditu, baina proba desberdinak egin ostean (ikusi Taula 13) 4000 hitz mantentzea erabaki da, *AttributeSelection* filtroa aplikatzeko behar den denbora eta Java memoriaren gaitasuna kontutan hartuta. Gure helburua hemen ahalik eta atributu kopuru gehien mantentzea izan da, gero atributu horietatik esanguratsuenak aukeratu ahal izateko.

-W	AttributeSelection Denbora
1000	1 min, 43 sec, 907 ms
2000	3 min, 3 sec, 932 ms
3000	6 min, 46 sec, 874ms
4000	8 min, 32 sec, 549ms
5000	OUT OF MEMORY

Taula 13: *Number of words to keep*, -W, parametroarekin egindako probak.

Ondoren, Aurre-prozesamendua atalean azaldu den moduan, *AttributeSelection* filtro gainbegiratua erabili da entrenamendu multzoan, irizpidea informazio irabazia maximizatzea izanik. Hau da, ditugun 3802 atributuetatik informazio gehien emango dituztenak soilik mantenduko dira, 146 BoW errepresentazioan eta 151 TD·IDF errepresentazioan. Atributu kopuru murrizketa oso nabarmena da, baina gehiago mantentzekotan kostu konputazionala denbora aldetik asko igoko litzateke sailkatzailea entrenatzerako momentuan. Taula 14n ikusten den moduan *InfoGain* aplikatuta duten datu multzoen denbora askoz baxuagoa da (4. iteraziotik aurrera programa etetea erabaki genuen denbora gehiegi behar izango zuela estimatu genuelako). Era berean, TD·IDF errepresentazio espazioko denbora pixka bat baxuagoa da, baina desberdintasuna ez da oso nabarmena eta beraz ia berdinak direla esan dezakegu. Azkenik, aipatu beharra dago denborak asko aldatzen direla kalkuluak egiteko erabiltzen den hardwarearen gaitasunen arabera (ordenagailu batetik bestera denborak bikoiztu daitezke).

	BoW		**TDIDF**	
Iterazioa	**InfoGain (146)**	**AttributeSelection (247)**	**InfoGain (151)**	**AttributeSelection (266)**
1	0h, 11min, 11sec	0h, 31min, 54sec	0h, 10min, 32sec	0h, 27min, 32sec
2	0h, 19min, 11sec	0h, 53min, 42sec	0h, 16min, 58sec	0h, 51min, 6sec
3	0h, 54min, 20sec	2h, 53min, 21sec	0h, 50min, 37sec	2h, 37min, 54sec
4	0h, 23min, 24sec	1h, 9min, 37sec	0h, 22min, 2sec	0h, 57min, 33sec
5	0h, 11min, 30sec	-	0h, 10min, 47sec	-
6	0h, 20min, 24sec	-	0h, 17min, 45sec	-
7	0h, 57min, 9sec	-	0h, 54min, 39sec	-
8	0h, 31min, 2sec	-	0h, 31min, 2sec	-
9	0h, 12min, 22sec	-	0h, 9min, 23sec	-
10	0h, 21min, 21sec	-	0h, 16min, 7sec	-
11	0h, 58min, 26sec	-	0h, 54min, 57sec	-
12	0h, 31min, 13sec	-	0h, 27min, 22sec	-
13	0h, 13min, 32sec	-	0h, 11min, 49sec	-
14	0h, 22min, 9sec	-	0h, 20min, 4sec	-
15	0h, 58min, 18sec	-	0h, 55min, 1sec	-
16	0h 31min, 8sec	-	0h 28min, 45sec	-
Guztira	**7h, 57min, 23sec**	**22h aprox.**	**7h, 33min, 50sec**	**20h aprox.**

Taula 14: Parametro ekorketa egitean *InfoGain* aplikatuta eta *AttributeSelection* soilik aplikatuta iterazio bakoitzerako beharrezkoa den denbora.

Datuak ikusita erabilitako atributu kopurua igotzeak entrenamendurako kalkulu denbora gure atazarako bideraezina izatea suposatzen du, proba esperimentalak ezin baitira bukatu denbora epe egingarri batean. Hori dela eta, atributuen hautapena gauzatzea behar beharrezkoa dela ondorioztatzen dugu, alde batetik informazio gehigarririk ematen ez duten atributuak mantentzean *underfitting* bezalako arazoak ekiditeko, eta bestetik kostu konputazionala asko igotzen delako.

Haatik, atributu kopurua hainbeste murrizteak eredu iragarlearen jardueran eragin oso nabaria izango du.

Gure datu multzoa kontutan izanda, benetan arrazistak diren Tweet-ak detektatzea da interesatzen zaiguna, baina horretarako zenbat eta atributu gehiago izan hobeto. Bestela, oso hitz gutxi erabilita oso zaila izango da ereduarentzat iragarpen egoki bat egitea. Atazan erabilitako bi ereduen jarduerak aztertuz (ikusi Taula 3, Taula 4, Taula 11 eta Taula 12) oso nabarmena da Tweet arrazistak detektatzerakoan errore asko egiten dituztela bi ereduek.

4.3.2 Atributu Espazioak

Esperimentuak egin ondoren esan dezakegu BoW eta TF·IDF artean ez dagoela alde esanguratsurik gure atazan eta datu sortan oinarrituta. Denboraren aldetik BoW exekuzio denbora txikiagoak ematen ditu ereduak eraikitzean eta kalitatea estimatzerakoan (Ikusi Taula 14 eta Taula 15) mespretxagarriak izan arren. Egia da, TF·IDF-k klase minoritarioarekiko *Recall* hobea lortzen duela, baina berriro ere alde infinitesimal batekin.

Bestalde, bi atributu espazioetako datuak *Sparse* eta *NonSparse* bezala ere landu dira, normalean eredu iragarleak *NonSparse* datuekin hobeto funtzionatzen baitute. Baina, emaitzak ikusita ez da ezberdintasun nabarmenik ikusi, ez exekuzio denboran ezta ereduen jardueran (hori dela eta datu guztiak ematerakoan ez da desberdinketarik egin *Sparse* zein *NonSparse* artean).

4.3.3 Ereduak

Logistic Regression eta *Multilayer Perceptron* ereduak konparatzerako orduan, esperimentuen emaitzak eztabaidaezinak dira, ataza honetarako *Multilayer Perceptron* ereduaren kostu konputazionalak eta denbora kostuak ez du batere merezi eredu sinpleagoarekin konparatuz (Ikusi Taula 15). Gainera, *Logistic Regression* eredu sinpleagoa izanik behe-borne bezala ezarrita ikus dezakegu *Multilayer Perceptron*-aren jarduera ez dela eredu sinpleagoarena baino askoz hobea, kasu batzuetan txarragoa izanik (Konparatu Taulak 3 eta 11 eta Taulak 4 eta 12).

Horrela izanda, eredu konplexuaren kostu konputazionala eta denbora kostuak lortzen den jarduera hobekuntzarekin alderatuz *Multilayer Perceptron* erabiltzeak pena merezi ez duela ondoriozta daiteke, batez ere eredu sinplearen jarduera oso antzekoa delako, denbora kostuak alde batera utziz.

	Logistic Regression		**Multilayer Perceptron**	
	BoW	**TDIDF**	**BoW**	**TDIDF**
Eredua Eraiki	0min, 19sec, 238ms	0min, 20sec, 32ms	3h, 2min, 15sec	3h, 12min, 44sec
Ez Zintzoa	0min, 3sec, 372ms	0min, 3sec, 562ms	0h, 0min, 54sec	0h, 0min, 56sec
100 Hold-out	0min, 14sec, 32ms	0min, 14sec, 722ms	0h, 1min, 39sec	0h, 1min, 4sec
10-fCV	1min, 5sec, 538ms	1min, 7sec, 36ms	28h, 14min, 42sec	11h, 39min, 10sec
Guztira	**1min, 42sec, 180ms**	**1min, 44sec, 280ms**	**31h, 19min, 30sec**	**14h, 53min ,54sec**

Taula 15: Ereduak eraikitzearen eta kalitatea estimatzearen denborak bi atributu espazioetan.

4.4 Diskusioa

Esan dezakegu lortutako emaitzak eztabaidaezinak direla, esperimentu nahiko egin arren, egia da esperimentu gehiago egitea gustatuko litzaiguke, eta are gehiago, *Multilayer Perceptron*-a hobetzea parametroen ekorpena hobetuz. Lehengo ekorketa batean 248 iterazio egitea gustatuko litzaiguke baina honek 108,5 ordu suposatzen zituen, orduan 16 iteraziotara (7 ordu) gutxitu behar izan genuen.

Garatutako softwarea ez da batere malgua, izan ere, *ad-hoc* eginda dago gure datu multzorako. Hala ere, datu multzo antzekoekin lan egiteko aproposa izan daiteke (adibidez, mezuak, *SPAM/HAM*; sentimenduak, *POSITIBO/NEGATIBO*; *Stack Overflow* galderak, *ERABILGARRI/EZ ERABILGARRI*; etab.). Gure modeloak hoberen sailkatzen dituen datu multzoak bi klase dituztenak dira, bi eredu iragarleen izaeragatik (*Logistic Regression* eta *Multilayer Perceptron*). Beraz, eredua gure datu multzoaren antzekoak diren datuekin entrenatzeko,

sailkatzeko eta iragarpenak egiteko gai izan beharko litzateke datuen egokitzapena aukera espazio zabalago batera bideratuz.

4.5 Proba Gehigarriak

Multilayer Perceptron eredu iragarleak sortzen dituen murriztapenak ikusita, bai denbora aldetik eta baita kostu konputazional aldetik, *Logistic Regression* eredua erabilita proba gehigarri batzuk egitea erabaki da. Proba hauen helburua ereduaren jardueran atributuek duten eragina aztertzea da. Horrela, atributu kopuruak eta atributuen korrelazioak ereduaren iragarpen ahalmenean zein eragin izango duen ikusi nahi da.

Probak egiteko atributu espazio bakarra erabili da, TF·IDF, aurreko probetan emaitza hoberenak lortzen dituena hain zuzen. Atazaren helburua Tweet-ek gorroto sentimenduak dituzten ala ez erabakitzea da, eta hori gauzatu ahal izateko eredua atributu kopuru oso txikiarekin entrenatzen bada, zailtasun ugari izango ditu gorroto kutsua dutenak identifikatzeko, oso hitz gutxirekin egiten duelako lan. Era berean, erabiltzen diren hitzak egokiak izan behar dira, hau da, informazio asko eskaini behar dute nahiz eta gutxi izan.

Baldintza horiek kontutan izanda, eredua entrenatzeko hitz kopuru ezberdinekin egin dira probak, eta baita *AttributeSelection* filtroa aplikatuz *InfoGain* irizpidearekin (ikusi Taula 16). Lortu ditugun emaitzak hainbat dira, adibidez, atributu kopuru oso handia erabiltzeak ez du emaitzak hobetzen, are gehiago esanda, hautapen edo bazterpen bat jasan ez duten atributu sortak erabiltzeak ereduaren jarduera oztopatzen du (ikusi Taula 16), InfoGain gabeko 10000 atributuk InfoGain duten 151 atributuk baino emaitzak txarragoak lortzen ditu, baita denborarekiko ere (Fig. 16). Hori gertatzen da tweet-ekin lan egiten ari garelako eta hitzak guztiz aldatu egiten dira hashtag(#) batekin idaztean, ez du pisu berdina *love* edo #*love* hitzak. 10000 atributuko espazioari atributu hautapena egin ondoren eta 151 atributuekin lan egitean, espazio hori definitzen duen hiztegia begiratuz ikusi dezakegu gutxi gorabehera hitzen %47 hashtag(#) duten hitzak direla (80 hitz hashtag gabe eta 71 hastag-rekin).

Erabilitako Ranker-aren *threshold*-a mugituz InfoGain bitartez aukeratutako atributu gehiago edo gutxiago baztertzen ditu. Ordenagailuaren konputazio ahalmena dela eta 15000 atributuko espazioa lortu dugu maximo bezala eta hiru *threshold* desberdinekin 3 froga egin ditugu. Azkenean ikusi dugu *threshold* bezala 0.0001 erabiltzetik 0.00001 erabiltzera hobekuntza minimoa dela, orduan, momentu horretan hobekuntza kurba lautzen da (Fig. 16).

	Logistic Regression					
10-fCV 'yes'	**3000 Atrib. No InfoGain**	**10000 Atrib. No InfoGain**	**10000 Atrib. InfoGain (151)**	**15000 Atrib. InfoGain (170)**	**15000 Atrib. InfoGain (3817)**	**15000 Atrib. InfoGain (3886)**
Precision	0.496	0.406	0.838	0.834	0.81	0.813
Recall	0.697	0.655	0.541	0.436	0.631	0.639
F-Measure	0.579	0.501	0.658	0.573	0.709	0.716
Accuracy	%92.9009	%90.8548	%96.0516	%95.4352	%96.3738	% 96.4364

Taula 16: *Logistic Regression* eredu iragarlearekin egindako proba ezberdinen emaitzak. Atributu kopurua aldatuz, eta InfoGain aplikatuz ala ez 10-fCV eginda klase minoritarioarekiko emaitzak.

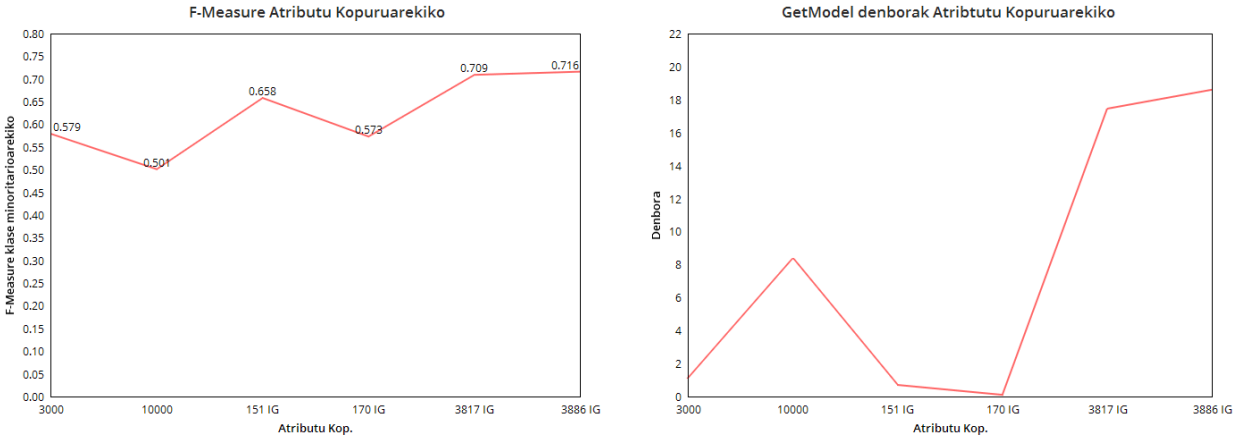
Proba gehigarri hauek *Logistic Regression*-ekin bakarrik egin ditugu, sinpleagoa delako eta exekuzio denbora baxuagoak ematen dituelako konputazio ahalmen txikiago baten truke. *Multilayer Perceptron*-aren kasuan, proba hauek egitea ezinezkoa egin zaigu bere konputazio kostu altua dela eta. Nagusiki erabilitako ordenagailuak hurrengo konfigurazioa dauka: AMD Ryzen 5 3600 6-Core prozesadorea eta 16 GB-ko RAM memoria. Lortutako azken emaitzak lortzeko RAM memoria guztia erabili behar izan dugu, hau da ordenagailua ezin izan dugu haratago eraman frogak egiteko. Frogen denbora, *Logistic Regression* izan arren, disparatu egiten dira (ikusi Taula 17), horregatik denborak estimatuz gero *Multilayer Perceptron*-aren 15000 Atrib. InfoGain-aren frogak aurreikusi ditzakegu. *Logistic Regression*-ek 150 atributurekin 1m 44s behar bazituen eta 3886 atributu izatean 18h 6m 34s behar izan baditu, *Multilayer Peceptron* 14h 53m 54s izatetik 9339h 14m 58s (389 egun) izatera

pasatuko litzateke (guztiz bideraezina guretzat).

Ikasleak garenez, *Cloud Computing* zerbitzu batzuk dohain erabiltzeko aukera dugu, *Azure Cloud Computing*, *Google Cloud Console* edo *Amazon Web Services* adibidez. Hala era, hauek eskainitako 'ordenagailuak' ez digute balio nahi ditugun frogak egiteko, RAM gutxi eta prozesadore ez oso ahaltsuak eskaintzen dituztelako.

	Logistic Regression					
	3000 Atrib. No InfoGain	**10000 Atrib. No InfoGain**	**10000 Atrib. InfoGain (151)**	**15000 Atrib. InfoGain (170)**	**15000 Atrib. InfoGain (3817)**	**15000 Atrib. InfoGain (3886)**
Eredua	0h 8m 53s	0h 47m 51s	0h 0m 5s	0h 0m 13s	0h 3m 1s	2h 53m 4s
Ez Zintzoa	0h 0m 1s	0h 0m 1s	0h 0m 1s	0h 0m 1s	0h 0m 5s	0h 0m 3s
10-fCV	1h 5m 9s	7h 14m 22s	0h 6m 57s	0h 1m 14s	17h 41m 51s	15h 13m 6s
100 Hold-Out	0h 0m 23s	0h 1m 48s	0h 0m 7s	0h 0m 27s	0h 0m 20s	0h 0m 21s
Guztira	**1h 14m 26s**	**8h 4m**	**0h 7m 55s**	**0h 1m 34s**	**17h 45m 17s**	**18h 6m 34s**

Taula 17: *Logistic Regression* eredu iragarlearekin atributuen probak egiteko denborak.



(a) *F-Measure* atributu kopuruarekiko

(b) GetModel denborak atributu kopuruarekiko

Fig. 16: Proba gehigarriak: (a) *F-Measure* klase minoritarioarekiko atributu espazioa kontutan izanda (b) *Logistic Regression* eredua entrenatzeko eta kalitatea estimatzeko beharrezko denborak atributu kopuruarekiko.

5 Ondorioak

Esleitu zaigun lana programa batzuk garatzea da, programa horiek, bakoitzak, funtzionalitate batzuk dituzte, hala nola, datu sorta bat ematean .arff formatura egokitzea, datu horiek hainbat bektore-espaziotan irudikatu ahal izatea eta horiek eredu prediktibo bat (*Multilayer Perceptron*) garatzeko erabiltzea. Ondoren, eredu sinpleago batekin (*Logistic Regression*) alderatuko dena, emaitzak alderatzeko. Eredu horiek eraiki aurretik parametro ekorketa bat egingo da. Bien emaitzak zenbait ebaluazio eskemekin haien kalitatea estimatuko da eta agindutako atazarako zein den onena ebaluatuko da.

Esleitu zaigun algoritmoa *Multilayer Perceptron* da, atzerako hedapena erabiltzen duen sailkatzaile bat, geruza anitzeko pertzeptroi bat ikasteko, instantziak sailkatzeko. Sailkatzaile honen parametro esanguratsuenak *Hidden Layers* eta *Learning Rate* dira.

- *Hidden Layers*: Sailkatzaileak hainbat motako geruza ditu: sarrerako nodo-geruza bat, irteerako nodo-geruza bat eta tarteko geruza bat edo batzuk daude. Barruko geruzak batzuetan "ezkutuko geruzak"deitzen dira, sistemaren sarrera eta irteeretatik ezin direlako zuzenean ikusi. Hau da, sarrerako eta irteerako geruzen arteko nodo-geruzak.[14]

- *Learning Rate*: Ikaskuntza-tasa hiperparametro bat da, eta ereduaren pisuak eguneratzen diren bakoitzean kalkulatutako erroreari erantzuteko eredua zenbat aldatzen den kontrolatzen du.[15]

Zein konfigurazio aukeratzeko orduan nahiko argi gelditu da *Multilayer Perceptron*-a ez dela batere egokia gure atazarako. Denbora eta kostu konputazional handiegiak suposatzen ditu hain ataza 'txikia' egiteko. Ez du zentzurik algoritmo konplexuago bat erabiltzeak *Logistic Regression* bezalako algoritmo bat erabiliz emaitza oso pareak lortzen baditugu denboraren ehunena erabiliz. Zein errepresentazio bektoriala aukeratzerako orduan, gure kasuan aldea mespretxagarria da. Erretxinak jarriz gero esan dezakegu TF·IDF-k emaitza hobeak lortzen ditu, baina berriz esanda, alde infinitesimala da. Denboraren aldetik, ordea, BoW ebaluatzeko TF·IDF-rekin konparatuz denboraren bikoitza behar da (Ikusi Taula 15). Datuen errepresentazio bektoriala Sparse/NonSparse aukeratzeko berdina gertatzen da, denbora eta kostu konputazionalaren aldetik aldea mespretxagarria da. Denbora eta kostu konputazionala modu handian murrizteko oso garrantzitsua da atributu hautapena burutzea. Hemen bai, alde izugarria dago *InfoGain* edo *AttributeSelection* aukeratzean. Gehien merezi duen aukera *InfoGain* da. Laburbilduz,

- Errepresentazio espazioa : TF·IDF eta Sparse/NonSparse
- Atributu hautapena : *InfoGain*
- Sailkapen Algoritmoa : *Logistic Regression*

Lortu nahi genituen helburu guztiak ez dira lortu, ezin izan ditugu nahi izan ditugun froga guztiak eta honek modu handi batean mugatu du gure lana. Eta, gainera, ez dugu nahiko genukeen adina sakondu, alde asko bakarrik gainazalean aipatu ditugu baina asko gustatuko litzaiguke lan honetan ezagutza horiek ikastea eta aplikatzea.

Proiektuaren txostenetik kanpo zenbait ekarpen egin ditugu. Lehenik, GUI (*Graphic User Interface*) bat garatu dugu proiektuaren fitxategien egokitzapena, FSS eta iragarpenak egiten dituena. Gainera, Javatik egin ezin den funtzio bat gaineratu diogu, tweet bat Raw testu bezala jaso dezake eta iragarpena pantailaratu. Bigarrenik, konklusio eta konparaketa guztiak BoW, TF·IDF eta Sparse/NonSparse-rekin egin ditugu. Azkenik, atributuen hautapenean *AttributeSelection* eta *InfoGain* alderatu ditugu, batez ere kostu konputazional aldetik.

Gure taldearen sendotasunak ugariak dira: ikasle onez osaturiko talde bat gara, lanak modu profesional eta serio batean lantzen ditugu; oso langileak gara, ez dugu azken momenturako lana uzten eta asko inplikatzen gara guztion lanarekin; lankidetza nabarmentzen da, modu egokian banatzen dugu lana modu eraginkorragoa eginez; Emmak eta Jonek lan handia egin dute bai inplementazioan bai dokumentazioan eta Xabierrek aurretiko ezagutzak zituen *Multilayer Perceptron* algoritmoari eta bere parametroei buruz.

Lan honen atal edo pieza bakoitzak bere sendotasunak eta ahuleziak aurkezten ditu:

- *Multilayer Perceptron*:
  - Sendotasunak:
    - Eredu ez-linealak sortzeko gai da.
    - Merkatuko aldaketetara egokitzeko trebetasuna, aldizkako berrentrenamendu baten bidez.
  - Ahuleziak:
    - Hiperparametro asko doitu behar dira: *Learning Rate*, *Hidden Layers*...
    - Ez du ondo estrapolatzen, hau da, sarea gaizki edo gutxiegi entrenatzen bada, irteerak zehaztugabeak izan daitezke.
- *Logistic Regression*:

  - Sendotasunak:
    * Inplementatzeko, interpretatzeko eta entrenatzeko oso efizientea.
    * Ez du inolako suposiziorik egiten klaseen banaketari buruz atributuen espazioan.
    * Oso azkarra da erregistro ezezagunen sailkapenean.
  - Ahuleziak:
    * Mendeko aldagaiaren eta aldagai independenteen arteko linealtasun-suposizioa.
    * Muga linealak eraikitzen ditu.
    * Arazo ez-linealak ezin dira ebatzi erregresio logistikoarekin, erabaki azalera lineala duelako.
- BoW/TF·IDF:
  - Sendotasunak:
    * Ulertzeko eta inplementatzeko oso erraza.
    * Konputatzeko erraza.
  - Ahuleziak:
    * Dimentsio handiko ezaugarriak dituen bektore batera darama, hiztegiaren tamaina handia dela eta.
    * Ez ditu baliatzen hitzen arteko erlazioak. Bestela esanda, hitz guztiak independenteak direla onartzen du.
    * Bektore oso sakabanatuak sortzen ditu.

Orokorrean, gure lanaren sendotasuna *Text Mining* eta tweet-en prozesaketarekiko zenbait ondorio lortu ditugula da. Tweet-ak, motzak izateagatik hitz gutxiz osatuta daude, eta lan honen ondoriotako bat hori da, testu motza horiek sailkatzeko atributu espazio handia behar da. Guk 150 atributuekin egiten dugu lan konputazio ahalmena dela eta, baina egoera utopiko batean, 10000 hitz baino gehiago dituen atributu espazio batekin lan egitea gustatuko litzaiguke, eta oinarri hori izanda hitzak handitzea eta murriztea emaitzak alderatuz atributu espazio egokia lortzeko. Denbora falta izan zaigu esperimentu hauek egiteko zenbait hilabete beharko genituelako. Tweet-en sailkapena asko zailtzen da haien luzeragatik eta *hashtag*(#) kontuan hartu behar direlako, ez da berdina love edo #love atributu bezala izatea, *hashtag*-ek askoz informazio gehiago eskaintzen digute. Beste sendotasun bat, esperimentuen zabalera da, errepresentazio espazio guztiekin lan egiten dugu (BoW eta TF·IDF, eta horien barruan Sparse/NonSparse), eta lan egiteko oso erabilgarria den GUI (*Graphic User Interface*) bat gehitzen diogula lanari.

Asko errepikatu den ahultasun bat esperimentuak nahi izan diren moduan bermatu ez direla da. Berriro ere, denbora faltagatik esperimentuen helmuga ez da lortu, emaitzak asko dira; gogobetekorik, ordea, ez.

Ildo honetatik jarraitzeko astia izango bagenu, algoritmoan (*Multilayer Perceptron*) askoz gehiago sakondu ahalko genuke. Lehen aipatu den bezala ez dugu nahiko genukeen adina sakondu, alde asko bakarrik gainazalean aipatu ditugu baina asko gustatuko litzaiguke lan honetan ezagutza horiek ikastea eta aplikatzea. Eta beharbada nahi genituen baino ekarpen gutxiago egin ditugu lanean, denbora gehiagorekin, berriro diot, askoz gehiago sakonduko genuen lanaren 'pieza' bakoitzean. Egia da lana berriro hasiko bagenu, lortutako esperientziarekin beste modu batean antolatuko genukeela. Konturatu gara kodearen inplementazioa baino, frogapenak direla denbora gehiago behar dutenak, baita informazioaren sakonketa ere. Ondorio nagusi bat atributu kopurua nahikoa ez dela da, proiektu guztiak berregingo bagenu konputazio ahalmen handiago bat

duen ordenagailu batekin egingo genituzke probak, atributu espazio askoz handiago bat hartuz eta atributu hautapenean sakonduko genuke kontuz handiz atributu espazio nahikoa lortzeko.

Bibliografia

[1] ALPAIDIN, E. (2014). Multilayer perceptrons. *Introduction to Machine Learning* (Third edition, pp. 267–316). Massachusetts Institute of Technology.

[2] HOTHO, A., NÜRNBERGER, A., PAASS, G. (2005, May). A brief survey of text mining. *Ldv Forum* (Vol. 20, No. 1, pp. 23-62). http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.447.4161&rep=rep1&type=pdf

[3] GREGORY PIATETSKY-SHAPIRO (2008). Machine Learning, Data Mining, and Knowledge Discovery: An Introduction. *KDnuggets*. https://sorry.vse.cz/~berka/docs/4iz451/dm01-introduction-ml-data-mining.pdf

[4] INMACULADA GONZÁLEZ DE MOLINA (2020, July). Los datos digitales generados por una persona al día ocupan el espacio de 86.400 ejemplares del Quijote. *La Razón*. https://www.larazon.es/actualidad/20200715/6wqnlceujbgdnapsgipumdrudm.html#:~:text=La%20Humanidad%20ha%20pasado%20de,a%201.000%20millones%20de%20Terabytes).

[5] ABHINAV RAI (2019, June). What is Text Mining: Techniques and Applications. *upGrad blog*. https://www.upgrad.com/blog/what-is-text-mining-techniques-and-applications/

[6] RAHELL SHAIKH (2018, Oct.). Feature Selection Techniques in Machine Learning with Python. *towards data science*. https://towardsdatascience.com/feature-selection-techniques-in-machine-learning-with-python-f24e7da3f36e

[7] DAVID MILWARD (2019, Aug.). What is Text Mining, Text Analytics and Natural Language Processing? *Linguamatics*. https://www.linguamatics.com/what-text-mining-text-analytics-and-natural-language-processing

[8] WITTEN, I. H., FRANK, E., AND HALL, M. A. (2011). Data Mining: Practical Machine Learning Tools and Techniques. The Morgan Kaufmann Series in Data Management Systems, 3rd edition.

[9] JASON BROWNLEE (2016, Apr.). Logistic Regression for Machine Learning.*Machine Learning Mastery*. https://machinelearningmastery.com/logistic-regression-for-machine-learning/

[10] SONOO JAISWAL (2018). Logistic Regression in Machine Learning.*Javatpoint*. https://www.javatpoint.com/logistic-regression-in-machine-learning

[11] CHARU C. AGGARWAL (2018) Neural Networks and Deep Learning. (pp. 24-25) https://link.springer.com/book/10.1007%2F978-3-319-94463-0

[12] CHARU C. AGGARWAL (2018) Neural Networks and Deep Learning. (pp. 25-37) https://link.springer.com/book/10.1007%2F978-3-319-94463-0

[13] MIROSLAV KUBAT (2017) An Introduction to Machine Learning. (pp. 91-99) https://www.springer.com/gp/book/9783319639123

[14] JASON BROWNLEE (2018 July). How to Configure the Number of Layers and Nodes in a Neural Network.*Machine Learning Mastery*. https://machinelearningmastery.com/how-to-configure-the-number-of-layers-and-nodes-in-a-neural-network/

[15] JASON BROWNLEE (2019 Jan.). Understand the Impact of Learning Rate on Neural Network Performance.*Machine Learning Mastery*. https://machinelearningmastery.com/understand-the-dynamics-of-learning-rate-on-deep-learning-neural-networks/