eman ta zabal zazu
Universidad del País Vasco
Euskal Herriko Unibertsitatea
ZIENTZIA ETA TEKNOLOGIA FAKULTATEA
FACULTAD DE CIENCIA Y TECNOLOGÍA

Gradu Amaierako Lana / Trabajo Fin de Grado
Elektronikako Gradua / Grado en Electrónica

Deep Learning-eko metodologia eta tresnak *Data Science*-an

Egilea/Autor/a:
Paul Navarlaz Muguruza
Zuzendaria/Director/a:
Mikel Peñagaricano Badiola

Leioa, 2019ko ekainaren 21a /Leioa, 21 de junio de 2019

Gaien Aurkibidea

1 Sarrera

Deep Learning-a *Machine Learning*-aren barneko teknika multzo bat da non sare neuronal artifizialak erabiltzen diren prozesu jakin bat automatizatzeko asmoz. *Deep Learning* honetan datu multzo batetik informazioa lortzen saiatzen da honetarako ikasteko gai diren hainbat geruza jarrai erabiliz, geruza hauetako bakoitzak geroz eta informazio esanguratsuagoa gordetzen duelarik [1].

Testuen klasifikazioa testu jakin bat aurretik definituriko multzo finitu batean sailkatzeko zeregina da. Honek hainbat aplikazio ditu, besteak beste: dokumentuen generoaren identifikazioa, *spam* filtroak, kontsumitzaileentzako informazio gaien aukeraketa selektiboa... Klasifikatu beharreko informazio kopurua handitu ahala, beraz, baliagarria da prozesu honen automatizazioa burutzea. Azken urteotan eman den testuen digitalizazioaren ondorioz, gainera, dokumentuen klasifikazioaren zereginearekiko interesa handitu egin da. Honela, problema honi eman ohi zaion ikuspuntua *Machine Learning*-arena da, aurretik klasifiktatuak izan diren dokumentu multzo batetik kategoria bakoitzaren ezaugarriak ikasiz klasifikazioa automatizatuz [2].

Lan honen helburua testuen klasifikazio prozesuan erabil daitezkeen hainbat teknika aztertu eta alderatzea da. Landu diren teknikak ondorengoak dira: hitzen agerpen maiztasuna, UBM-a, erregresio logistikoa eta sare neuronalak. Lan honetan hauetariko bakoitza aztertzen da, bakoitzaren abantailak eta ezberdintasunak azalduz.

Klasifikazioaren helburua esaldien autorea klasifikatzea da, honetarako esaldiak hitzetan banatuz, eta hitz hauek predikzio prozesurako erabiliz. Lan honetan erabiltzen diren testuak ingelesez badira ere, prozedura hauek orokorrak dira beste edozein hizkuntzarentzat, baldin eta esaldiak hutsunez banandutako hitzez osatzen badira.

2 Metodologia

2.1 Prozedura

Testu baten autoretzaren klasifikazioa egiteko metodoak anitzak izanik ere, orokorrean guztiek prozedura amankomun bat jarraitzen dute.

- **Datuen aurre prozesaketa** Idealki, abstrakzio maila handiena kontsideratuz, makina abstraktu bat sortu nahi da zeini testu edo esaldi bat ematen zaion sarrera gisa. Honek irteeran esaldia zein idazlerena den azaltzen du. Hortaz, testu hau nola karakterizatu erabaki beharra dago makinak informazio hau jaso dezan. Karakterizazio honek esaldiari buruzko informazioa gorde behar du sailkapena eraginkorra izan dadin, baina karakterizazio hau nolakoa izan ereduaren sortzailearen araberakoa da.
- **Modeloaren entrenamendua** Behin sarrerako datuak tratatu ondoren, modeloaren entrenamendua egiten da. Honetarako, *gainbegiraturiko entrenamendua* (*Supervised Leaarning*) edo *gainbegiratu gabeko entrenamendua* (*Unsupervised learning*) erabil daitezke. Azken honen kasuan sareak sarrerako datuak jasotzen ditu eta bere kabuz patroiak bilatzen ditu, hauen arteko loturak ikasiz. Gainbegiratuen kasuan, aldiz, entrenamendu prozesuan sareak sarrerako datuez gain, irteeran idealki nahi den egitura jasotzen du, honekin barne egitura eraldatzen duelarik emaitza hobetzeko asmoz. Aztergai den problema konkretuaren kasuan gainbegiraturiko entrenamendua burutu da, esaldi bakoitza zein idazlerena den adieraziz progamari.
- **Modeloaren ebaluazioa** Atal honetan entrenamendu prozesuaren eraginkortasuna aztertzen da. Hau da, makinari guk ezagunak ditugun esaldiak ematen zaizkio, honek zein idazlerena den aurresan dezan. Ebaluazioko esaldi hauen benetako idazlea ezaguna denez, makina honen eraginkortasuna neurtzen dugu honela. Ebaluazio prozesu honetan ezinbestekoa da entrenamendu prozesuan erabili ez diren esaldiak erabiltzea. Izan ere, gure helburua ezezagunak diren esaldien autorea ezagutzea da. Hori dela eta, makina honek aurretik ikusi ez dituen esaldiak erabili behar dira egokitasuna aztertu ahal izateko. Hau da, makinak problema orokorrera generalizatzeko eta esaldi ezezagunen aurrean klasifikazio egokia egiteko gai izan behar du.

  Halaber, ebaluazio prozesu honetan bi datu sorta erabiliko dira: entrenamendu prozesuan erabilitako datuak eta aurretik erabili ez direnak. Lehenengo datu sortaren bidezko ebaluazioa egokia bada eta bigarren datu sortaren bidezkoa ez, gehiegizko entrenamendua burutu denaren adierazle da (*Overfitting*). Hau da, entrenamendu datuetara egokitu da emaitza onak emanez, baina ez da gai generalizatzeko. Gerta liteke, ordea, entrenamendu datuekin ere klasifikazioa egokia ez izatea. Kasu honetan gutxiegiko entrenamendua (*Underfitting*) ematen dela adierazten du. Beste era batean esanda, modeloa ez da gai entrenamendu datuak erabiliz sarrerako datuekin eta irteerak jakinik, hauen arteko patroirik aurkitzeko.

Adierazi beharra dago, sareak bere eginkizuna burutzeko beharrezkoa dela entrenamendu prozesuan erabilitako datuak klasifikatu nahi den klasearen ahalik eta orokorrenak izatea, hauen adierazpide egokia izateko. Beste era batean esanda, idazle jakin baten kasuan testuak erabiltzean, komenigarria da hainbat iturri ezberdinetako esaldiak erabiltzea. Hau egin ezean, datu sorta honek ez ditu egoki adieraziko idazle konkretu horrek idazkeran dituen joerak.

2.2 Problemaren analisia

Azterketa honetan dokumentu edo testu batzuk idazle ezberdinen artean banatzea dugu helburu. Honetarako A kategoria edo klase sorta bat izango dugu:

$$A = \{a_k | k = 1, 2, ..., n\} = \{a_1, a_2, ..., a_n\} \tag{2.2.1}$$

kategorietako bakoitza idazle bat izanez. Hauek izanik, Y dokumentu edo textu multzo bat jasoko dugu y_i esaldiez osatua dagoena.

$$Y = \{y_i | i = 1, 2, ..., I\} = \{y_1, y_2, ..., y_I\} \tag{2.2.2}$$

non y_i dokumentuari dagokion a_k autorea ezezaguna den. Klasifikazio prozesu hau egiteko, y_i testuari dagokion A sortako klase bakoitzekoa izateko probabilitatea kalkulatzen da, hau da:

$$P(y_i) = \{P(a_1|y_i)\ ,\ P(a_2|y_i)\ ,\ ...\ ,\ P(a_n|y_i)\} \tag{2.2.3}$$

Probabilitate hauek kalkulatu ondoren, probabilitate hauetan balio altuena duen klaseari esleitzen zaio y_i esaldia.

$$\hat{a}(y_i) = \arg\max_{a_k \epsilon A} P(a_k|y_i) \tag{2.2.4}$$

Honenbestez, hau egin ahal izateko $P(a_k|y_i)$ probabilitateak esleitzeko prozedurak proposatzea beharrezkoa da. Hori dela eta, lan honetan probabilitate hauek lortzeko hainbat metodo ezberdin aztertzen dira, hauen abantaila eta desabantailak aztertuz.

2.3 Datuen analisia

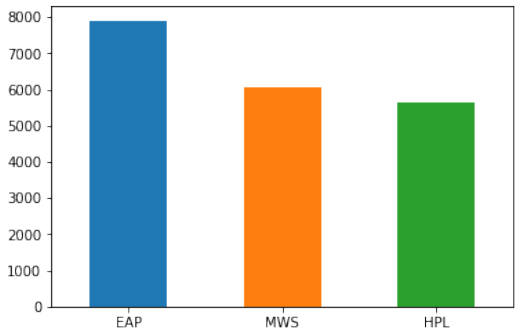
Hau burutu ahal izateko hainbat autoreren testuak dituen datu base bat erabiltzea beharrezkoa da. Hori dela eta, *Kaggle* webguneaz baliatu gara. Web orri honetan hainbat datu base biltzen dira, erabiltzaileek beren ebazpenak igo eta elkarbanatzen dituztelarik. Hemen erabiliko den datu basea *Spooky Author Identification* [3] lehiaketako datu basea izan da. Hortaz, lehenik datu basean eskuragarri dagoen informazioa aztertuko da. Datuen analisia eta azaltzen diren emaitzak datu base hau erabiliz lortu dira. Hala eta guztiz ere, proposaturiko teknikak orokorrak dira edozein datu baserentzat.

Lehenik, datu basean dugun informazioa aztertu beharra dago. Honetarako datu basearen lehen lerroak adierazi dira 2.1 taulan.

	id	text	author
0	id26305	This process, however, afforded me no means of...	EAP
1	id17569	It never once occurred to me that the fumbling...	HPL
2	id11008	In his left hand was a gold snuff box, from wh...	EAP
3	id27763	How lovely is spring As we looked from Windsor...	MWS
4	id12958	Finding nothing else, not even gold, the Super...	HPL
5	id22965	A youth passed in solitude, my best years spen...	MWS
6	id09674	The astronomer, perhaps, at this point, took r...	EAP
7	id13515	The surcingle hung in ribands from my body.	EAP

Taula 2.1: Datu basearen egituraren analisia.

Datu base honetan datu guztiak 19579×3 tamainako matrize batean gordetzen dira, taulan matrize honen lehen 8 lerroak adieraziz. Lehen zutabean esaldi bakoitzaren identifikatzaile bat gordetzen da; bigarrenean, esaldia bera; eta hirugarrenean, esaldi horren idazlea. Ondorengo 2.3.1 irudian idazle bakoitzak duen esaldi kopurua adierazi da.

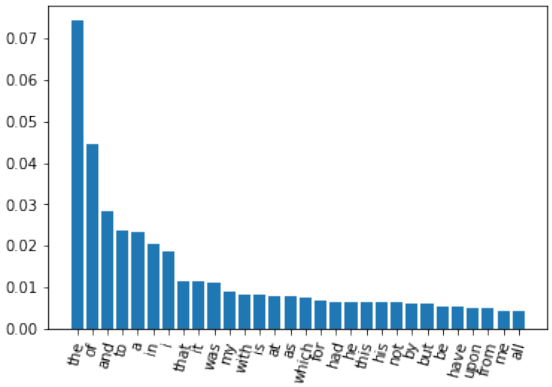


Irudia 2.3.1: Idazle bakoitzaren esaldi kopurua datu basean.

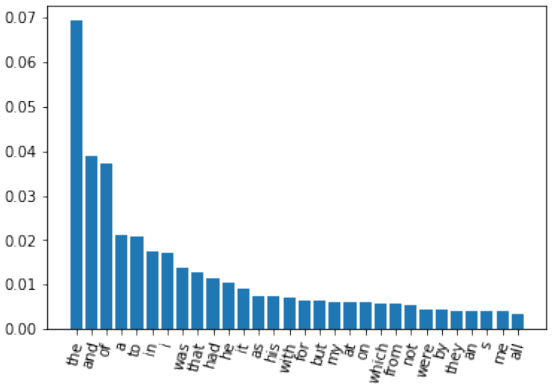
Grafikoan ikus daitekeenez, datu base honetan hiru idazle ezberdin azaltzen dira: Edgar Allan Poe (EAP), HP Lovecraft (HPL) eta Mary Wollstonecraft Shelly (MWS). Idazle bakoitzak duen esaldi kopurua ez da berdina. Autoreen esaldi kopuru ezberdina izatea ohikoa da, eta orokorrean ez da kaltegarria diferentzia oso nabarmena ez bada. Idazle bakoitzaren ahalik eta informazio kopuru handiena izatea interesatzen da betiere.

Ondoren, idazle bakoitzak esaldietan erabiltzen dituen hitzen histogramak sortu dira eta idazle bakoitzak gehien erabiltzen dituen hitzak adierazten dira.

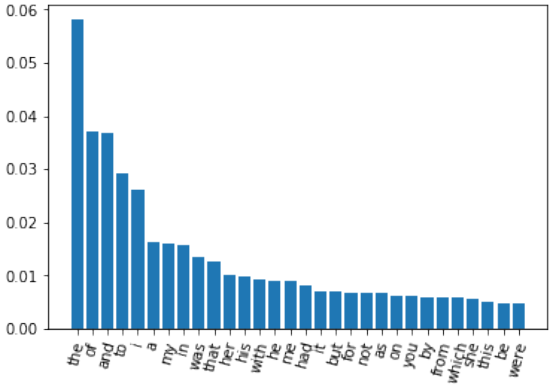
2.3.2 irudietan azaltzen denez, idazle guztien kasuan gehien erabilitako hitzen artean hitz errepikatu asko azaltzen dira. *"the","of","and"...* eta antzeko lokailuak dira nabarmentzen direnak beste hitzen gainetik agerpen maiztasunean. Idazle bakoitzaren kasuan ezberdintasun txikiak azaltzen dira, kasuan kasu duten hitz bakoitza erabiltzeko joeraren arabera. Hortaz, hitzen erabilpen kopuru ezberdin erlatibo hau erabil daiteke idazleen arteko ezberdintasuna bilatzeko. Kasu honetan, puntuazio zeinuak eta karaktere larriak ez dira kontuan hartu. Ondorengo 2.3.3 irudietan histogramak adierazten dira berriro ere, kasu honetan puntuazio zeinuak eta letra larriak kontuan hartuz. Hau da, bigarren kasu honetan *"Finding"* eta *"finding"* hitzak, adibidez, ezberdintzat hartzen dira. Puntuazio zeinuei dagokienez, aurreko kasuan hauen presentzia



(a) Edgar Allan Poe idazlearen hitzen agerpen probabilitatea datu baseko esaldietan.



(b) H.P. Lovecraft idazlearen hitzen agerpen probabilitatea datu baseko esaldietan.

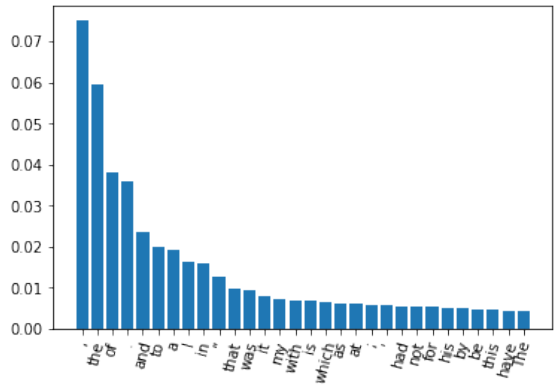


(c) Mary Wollstonecraft Shelley idazlearen hitzen agerpen probabilitatea datu baseko esaldietan.

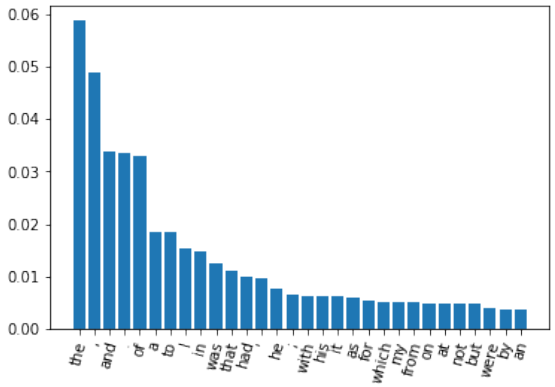
Irudia 2.3.2: Idazle bakoitzak erabilitako hitzen agerpen probabilitatea. Puntuazio zeinuak eta karaktere larriak ez dira kontuan hartu.

arbuiatu egiten zen; oraingoan, berriz, hauen agerpen maiztasuna ere kalkulatzen da.

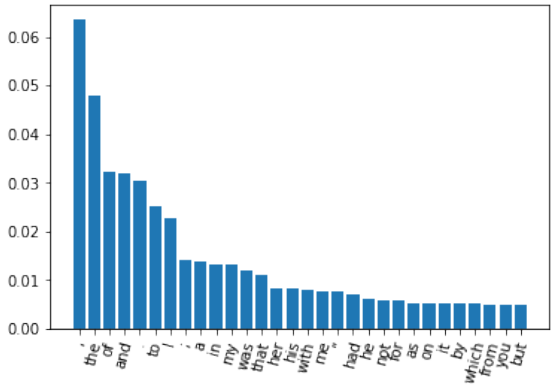
2.3.3 irudietan ikus daitekeenez, puntuazio zeinuak gehien erabiliriko elementuen artean agertzen dira. Hau espero litekeen kasua da, puntuazio zeinuak esaldi guztietan erabiltzen baitira. Printzipioz suposa genezake puntuazio zeinuek eta karaktere larriek eragina dutela testuaren autore identifikazioan. Adibidez, idazle bakoitzak esaldiak hitz batekin hasteko joera izan lezake. Hortaz, informazio hori gordetzea erabilgarria litzake idazlea identifikatzeko. Hori dela eta, azteketa honetan bai puntuazio zeinuen eta bai letra larrien agerpenak duten eragina aztertu da.



(a) Edgar Allan Poe idazlearen hitzen agerpen probabilitatea datu baseko esaldietan.



(b) H.P. Lovecraft idazlearen hitzen agerpen probabilitatea datu baseko esaldietan.



(c) Mary Wollstonecraft Shelley idazlearen hitzen agerpen probabilitatea datu baseko esaldietan.

Irudia 2.3.3: Idazle bakoitzak erabilitako hitzen agerpen probabilitatea, puntuazio zeinuak eta karaktere larriak kontuan hartuz.

2.4 Ebaluazioa

Entrenamendu prozesua egin ondoren modelo baten klasifikaziorako eraginkortasuna neurtzea beharrezkoa da, aurreko 2.1 atalean azaldu den bezala. Honetarako bi faktore erabili dira aztergeta honetan: zehaztasuna (edo doitasuna) eta galera logaritmikoa (*logarithmic loss*) *categorical crossentropy* izenarekin ere ezagutzen dena.

2.4.1 Zehaztasuna

Ereduak sailkatu nahi den y_i sarrerako elementu bat jasotzen du eta $A = \{a_1, a_2, ..., a_n\}$ multzoetako batean klasifikatzen du. Horretarako a_k autoreetako bakoitzari probabilitate bana ematen dio 2.2.3 adierazpenak azaltzen duen moduan. Honela, modeloak aurresaten duen klasea probabilitate hauetan balio altuena duena izango da, $\hat{a}$ bidez adierazten dena 2.2.4 ekuazioan azaldu denez. y_i sarrerako datuaren a_{true} benetako klasean klasifikatzeko modeloaren zehaztasuna honela

kalkulatzen da:

$$Zehaztasuna \ = \frac{1}{I}\sum_{i=1}^{I}\delta_{\hat{a},a_{true}} \tag{2.4.1}$$

non $\delta_{\hat{a},a_{true}}$ Kroneckerren delta den. Balio honek, espero bezala, egoki klasifikaturiko datu kopuruen ehunekoa adierazten du $[0,1]$ tarte bornatuan.

2.4.2 Galera logaritmikoa

Galera logaritmikoa, *categorical crossentropy* izenez ere ezaguna dena klasifikazio modelo baten eraginkortasuna neurtzeko erabiltzen den tresna matematikoa da. Honetan, bi probabilitate distribuzioren alderaketa egiten da. Erreferentziazko probabilitate distribuzio bat hartuz, bigarren probabilitate banaketa batek honekiko duen entropia adierazten du. Azterketa honetako bi distribuzio probabilitateak ondorengoak dira: lehena, 2.2.3 adierazpeneko klasifikazio modeloak aurresaten dituen P probabilitateen banaketa. Bigarren distribuzio probabilitatea Z emaitza zuzena edo jomuga da. Honetan y_i esaldiaren autore zuzena q posizioan baldin badago, lortu nahi den emaitza zuzenak ondorengo probabilitate banaketa du.

$$Z(y_i) = \{z_{i1} = 0, z_{i2} = 0, ..., z_{i(q-1)} = 0, z_{iq} = 1, z_{i(q+1)} = 0, ..., z_{in} = 0\} \tag{2.4.2}$$

Honenbestez, erreferentziazko probabilitate distribuzioa edo banaketa Z banaketak ematen du. Galera logaritmikoak P multzoko probabiltateek Z banaketarekiko duen entropia neurtzen du. Nahasketa entropia hau honela definitzen da [4]:

$$\mathcal{S}(Z(y_i), P(y_i)) = -\sum_{k=1}^{n} z_{ik} log(P(a_k|y_i)) \tag{2.4.3}$$

2.4.3 ekuazioan y_i esaldiak ematen duen entropia kalkulatzen da. Hortaz, Y ebaluazio multzoko esaldi guztien entropia kalkulatzeko, bakoitzaren entropiaren batura egin behar da.

$$\mathcal{S}(Y) = \sum_{j=1}^{I}\mathcal{S}(Z(y_j), P(y_j)) = -\sum_{j=1}^{I}\sum_{k=1}^{n} z_{jk} log(P(a_k|y_j)) \tag{2.4.4}$$

Azkenik, klasifikazio problema honetan Z banaketaren definizioa erabiliz definitzen da galera logaritmikoa. Izan ere, klasifikazio prozesu honetan Z multzoko elementu bakarra izango da ez nulua esaldi bakoitzerako. Hau da, 2.4.4 adierazpeneko z_{jk} balio gehienak nuluak dira, eta ez dute ekarpenik egiten entropiaren kalkuluan. Horrez gain, Y multzoko esaldi kopuruarekin zatitzen da entropia hau, honela, esaldi unitateko entropia definituz.

$$Log\ loss = -\frac{1}{I}\sum_{j=1}^{I}\sum_{k=1}^{n}\delta_{jk,jq} log(P(a_k|y_j)) \tag{2.4.5}$$

non

- $P(a_k|y_j)$ gaia k autorearen j. esaldiari emaniko probabilitatea den.
- $\delta_{jk,jq}$ Kroneckerren delta da, hau da, balio unitarioa hartzen du j.esaldia a_{true} autorearena bada eta nulua bestela.

Funtzio honen balio minimoa $Log\ loss = 0$ da. Hau beteko da baldin eta modeloak benetako idazle guztiak asmatzen baditu %100 probabilitatearekin, hau da, $P(a_{true}|y_i) = 1$ adierazten bada Y multzoko esaldi guztietarako. Galera logaritmiko honek, ordea, ez du balio maximorik. Izatez, esaldi baten kasuan modeloak probabilitate nulua emanez gero autore zuenari, hau da,

modeloak $P(a_{true}|y_i) = 0$ aurresanez gero, funtzio honek infiniturantz jotzen du. Arazo hau eman ez dadin, galera logaritmikoaren kalkulua egitean $P(a_{true}|y_l) = 0$ probabilitateari mugapenak ezartzen zaizkio [5].

$$P(a_k|y_i) = max\left(min\left(p, 1 - 1 \cdot 10^{-15}\right), 1 \cdot 10^{-15}\right) \tag{2.4.6}$$

Probabilitate hau ez bada mugako 0 eta 1 balioetatik oso hurbila, mugapen honek ez du aldatzen modeloak ematen duen probabilitatea. Balioa muturretariko batetik hurbil badago, ordea, mugapen honek galera logaritmikoa infiniturantz joatea galarazten du, klasifikazio oker batek *log loss* honetan duen ekarpena mugatuz. Arrazoi honen ondorioz $P(a_{true}|y_i)$ balioari minimo hau ezartzeagatik goi limite bat ere ezartzen zaio. Printzipioz $P(a_{true}|y_i) = 1$ probabilitateak ez du arazorik ematen galera logaritmikoaren kalkuluan baina minimoko mugapenaren ondorioz maximoan ere mugatu egiten da konpentsazioagatik.

2.4.3 Galera logaritmikoaren balioaren esangura

Galera logaritmikoaren interpretazioa egiteko lehenik informazio gabeko modeloa kontsideratuko dugu. Eredu honetan, a_k klaseei buruzko informaziorik ez dugunez, denei probabilitate berbera esleitzen zaie.

$$P(a_k|y_i) = \frac{1}{n} \quad \forall k \in [1, n] \tag{2.4.7}$$

Probabilitate hauen esleipena eginaz gero esaldi guztietan, *log loss* funtzioarekin lortzen den emaitza ondorengoa da.

$$Log\ loss = -\frac{1}{I}\sum_{j=1}^{I} log\left(\frac{1}{n}\right) = -\frac{1}{I} log\left(\prod_{j=1}^{I}\frac{1}{n}\right) = -\frac{1}{I} log\left(\left(\frac{1}{n}\right)^I\right) = log(n) \tag{2.4.8}$$

Logaritmo hau oinarri bitarrean erabiliz gero, *log loss*-ak esaldiko bataz bestean galtzen den informazioa adierazten du oinarri bitarrean. Adibidez, idazle kopurua $n = 8$ balitz *log loss*= 3 izango litzake, hau da, hiru bit informazio gehiago behar dira klasea era egokian aukeratzeko. Modeloak klasifikatu nahi den problemako informazio kopurua handitu ahala 2.4.7 ekuazioko probabilitateak aldatu egiten dira, galera honen balioa txikituz. Lan honetan logaritmo naturala erabili da oinarri bitarra ordez. Honek eskala aldaketa bat suposatzen du, baina bi kasuetan esaldiei ematen zaien entropia adierazten da.

Loss funtzio honen beste interpretazio bat probabilistikoa da. 2.4.5 ekuazioko $\delta_{jk,jq}$ terminoaren definizioagatik, batugaian erabiltzen diren gai ez nulu bakarrak egiazko autoreei ematen zaizkien $P(a_{true}|y_l)$ probabilitateak dira. Hau jakinik:

$$\begin{aligned} Log\ loss &= -\frac{1}{I}\sum_{j=1}^{I}\sum_{k=1}^{n}\delta_{jk,jq} ln(P(a_k|y_j)) = -\frac{1}{I}\sum_{j=1}^{I} ln(P(a_{true}|y_j)) = \\ &= -\frac{1}{I} ln\left(\prod_{j=1}^{I} P(a_{true}|y_j)\right) = -ln\left(\sqrt[I]{\prod_{j=1}^{I} P(a_{true}|y_j)}\right) \end{aligned} \tag{2.4.9}$$

2.4.9 ekuazioko azken adierazpenean logaritmo naturalaren barnean dagoen terminoa probabilitateen batez-besteko geometrikoa da. Hortaz, modeloak autore zuzenari eman dion probabilitatea

batezbestean zenbatekoa den adierazten du, batezbesteko hau geometrikoa izanez.

$$< P(a_{true}|y_i) >_g = \sqrt[I]{\prod_{j=1}^{I} P(a_{true}|y_j)} \tag{2.4.10}$$

Probabilitateak normalizaturik egonaz batezbesteko geometrikoa ere $\left[1 \cdot 10^{-15}, 1 - 1 \cdot 10^{-15}\right]$ balioen artean mugaturik dago 2.4.6 baldintzaren ondorioz. Logaritmo naturalak tarte honetan balio negatiboak hartzen dituenez, zeinu negatiboa dela eta balio positiboa hartzen du galera logaritmikoak. Bestalde, batezbesteko geometrikoa zenbat eta handiagoa izan, galera logaritmikoa zerotik hainbat eta hurbilago egongo da.

3 Hitzen agerpen maiztasun bidezko klasifikazioa

3.1 Entrenamendu eta ebaluazio prozesuak

Problemaren analisian azaldu den moduan, gure helburua y_i textu bakoitzari a_k autore bakoitzarena izateko probabilitate bat ematea da: $P(a_k|y_i)$. Probabilitate honek y_i gertaera eman bada a_k gertatzeko probabilitatea adierazten du, hau da, y_i esaldia edukita hau a_k idazleak idatzia izateko probabilitatea. Probabilitate hau kalkulatu ahal izateko beharrezkoa da entrenamendu prozesu bat egitea, autore ezberdinei buruz informazioa ikasteko.

3.1.1 Entrenamendu prozesua

Entrenamendu prozesuan zehar a_k autoreetako bakoitza ezaugarritzen duten karakteristikak definitzen dira, ondoren probabilitateak ezaugarri hauen bidez definituz. Lan honen kasuan karakteristika hauek definitzeko autore bakoitzak erabiltzen dituen hitzen agerpen maiztasuna erabiliko da. Entrenamendu prozesuan X dokumentu edo testu multzo bat izango dugu, hainbat esaldiz osatua dagoena eta esaldi bakoitza zein idazlek idatzi duen ezaguna izanda.

$$X = \{x_i|i = 1, 2, ..., T\} = \{x_1, x_2, ..., x_T\} \tag{3.1.1}$$

Esaldi hauetako bakoitza hitz segida batez osatua dago aldiberean.

$$x_i = (w_{ij}|j = 1, 2, ..., w_{iL_i}) = (w_{i1}, w_{i2}, ..., w_{iL_i}) \tag{3.1.2}$$

non

- i indizeak entrenamendu multzoko $i.$ esaldia adierazten duen.
- L_i balioa esaldi honen luzeera den.

Esaldi bakoitzaren idazlea ezaguna denez posiblea da entrenamendu prozesuan idazle batek idatzi dituen hitz guztien multzoa osatzea.

$$W_{a_k} = \{w_j|j = 1, 2, ..., J_k\} = \{w_1, w_2, ..., w_{J_k}\} \tag{3.1.3}$$

a_k idazlearen hitz multzo hau izanik, hitz hauetako bakoitzaren agerpen maiztasuna kalkulatzen da.

$$f_{a_k}(w_i) = \frac{\#(w_i|a_k)}{\sum\limits_{w_j \in W_{a_k}} \#(w_j|a_k)} \tag{3.1.4}$$

Agerpen maiztasun hauek erabiliz a_k autoreak w_i hitza erabiltzeko duen probabilitatea agerpen maiztasun hauekin hurbilduko dugu. Datu basean a_k idazle honen zenbat eta esaldi gehiago izan, hurbilketa hau orduan eta hobea izango da.

$$P(w_i|a_k) = f_{a_k}(w_i) \tag{3.1.5}$$

Honenbestez, entrenamendu prozesuan datu base bat jasotzen da A multzoko idazle bakoitzaren esaldiekin. Idazleak ezaugarritzeko, 3.1.4 ekuazioko hitzen agerpen maiztasunak erabiltzen dira. Ondoren hauek nola erabili aztertzen da, gure helburu diren $P(a_k|y_i)$ probabilitateak lortzeko.

3.1.2 Ebaluazio prozesua

Ebaluazio prozesu honetako y_i esaldietako bakoitza hitz segida batez osatua dago.

$$y_i = (w_{im}|m = 1, 2, ..., L_i) = (w_{i1}, w_{i2}, ..., w_{iL_i}) \tag{3.1.6}$$

non:

- i indizeak Y ebaluazio sortako esaldietan i. esaldia adierazten duen.
- m indizeak y_i esaldiko m. hitza adierazten duen.
- L_i luzeera y_i esaldiaren luzeera den.

Esaldi hauek izanik, $P(a_k|y_i)$ probabilitateak kalkulatzeko Bayes-en teorematik abiatuko gara [6]:

$$P(a_k|y_i) = \frac{P(y_i|a_k)P(a_k)}{P(y_i)} = \frac{P(y_i|a_k)P(a_k)}{\sum\limits_{a' \in A} P(y_i|a')P(a')} \tag{3.1.7}$$

non $P(a')$ gaia *prior*-a den, hau da, ebaluazio prozesuan a' autorearen esaldiak agertzeko berezko probabilitatea. Probabilitate hau izatez ezezaguna da, baina hurbilketa gisa $P(a') = \frac{1}{n}\ \forall a' \in A$ erabil daiteke. Hau da, multzo honetako edozein idazleren esaldiak ebaluatzeko probabilitatea berbera dela onartzen da. Honela, 3.1.7 ekuazioko probabilitatearen adierazpena ondorengo eran sinplifikatzen da.

$$P(a_k|y_i) = \frac{P(y_i|a_k)}{\sum\limits_{a' \in A} P(y_i|a')} \tag{3.1.8}$$

Bayesen teoremaren aplikazio honek gure helburu diren $P(a_k|y_i)$ probabilitateen kalkulua burutzea ahalbidetzen du. Izan ere, entrenamendu prozesuan lortutako probabilitateen bitartez $P(y_i|a_k)$ kalkulatzea posible dugu. 3.1.8 ekuazioa erabiliz, beraz, $P(a_k|y_i)$ eta $P(y_i|a_k)$ probabilitateen arteko lotura lortzen da. Honenbestez, azken probabilitate hau entrenamendu prozesuan lortutako hitzen erabilpen probabilitatearekin erlazionatu behar da ebaluazio prozesurako.

a_k idazle bat dugula onartuz, y_i esaldia berak idatzia izateko probabilitatea 3.1.9 ekuazioaren bidez kalkulatzen da.

$$P(y_i|a_k) = P(w_{i1}|a_k) \cdot P(w_{i2}|a_k) \cdot ... \cdot P(w_{iL_i}|a_k) = \prod_{m=1}^{L_i} P(w_{im}|a_k) \tag{3.1.9}$$

3.1.9 adierazpen hau matematikoki zuzena bada ere, arazo bat dakar probabilitateak kalkulatzerako garaian. Izan ere, y_i esaldia osatzen duten w_{im} hitzetako bat entrenamendu datuetan agertu ez bada prozedura hau jarraituz gero $P(a_k|y_i) = 0$ emaitza lortzen da zuzenean. Hau oztopo bat da probabilitateen kalkulurako garaian, esaldian agertzen diren beste hitzek ematen

duten informazioa galtzen baita honen ondorioz. Hau saihesteko aukera posible bat hitzen agerpen probabilitateari desplazamendu lineal bat aplikatzea da.

$$P'(w_i|a_k) = \frac{\#(w_i|a_k)}{\sum\limits_{w_j \in W_{a_k}} \#(w_j|a_k)} + \alpha \tag{3.1.10}$$

non α konstante erreal positibo bat den. Desplazamendu honetan erabiltzen den α balioa txikia aukeratzea komenigarria da. Izan ere, autore batek entrenamendu prozesuan w_i hitza erabili ez badu, beste esaldi batzuetan erabiltzeko probabilitatea orokorrean txikia izatea espero da. Hala ere, 3.1.5 ekuazioan probabilitateak definitu diren eran, entrenamendu prozesuan agertu ez den edozein hitz erabiltzeko probabilitatea nulutzat hartzen da eta mugapen hau oso murriztailea da. Lan honetan aukeratu den α konstantearen balioa ondorengoa da.

$$\alpha = \frac{1}{2} min\left[f_{a_k}(w_i)\right] \quad w_i \in W_{a_k} \tag{3.1.11}$$

Honela, ebaluazio prozesuko probabilitateen kalkulua 3.1.10 ekuazioko probabilitate berriekin egin da.

$$P(y_i|a_k) = \prod_{m=1}^{L_i} P'(w_{im}|a_k) \tag{3.1.12}$$

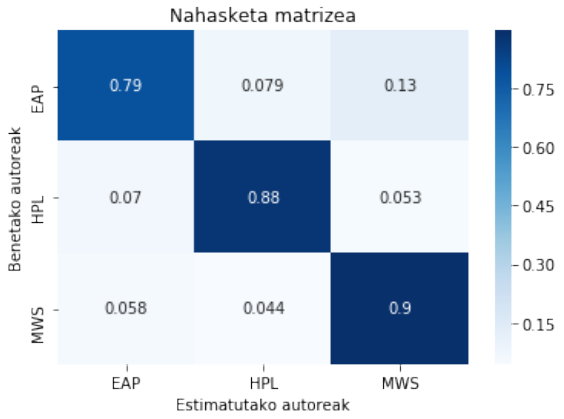
3.2 Emaitza numerikoak

Hitzen agerpen maiztasunaren prozeduraren bidez entrenamendu eta ebaluazioak egin dira. Kasu honetan puntuazio zeinuen eta karaktere larrien agerpenak emaitzetan duten eragina aztertu da, 2.2 atalean azaldu denez. Ondorengo taulan azaltzen dira lortu diren emaitzak.

	Keep Punctuation Marks	Keep Uppercase	Accuracy	Log Loss
1	True	True	0.847	0.657
2	True	False	0.840	0.657
3	False	True	0.843	0.599
4	False	False	0.838	0.589

Taula 3.1: Hitzen agerpen bidez lortzen diren emaitzak, puntuazio zeinu eta letra larrien eragina alderatuz.

Ereduaren zehaztasuna adierazten digun funtzioa galera edo *log loss* funtzioa dugu. Bertan adierazten denez, puntuazio zeinuen agerpen maiztasuna gordetzeak emaitza okertzen du, galera funtzioaren balio altuagoa ematen baitu. Bestalde, karaktere larrien agerpenak eragin txikia du emaitzetan. Honela, *target* edo benetako autoreari ematen zaion probabilitatearen batazbesteko geometrikoa $< P(a_{true}|y_i) >_g =$ 0.52 baliokoa da puntuazio zeinuak erabiltzeko kasuan eta $< P(a_{true}|y_i) >_g =$ 0.55 bestela. Zehaztasunari erreparatuz, lau kasuetan emaitza oso antzekoa lortzen dela ikusten da, %84 doitasuna lortuz. Ondoren klasifikazio prozesuan zehar lortzen den nahasketa matrizea adierazi da.



Irudia 3.2.1: Nahasketa matrizea hitzen agerpen maiztasunaz baliatuz.

3.2.1 irudian hitz agerpen maiztasunak erabiliz lorturiko nahasketa matrizea azaltzen da. Matrize honetan *target*-a zein den jakinik, nahasketa matrizeak idazle bakoitzarekin lortzen dugun zehaztasuna adierazten du. Bestalde, idazle bakoitza besteekin zenbatetan nahasten den ere ikus daiteke. Taulan ikusten denez, EAP idazlearekin beste idazleekin baino doitasun baxuagoa lortzen da. Kasu honetan idazle hau MWS idazlearekin nahasten da gehienbat, kasuen %13 an nahasiz. Adierazgarria da, bestalde, kontrako kasuan ez dela hau gertatzen, hau da, idazlea MWS den kasuan modeloak gutxiagotan aurresaten du EAP klasea.

Prozedura honi dagokionez, zehaztasun handia lortzen dela ikusi da. Hala ere, loss funtzioa hobetzea posible litzake helburu den klaseari ematen zaion probabilitatea handituz. Probabilitateen kalkulurako prozedura honek mugapen batzuk ditu. Alde batetik, probabilitateen emaitzak α parametroaren aukeraketaren menpekotasun nabaria du. Honela, bada, parametro honen balioaren egokitasuna aztertzea beharrezkoa litzake. Bestalde, problema honen inplementazioan zehaztasun arazoak ematea posible da. Izan ere, $P(w_{ij}|a_k)$ hitzen agerpen probabilitateen balioak orokorrean txikiak izatea espero da. Datuen analisian ikusi den moduan, gehien erabilitako hitzen kasuan hauen agerpen maiztasuna $\%6 - \%7$ tartekoa zen, ondorengo hitzen maiztasuna esponentzialki jaitsiz. Hau jakinik 3.1.9 ekuazioko probabilitateak bidertu egiten direnez, balio honek zerorantz jotzen du esaldiaren luzeera handitu ahala. Honen ondorioz errore numerikoak handitzea posiblea da. Hau jakinik, beste teknika batzuk aztertuko dira klasifikazio prozesu berbera burutzeko.

4 Bektore bidezko klasifikazioa

Atal honetan, $P(y_i|a_k)$ probabilitateak esleitzeko beste teknika bat proposatzen da. Honek aurretik sortu den errore numerikoaren arazoa konpontzeko irtenbide bat proposatzen du, horrez gain ebaluazio prozesua azkartzeko aukera emanez.

4.1 Probabilitateen definizio eraldatua

Eredu honetan, 3.1.9 ekuazioan erabili den biderkaduraren ordez probabilitateen batura erabili nahi da. Honela, a_k idazleak y_i esaldia idatzi izaneko probabilitatea ondorengo moduan definitzen dugu [7].

$$P(y_i|a_k) = \sum_{m=1}^{L_i} P(w_{im}|a_k) \tag{4.1.1}$$

Jakina da probabilitatearen kalkulu hau izatez ez dela matematikoki zehatza. Hala ere, idazleak w_{ij} hitza erabiltzeko probabilitatea zenbat eta handiagoa izan, esaldia idazlearena izateko probabilitatea orduan eta handiago egiten du. Probabilitatearen adierazpenaren hautaketa honek, gainera, hainbat abantaila ditu. Alde batetik, esaldiko hitz baten $P(w_{im}|a_k)$ probabilitate nuluak ez du aurreko atalean izan dugun arazoa ekartzen. Honen ondorioz, jada ez dugu α parametroa sartzeko beharrik. Honela, entrenamendu prozesuan idazle batek ez badu w_{im} hitza erabili, honek ez du idazleari buruzko informaziorik sartzen probabilitatearen kalkuluan.

Beste abantaila bat ebaluazio prozesua egiteko abiaduran antzematen da. Izan ere, probabilitate hau batukari moduan adieraztearen ondorioz, kalkulua bektoreen arteko biderkadura eskalar gisa idaztea posible da, 4.1.1 adierazpena berrantolatuz. Ondorengo atalean hau nola egin den azaltzen da.

4.2 Probabilitatearen kalkulua biderkadura eskalarraren bidez

Entrenamendu prozesu honetan idazle guztiek erabilitako hitzen multzo bat osatzen da.

$$W = \{w_l|j = 1, 2, ..., Z\} = \{w_1, w_2, ..., w_Z\} \tag{4.2.1}$$

Definizio honen ondorioz, idazle bakoitzaren W_{a_k} hitz multzoa W-ren azpimultzoa da, hau da: $W_{a_k} \subseteq W$. W multzo honetako hitz guztiak hartuz bektore bat definitzen da. Bektoreko posizio bakoitza W multzoko hitz bati dagokio.

$$\mathbf{V} = [w_1\ ,\ w_2\ ,\ ...\ ,\ w_Z] \tag{4.2.2}$$

Honetan oinarrituz, bi bektore ezberdin definitzen dira: lehenengoa, idazle bakoitza ezaugarritzen duen bektorea da. Bektore honetan hitzen agerpen maiztasuna gordetzen da. Hitzen agerpen maiztasun hau 3.1.4 ekuazioko adierazpen berbera da.

$$\mathbf{V}(a_k) = [f_{a_k}(w_1)\ ,\ f_{a_k}(w_2)\ ,\ ...\ ,\ f_{a_k}(w_Z)] = [P(w_1|a_k)\ ,\ P(w_2|a_k)\ ,\ ...\ ,\ P(w_Z|a_k)] \quad (4.2.3)$$

Bestalde, ebaluatu nahi den y_i esaldiko hitzen maiztasuna ondorengo eran definitzen da:

$$f_{y_i}(w_l) = \frac{\#(w_l|y_i)}{\sum\limits_{w_m \in y_i} \#(w_m|y_i)} \quad (4.2.4)$$

Honenbestez, y_i esaldiari dagokion bektorea maiztasun hauetan oinarrituz definitzen da.

$$\mathbf{V}(y_i) = [f_{y_i}(w_1), f_{y_i}(w_2), ..., f_{y_i}(w_Z)] \quad (4.2.5)$$

Azken bektore hau *sparse* motako bektorea da. Hau da, bektoreko elementu gehienak nuluak dira. Honen arrazoia, y_i esaldiaren luzeera da. Ebaluazio esaldi honen luzeera entrenamenduan agertu den hitz ezberdin kopurua baino txikiagoa dela onartzen dugu ($Z >> L_i$).

Esaldi eta autoreei dagokien bektoreen hauek erabiliz, 4.1.1 ekuazioko probabilitatea biderkadura eskalar gisa kalkulatzen da, era zuzenean.

$$P(y_i|a_k) = \mathbf{V}(y_i) \cdot \mathbf{V}(a_k) = \sum_{w_i \in W} f_{y_i}(w_i) P(w_i|a_k) \quad (4.2.6)$$

Bi bektore hauen definizioaren ondorio zuzenez, normalizaturiko bektoreak dira. Honenbestez, hauen biderkadura eskalarra $[0, 1]$ tartean bornaturik dago. Metodo honen bitartez abstrakzio maila handiago bat lortzen dugu. Bektore hauek Z hitz kopuru haina dimentsioko oinarri bektorial batean kokatzen dira. Idazleetako bakoitza bera ezaugarritzen duen bektore batez definitzen da. Ebaluazio prozesuko esaldiak oinarri bektorial berdinean kokatzen dira. Honela, autore eta esaldien bektoreen biderkadura eskalarrak bektore hauen arteko angelua adierazten digute. Beste era batean esanda, zenbateko antzekotasuna duten esaldiak eta autoreak. Bektore hauek elkarrengandik zenbat eta hurbilago egon, biderkadura eskalarra handiagoa da, eta esaldia eta autorea orduan eta antzekoagoak direla adierazten digu.

Prozedura honen desabantaila nagusia ondorengoa da: esaldiaren hitz kopurua txikia izan arren, ebaluazio prozeduran probabilitatea kalkulatzeko Z luzeerako bektorea sortu behar da esaldiarentzako. Hala ere, ondoren ebaluaketa burutzerako orduan Y entrenamendu sortan esaldi kopurua handia bada hau erabilgarria da, multzo honetako esaldi guztien probabilitateak paraleloan kalkulatzea berahalakoa baita. Izan ere, konputazionalki biderkadura eskalarra burutzeak duen kostua oso txikia da. Hori dela eta, matrizeen arteko biderkaduraren inplementazioa burutzea ere azkarra da. Gainera, behin Y multzoko esaldi guztien $\mathbf{V}(y_i)$ bektoreak lortu ondoren, probabilitate guztien kalkulua honela egiten da matrizeen bidez:

$$M = \begin{array}{c} \\ y_1 \\ y_2 \\ \\ y_I \end{array} \begin{array}{c} \begin{array}{cccc} w_1 & w_2 & ... & w_Z \end{array} \\ \begin{pmatrix} f_{y_1}(w_1) & f_{y_1}(w_2) & \cdots & f_{y_1}(w_Z) \\ f_{y_2}(w_1) & f_{y_2}(w_2) & \cdots & f_{y_2}(w_Z) \\ \vdots & \vdots & \ddots & \vdots \\ f_{y_I}(w_1) & f_{y_I}(w_2) & \cdots & f_{y_I}(w_Z) \end{pmatrix} \end{array} = \begin{pmatrix} \mathbf{V}(y_1) \\ \mathbf{V}(y_2) \\ \vdots \\ \mathbf{V}(y_I) \end{pmatrix} \quad (4.2.7)$$

Matrize honetako lerro bakoitzean ebaluazio sortako esaldi bat gordetzen da *sparse* bektore moduan. Ondorengo matrizean, berriz, idazleak karakterizatzen dituzten bektoreak gordetzen dira zutabeka.

$$A = \begin{array}{c} \\ w_1 \\ w_2 \\ \\ w_Z \end{array} \begin{array}{c} \begin{array}{cccc} a_1 & a_2 & ... & a_n \end{array} \\ \begin{pmatrix} P(w_1|a_1) & P(w_1|a_2) & \cdots & P(w_1|a_n) \\ P(w_2|a_1) & P(w_2|a_2) & \cdots & P(w_2|a_n) \\ \vdots & \vdots & \ddots & \vdots \\ P(w_Z|a_1) & P(w_Z|a_2) & \cdots & P(w_Z|a_n) \end{pmatrix} \end{array} = \tag{4.2.8}$$

$$= \begin{pmatrix} \mathbf{V}^T(a_1) & \mathbf{V}^T(a_2) & \cdots & \mathbf{V}^T(a_n) \end{pmatrix} \tag{4.2.9}$$

Matrize hauen definizio hauek erabiliz ebaluazio prozesuko Y multzoko esaldi guztien probabilitateen kalkulua egiteko matrizeen arteko biderkadura eskalarra egin behar da.

$$M \cdot A = \begin{array}{c} \\ y_1 \\ y_2 \\ \\ y_{L_i} \end{array} \begin{array}{c} \begin{array}{cccc} a_1 & a_2 & ... & a_n \end{array} \\ \begin{pmatrix} P(y_1|a_1) & P(y_1|a_2) & \cdots & P(y_1|a_n) \\ P(y_2|a_1) & P(y_2|a_2) & \cdots & P(y_2|a_n) \\ \vdots & \vdots & \ddots & \vdots \\ P(y_{L_i}|a_1) & P(y_{L_i}|a_2) & \cdots & P(y_{L_i}|a_n) \end{pmatrix} \end{array} \tag{4.2.10}$$

Honenbestez, amaierako matrizearen lerro bakoitzean ebaluazioko esaldi baten probabilitateak lortzen dira. Zutabeetako bakoitzean A multzoko idazleetako bati dagozkion probabilitateak adierazten dira. Azkenik, lerroz lerro 3.1.8 adierazpena erabiltzen da, $P(y_i|a_k)$ eta $P(a_k|y_i)$ probabilitateen arteko lotura egiteko.

4.3 Universal Background Model

Universal Backround Model-a (UBM) idazleekiko independentea den eredu bat da, non hizkuntza batean hitz batek agertzeko duen probabilitatea ereduztatzen den. Probabilitate hau idazleekiko independentea da printzipioz. Sarrerako 2.2 atalean ikusi den moduan, idazle guztiek gehien erabiltzen dituzten hitzak antzekoak dira, orokorrean hauek lokailuak izanez. Hori dela eta, UBM hau erabiliz hitz bakoitzari pisu ezberdina ematen zaio probabilitatearen kalkulua egitean.

Entrenamendu prozesua

4.2.1 adierazpeneko hitzen multzora itzuliz, hitz bakoitzaren idazlearekiko independentea den agerpen probabilitatea esleitzea posible dugu. Horretarako, 3.1.4 ekuazioan egin den antzera egingo da kalkulua, baina agerpen hau idazlearekiko independentea izanez.

$$f(w_i) = \frac{\#(w_i)}{\sum\limits_{w_j \in W} \#(w_j)} = P(w_i|background) \tag{4.3.1}$$

Honi *background*-eko multzoa deritzo. Hau definiturik, hitzen agerpena *background* honekin pisatuko da. Ondorengo frogapena "*High-Level Speaker Verification with support vector machines*"artikuluan oinarritzen da [8].

$$\frac{P(y_i|a_k)}{P(y_i|background)} = \prod_{j=1}^{L_i} \frac{P(w_j|a_k)}{P(w_j|background)} \tag{4.3.2}$$

Azken adierazpen honetan, aurretik egin den moduan hitzen agerpen probabilitateak independenteak direla onartzen da. Zatiketa hau eginez, hitz batek puuntuazio honetan ekarpen handia egingo du baldin eta idazleak hitz hau gehiagotan erabiltzen badu orokorrean erabili ohi dena baino, hau da, $P(w_j|a_k) > P(w_j|background)$ betetzen bada.

Honela, autore bakoitzari emango zaion puntuazio edo *score*-a edo puntuaketa 4.3.2 adierazpenaren logaritmoa izango da. Horrez gain, L_i esaldiaren luzeerarekin zatitzen da.

$$score = \frac{1}{L_i}\sum_{j=1}^{L_i} log\left(\frac{P(w_j|a_k)}{P(w_j|background)}\right) = \sum_{l=1}^{Z}\frac{\#(w_l|y_i)}{L_i} log\left(\frac{P(w_l|a_k)}{P(w_l|background)}\right) \tag{4.3.3}$$

non w_l gaiak idazlearen independenteak diren W multzoko hitzak diren. Honenbestez batukariko lehen zatikiak y_i esaldiko w_j hitzaren agerpen maiztasuna adierazten du. Adierazpen hau logaritmoaren $(x-1)$ inguruko Taylorren garapena eginez hurbiltzen da, honetarako $P(w_l|a_k) \simeq P(w_l|background)$ onartuz.

$$\sum_{l=1}^{Z} f_{y_i}(w_l) log\left(\frac{P(w_l|a_k)}{P(w_l|background)}\right) \simeq \sum_{l=1}^{Z} f_{y_i}(w_l)\left(\frac{P(w_l|a_k)}{P(w_l|background)} - 1\right) =$$
$$= \sum_{l=1}^{Z} f_{y_i}(w_l)\frac{P(w_l|a_k)}{P(w_l|background)} - 1$$

Azken pausu honetan, *background*-eko hitzen probabilitateak normalizaturik daudela onartuz. Simetriagatik izendatzailea bi gaietan banatuz, azken emaitza hau lortzen da:

$$score \simeq \sum_{l=1}^{Z}\frac{f_{y_i}(w_l)}{\sqrt{P(w_l|background)}}\frac{P(w_l|a_k)}{\sqrt{P(w_l|background)}} - 1 \tag{4.3.4}$$

Honenbestez, *Log likelihood ratio weighting* prozedura erabiliz esaldia idazle bakoitzarena izateko puntuazio bana lortzen da. UBM erabiltzeko kasuan kalkulua bektoreen bidezko biderkadura eskalarraren bidez egiten jarraitu daiteke, $\mathbf{V}(a_k)$ eta $\mathbf{V}(y_i)$ bektoreak honela berdefinitzen badira.

$$\mathbf{V}(a_k) = [g(w_1)P(w_1|a_k)\ ,\ g(w_2)P(w_2|a_k)\ ,\ ...\ ,\ g(w_Z)P(w_Z|a_k)] \tag{4.3.5}$$

$$\mathbf{V}(y_i) = [g(w_1)f_{y_i}(w_1)\ ,\ g(w_2)f_{y_i}(w_2)\ ,\ ...\ ,\ g(w_Z)f_{y_i}(w_Z)] \tag{4.3.6}$$

non

$$g(w_j) = \frac{1}{\sqrt{P(w_j|background)}} \tag{4.3.7}$$

4.4 Emaitza numerikoak

Ondorengo atalean, biderketa bidezko eredu zuzena eta UBM metodoarekin lortzen diren emaitzak adierazten dira. Emaitza hauek lortzeko prozedura eranskinetan adierazi da.

4.1 taulan ikus daitekeenez, oinarrizko biderkadura eskalar bitartez zehaztasuna nahiko txikia lortzen da. Puntuazio zeinuak erabiltzeko kasuetan, zehaztasuna %45-ekoa izan da. Puntuazio zeinuak kontuan hartu ez diren kasuan, ordea, balio hau %49-ra igo da. Hortaz metodo honen bidez kalkulua egitean eragina dute puntuazio zeinuek. Nolanahi ere, emaitza hauek ez dira oso

	Keep Punctuation Marks	Keep Uppercase	Accuracy	Log Loss
1	True	True	0.454	1.072
2	True	False	0.448	1.072
3	False	True	0.498	1.078
4	False	False	0.493	1.078

Taula 4.1: Biderkaketa bidezko ereduko emaitzak.

esanguratsuak. Izan ere, informaziorik gabeko ereduan hausaz klasifikatuko bagenu ere, hiru idazle izatearen ondorioz %33-ko doitasuna izango genuke. Galera logaritmikoari dagokionez ere balio hauek aurreko metodoarenak baino okerragoak dira. Log loss-aren balio hauek benetazko autoreari batazbestean $< P(a_{true}|y_i) >_g =$ %34-ko probabilitatea eman zaiola adierazten dute. Honenbestez, biderkadura bidezko eredua erabiliz zuzenean, emaitza ez oso egokiak lortu direla ondoriozta dezakegu.

Ondoren, UBM metodoaren bidez lortzen diren emaitzak adierazi dira, aurreko hauekin alderatu ahal izateko.

	Keep Punctuation Marks	Keep Uppercase	Accuracy	Log Loss
1	True	True	0.843	0.916
2	True	False	0.839	0.921
3	False	True	0.843	0.906
4	False	False	0.836	0.912

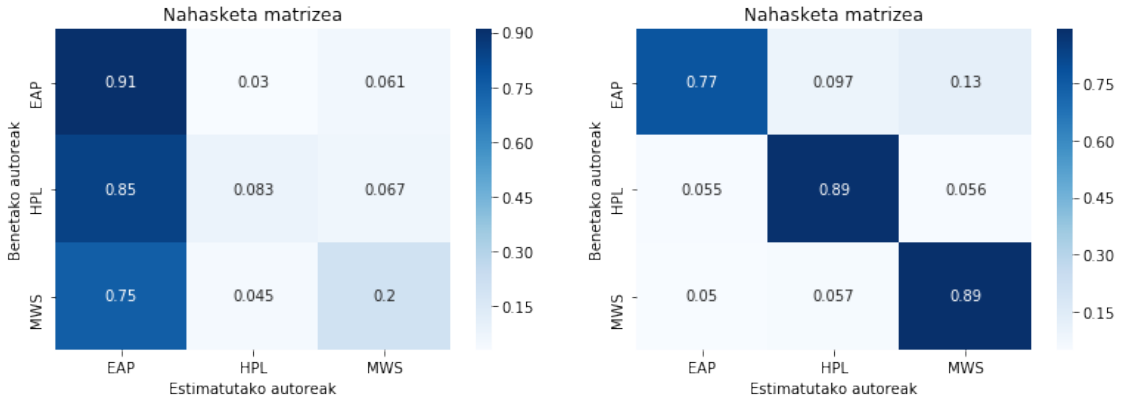
Taula 4.2: UBM metodoaren bidez lorturiko emaitzak.

4.2 taulan ikus daitekeenez, UBM metodoa aplikatuz puntuazio zeinuek eta karaktere larrriek ez dute eragin handirik emaitzetan. Numerikoki gorabeherak badaude ere, ez dago alde nabarmenik prozesaketa ezberdinen artean. Zehaztasunari dagokionez kasu honetan lortzen den balioa $\sim$ %84 ingurukoa da, Bayes-en metodoarekin lortu den doitasun antzekoa hortaz. Log loss-ari dagokionez, ordea, UBM metodo honekin lortzen diren emaitzak Bayes-enak baino okerragoak dira. Kasu hauetan, benetako autoreari ematen zaion batez besteko probabilitate geometrikoa $< P(a_{true}|y_i) >_g =$ %40 da. Hortaz, aurreko kasuan baino probabilitate baxuagoak lortzen dira klase zuzenetan.

Ondoren, 4.4.1 irudietan biderkadura eskalar bidezko eredua eta UBM-arekin lortutako nahasketa matrizeak irudikatu dira.

4.4.1 (a) irudian adierazten denez, biderkadura eskalar zuzenaren bidezko klasifikazio metodo hau ez da gai sailkapena zuzen egiteko. Sarrerako testuaren idazlea edozein izanik ere metodo honen bidez EAP idazlea aurresaten du gehienbat. Honen ondorio da 4.1 taulan lortutako zehaztasuna hain baxua eta *log loss*-a altua izatea. UBM-a erabiliz gero, ordea, zehaztasun nahiko ona lortzen dela ikusten da 4.4.1 (b) irudian. Bertan ikusten denez HPL eta MWS idazleen kasuan %90-eko doitasuna lortzen da. EAP idazlearen kasuan probabilitate hau txikiagoa da, %77 baliokoa hain zuzen. Adierazgarria da Bayes-en oinarrituriko prozeduran zein UBM honetan zehaztasun baxuena EAP idazlearen kasuan lortzen dela, idazle hau gehienbat MWS idazlearekin nahastuz.

Azkenik, 4.3 taulan orain arte aztertutako metodo ezberdinen emaitzak laburbiltzen dira.



(a) Nahasketa matrizea entrenamendu datuetan.

(b) Nahasketa matrizea ebaluazio datuetan.

Irudia 4.4.1: Entrenamendu eta ebaluazio datuetarako nahasketa matrizeak UBM metodoaren bidez.

		Accuracy	Log Loss
1	**Hitzen agerpen maiztasuna**	0.838	0.589
2	**Oinarrizko bektoreen bidez**	0.454	1.072
3	**UBM ereduarekin**	0.843	0.916

Taula 4.3: Aurreko emaitzen alderaketa bektoreen gaineko erregresio logistikoarekin.

Honenbestez, biderkadura eskalarra agerpen maiztasunei zuzenean aplikatuz klasifikazio prozedura ez da egokia. Biderkadura eskalar hau, ordea, erabilgarria da UBM-en bidez pisaturik maiztasun hauek. Bestalde, Bayes-en bidez ere zehaztasun egokia lortu dela ikusi da. Halaber, log loss-a espero genukeena baino handiagoa da. Ondorengo atalean erregresio logistikoa aztertu da, *crossentropy* hau minimizatzeko asmoz.

5 Erregresio logistikoa

5.1 Oinarri teorikoa

Erregresio logistikoa aldagai kategoriko baten emaitza aurresateko teknika bat da, sarrerako datu batzuk erabiliz. Aurrez ikusi den moduan Bayes bidezko probabilitateek eta UBM modeloek zehaztasun egokia ahalbidetzen dute, baina benetako idazle honi ematen zaion probabilitatea ez da handia. Hori dela eta erregresio logistikoa erabiliz probabilitate hauek aldatzea lortu nahi da.

Erregresio logistiko honetan amaierako probabilitatea kalkulatzeko $\mathbf{T} = [t_1, t_2, ..., t_r]$ sarrerako bektore bat kontsideratzen da. Erregresioaren helburua $\boldsymbol{\beta} = [\beta_0, \beta_1, \beta_2, ..., \beta_r]$ parametro bektore bat optimizatzea da, ezaguna den irteerako klasifikazioa ahal bezain egokien hurbiltzeko. Sarrerako $\mathbf{T}$ bektoreaz baliatuz irteerako probabilitatea ondorengo 5.1.1 ekuazioan adierazi den moduan kalkulatzen da.

$$P(\mathbf{T}) = \frac{1}{1 + e^{-(\beta_0 + \beta_1 t_1 + ... + \beta_r t_r)}} \tag{5.1.1}$$

Funtzio hau probabilitateekin erlazionaturiko problemetan erabilgarria da honen irteerako balioa $[0, 1]$ tartean mugatuta egotearen ondorioz. Aurreko 5.1.1 ekuazioa berrantolatuz gero [9]:

$$logit(P(\mathbf{T})) = ln\left(\frac{P(\mathbf{T})}{1 - P(\mathbf{T})}\right) = \beta_0 + \beta_1 t_1 + ... + \beta_r t_r \tag{5.1.2}$$

Adierazpenean logit funtzioa definitzen da. Definizio honekin irteerako probabilitateak ezagunak izanik, probabilitate hauen *logit* funtzioaren erregresio lineala burutzen da $\boldsymbol{\beta}$ bektoreko balioak lortzeko. Bektore honetan β_0 erregresio lineal honetako gai askea da eta bektoreko beste osagaiak sarrerako datuen proportzionaltasun faktorea adierazten du. $\boldsymbol{\beta}$ bektoreko parametroak era iteratiboan ebazten dira. Iterazio hauek *log likelihood* funtzioaren (UBM-a justifikatzean erabili den berbera) minimo lokal bat aurkitzean amaitzen dira [9].

Erregresio logistiko honek eta aurretik definitu diren metodoek ezberdintasun garrantzitsu bat dute. Aurreko predikzio modeloen kasuan $P(y_i|a_k)$ probabilitateak atxikitzeko prozedura ezberdinak deskribatu dira, bakoitzaren justifikazioa eginez. Erregresio logistikoaren kasuan, ordea, entrenamenduaren helburu bakarra jasotzen ditun datuekin *log loss* funtzioaren balio minimoa aurkitzea da. Honek, beraz, gehiegizko entrenamendua edukitzeko arriskua dakar. Hori dela eta, modeloen ebaluaketa egitean entrenamendu eta ebaluazio datuekin lortzen diren emaitzak azaltzen dira, gehiegizko entrenamendua agertzen ote den ikusteko.

Behin hau definituta, lan honetan erregresio logistiko hau sarrerako bi bektore ezberdin erabiliz egin da. Lehen kasuan, sarrerako bektore gisa 3 atalean estimatutako $P(a_k|y_i)$ probabilitateak erabili dira. Erabili den datu basean hiru idazle ezberdin daudenez, lehenengo kasu honetan bektoreak 3 elementuko luzeera du.

$$\mathbf{T} = [t_1 \ , \ t_2 \ , \ ... \ , \ t_r] = [P(a_1|y_i) \ , \ P(a_2|y_i) \ , \ ... \ , \ P(a_n|y_i)] \tag{5.1.3}$$

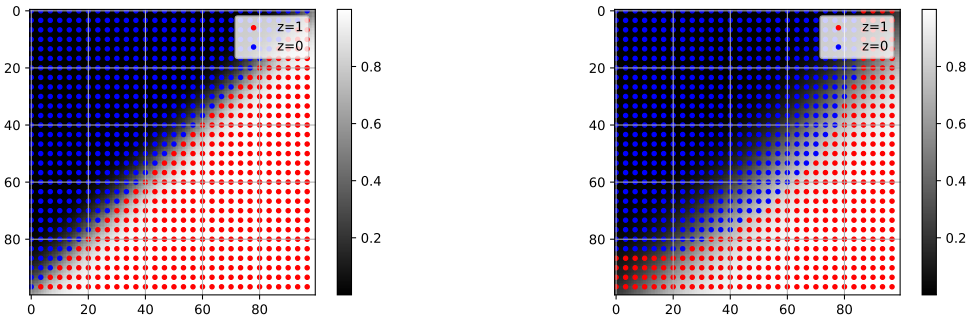
Bigarren kasuan, berriz, 4.3 atalean kalkulatzen diren bektoreak erabili dira, posizio bakoitzean entrenamendu datu baseko hitz baten agerpen maiztasuna egonez.

$$\mathbf{T} = \mathbf{V}(y_i) = [f(w_1)P(w_1|y_i)\ ,\ f(w_2)P(w_2|y_i)\ ,\ \ldots\ ,\ f(w_Z)P(w_Z|y_i)] \tag{5.1.4}$$

Kasu honetan bektorearen tamaina Z entrenamendu prozesuko hitz ezberdin guztien agerpen kopurua da. Hori dela eta bigarren kasu honetan $\boldsymbol{\beta}$ bektorean optimizatu beharreko parametro kopurua era nabarmenean handitzen da.

Erregresio logistikoaren nondik-norakoak ezagutzeko adibide pare bat egin dira, sarrerako bektorea bidimentsionala definituz, klasifikazio ezberdinetan duen eraginkortasuna ikertzeko.

Erregresio logistikoaren adibideak



(a) Erregresio logistikoa sarrerako datuen banaketa lineal kasuan.

(b) Erregresio logistikoa sarrerako datuen banaketa zirkular kasuan.

Irudia 5.1.1: Erregresio logistikoa sarrerako bi datu sorta ezberdinetan.

5.1.1 irudietan erregresio logistikoaren bi adibide azaltzen dira, sarrerako bektorea bi dimentsioko bektorea izanez. Irudiotan erregresio logistikoaren sarrerako datuak puntuen bidez azaltzen dira; irteerako balioa $z = 0$ balioa duten datuak urdinez adierazten dira eta $z = 1$ balioa dutenak aldiz, berdez. Erregresio logistikoaren bidez lortzen diren balioak, aldiz, atzeko kolorearen bidez adierazi dira; beltzez $z = 0$ balioa estimaturiko irteerak adierazi dira eta txuriz $z = 1$ baliokoak. Irteerako probabilitatearen balioa tarteko balioa izanez gero, gris eskalan azaltzen dira. Ezkerreko irudian irteerako $z = 0$ eta $z = 1$ balioak zuzen baten bidez banangarriak dira, eta ikus daiteke erregresio logistikoak kasu honetan banaketa hau ongi burutzen duela. Eskuineko irudiaren kasuan, ordea sarrerako datuek banaketa zirkularra betetzen dute, puntu bat erradio jakin bat baino gertuago badago jatorritik, 0 balioa du irteera idealak, eta kontrako kasuan 1 balioa. Hortaz, irteerako bi klaseen banaketa ez da sarrerako datuen konbinazio lineal baten bidez bereizgarria. Irudian ikusten denez, kasu honetan ere erregresio logistikoak sarrerako datuen konbinazio lineal baten bidez ezartzen du bi klaseen arteko muga klasifikazio hau optimizatuz, baina kasu batzuetan klasifikazio okerra burutzen duela ikus daiteke.

Ondorioz, erregresio logistikoaren bidez sarrerako $\mathbf{Y}$ bektore bat izanik eta honen irteera ideala ezagutuz, erregresio logistikoa erabiltzea posible da, irteera idealeko probabilitatearen *logit* funtzioa kalkulatuz eta honen erregresio lineala eginaz. Honela, $\beta_0 + \beta_1 y_1 + \ldots + \beta_n y_n = 0$ funtzioa n dimentsioko hiperplanoa kontsideratu daiteke, hiperplano honek irteerako 0 eta 1 balioa banantzen dituelarik. Beste era batean esanda, hiperplano honetan kokatzen diren puntuek

$P(\mathbf{Y}) = 0.5$ balioa dute. Hori dela eta, klasifikazio prozesu honetan klase bat sarrerako $\mathbf{Y}$ bektoreko balioen konbinaketa lineal baten bidez banangarria dela onartzen da.

Behin erregresio logistikoak egiten duen kalkuluaren interpretazioa eginda, lan honetako idazleen klasifikazio prozesuan duen eraginkortasuna aztertuko dugu.

5.2 Emaitza numerikoak

Erregresio logistikoa 3 ataleko emaitzekin

Hasteko, Bayes bidezko probabilitateen kasuan lortzen diren $P(a_k|y_i)$ probabilitateak aztertuko ditugu. Hau egiteko emaitzetako lehenengo esaldien predikzioak azaldu dira 5.1 taulan. Erregresio logistikoa aplikatzeko garaian *Scikit-Learn* liburutegiko metodoa erabili da [10]. Ondorengo tauletan erregresio logistikoa aplikatu aurretik eta ondoren aurresaten diren porbabilitateak azaltzen dira.

	EAP	HPL	MWS
0	0.000	1.000	0.000
1	1.000	0.000	0.000
2	1.000	0.000	0.000
3	1.000	0.000	0.000
4	0.017	0.000	0.983
5	0.195	0.800	0.005
6	0.000	0.000	1.000
7	0.119	0.000	0.880

Taula 5.1: Bayes-en bidez kalkulatutako probabilitateen adibidea.

$\Longrightarrow$

	EAP	HPL	MWS
0	0.081	0.880	0.039
1	0.956	0.024	0.020
2	0.956	0.025	0.019
3	0.956	0.024	0.020
4	0.102	0.035	0.863
5	0.931	0.038	0.031
6	0.094	0.033	0.873
7	0.174	0.045	0.781

Taula 5.2: Probabilitate berriak erregresio logistikoa aplikatu ondoren.

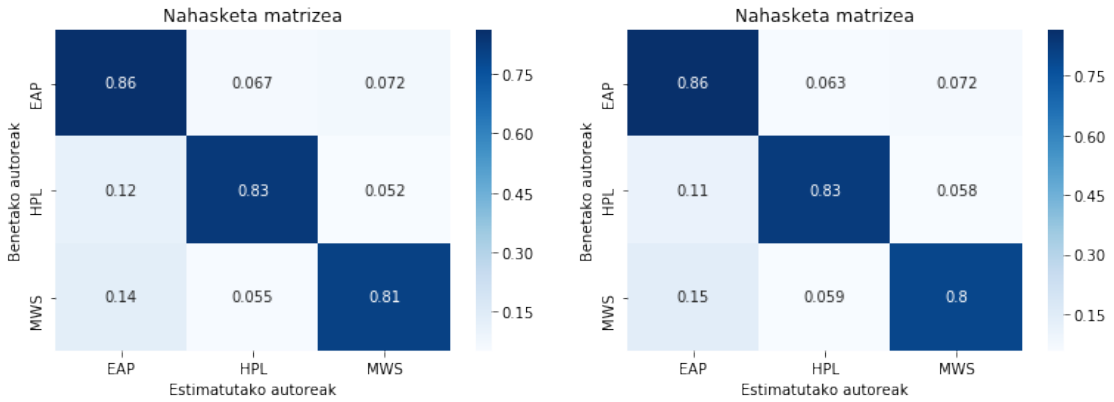
Lehenengo 5.1 taulan ikus daitekenez, Bayes-en oinarrituriko metodoarekin probabilitateak kalkulatuz idazleetako bati probabilitate altua ematen zaio askotan, beste idazleei probabilitateen balio oso txikiak emanez. Hau egokia litzake zehaztasun oso handiko eredu bat izango bagenu. Kasu honetan, ordea, zehaztasuna $\simeq$ %84 denez, klasifikazioa oker egiten diren kasuek *logloss* funtzioaren balioa handitzen dute. Aipatu beharra dago, taulan 0 balioak azaltzen badira ere, honen arrazioa biribilketa izan dela. Probabilitateen balio zehatzak, txikiak izanik ere, ez nuluak dira orokorrean. 5.2 taulan ikus daitekenez, erregresio logistikoa aplikatu ondoren probabilitate hauek muturretako 0 eta 1 balioetatik urruntzen dira. Orokorrean balio altueneko probabilitatea mantentzen jarraitzen du, baina balio numerikoak ez dira hain muturrekoak.

Probabilitate kalkuluen eredu berri honek zehaztasunean eta galera logaritmikoan duen aldaketa adierazten da ondoren. Aurreko 3 atalean ikusi den moduan, Bayes-en oinarrituriko metodoa aplikatuz puntuazio zeinuek eta letra larriek ez zuten eragin nabarmenik emaitzetan. Hori dela eta, kasu horietako bakarrean bakarrik egin da kalkulua (puntuazio zeinuak eta karaktere larriak mantenduz, hain zuzen).

	Accuracy	Log Loss
Entrenamendu datuetan	0.851	0.430
Ebaluazio datuetan	0.847	0.428

Taula 5.3: Erregresio logistikoaren emaitzak Bayes-en oinarrituriko metodoan.

5.3 taulako emaitzak aztertuz, *log loss*-a aurreko emaitzetatik hobetzen dela ondorioztatzen da. Zehaztasuna antzeko balioetan mantentzen da, baina *log loss*-aren balio berriarekin, idazle zuzenari ematen zaion probabilitatearen batazbestekoa $< P(a_{true}|y_i) >_g =$ %65 baliokoa da. Bestalde, ikus daiteke entrenamendu eta ebaluazio datuen artean ez dagoela ia alderik. Hau da, entrenamenduan lortzen diren emaitzak orokortu daiteke esaldi ezezagunen kasura, ez dugu gehiegizko entrenamendurik.



(a) Erregresio logistikoa erabiliz nahasketa matrizea entrenamendu datuetan.

(b) Erregresio logistikoa erabiliz nahasketa matrizea ebaluazio datuetan.

Irudia 5.2.1: Entrenamendu eta ebaluazio datuetarako nahasketa matrizeak erregresio logistikoa probabilitateei aplikatuz.

5.2.1 irudietan aurretik ondorioztaturikoa indartzen da: entrenamendu eta ebaluazio datuetan lortzen diren emaitzak oso antzekoak dira. Horrez gain, idazlez-idazle doitasuna aldatzen bada ere, hauen arteko aldea ez da handia.

Ondoren UBM modeloarekin sortu diren bektoreetan oinarrituko gara erregresio logistikoa egiteko.

Erregresio logistikoa bektoreekin

Ondorengo taulan, UBM metodoan definitu diren bektoreen bidezko erregresio logistikoan lortutako emaitzak azaltzen dira, aurretik lortutako emaitzekin alderatuz.

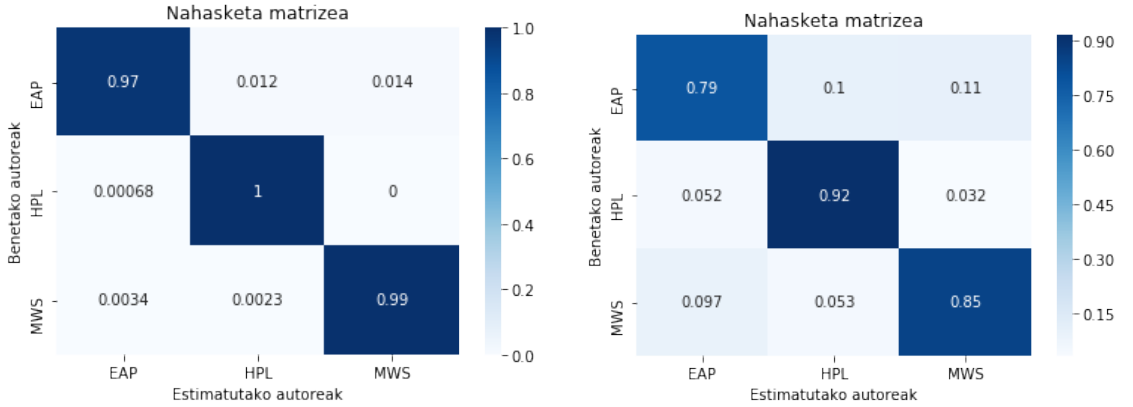
5.4 taulan ikus daitekeenez, kasu honetan entrenamendu prozesuan lortzen den zehaztasuna askoz handiagoa da, %98.7-ko doitasuna lortuz. Autore zuzenari batezbestean ematen zaion

	Accuracy	Log Loss
Entrenamendu datuetan	0.987	0.151
Ebaluazio datuetan	0.839	0.450

Taula 5.4: Erregresio logistikoaren emaitzak Bayes-en oinarrituriko metodoan.

probabilitatea $< P(a_{true}|y_i) >_g =$ %86 baliokoa da. Hala ere, emaitza hau ez da problema nagusira orokortzen; ebaluazio datuetan %83.9 zehaztasuna eta $< P(a_{true}|y_i) >_g =$ %63.8 lortzen da. Beraz, kasu honetan gehiegizko entrenamendua eman da.

Kasu honetan entrenamendu datuetan emaitza hain onak lortzeko arrazoia optimizatzeko eskuragarri dagoen parametro kopurua da. Prozedura hau jarraituz Z hitz kopuru totala haina parametro daude optimizatzeko. Hori dela eta, erregresioa erraz molda daiteke entrenamendu datuetako esaldi guztietan emaitza onak lortzeko, baina honek ez du ziurtatzen generalizatzeko gai denik. Ondorengo irudietan nahasketa matrizeak adierazi dira entrenamendu eta ebaluazio datuetarako.



(a) Bektoreekiko erregresio logistikoa erabiliz nahasketa matrizea entrenamendu datuetan.

(b) Bektoreekiko erregresio logistikoa erabiliz nahasketa matrizea ebaluazio datuetan.

Irudia 5.2.2: Entrenamendu eta ebaluazio datuetarako nahasketa matrizeak erregresio logistikoa bektoreei aplikatuz.

5.2.2 irudietako nahasketa matrizeetan garbi ikusten da gehiegizko entrenamenduaren agerpena. Entrenamendu datuei erreparatuz, esaldien ia kopuru osoa egoki klasifikatzeko gai da erregresio hau, baina emaitza on hauek ez dira ebaluazio datuetara zabaltzen.

Aurreko guztia kontuan hartuta, erregresio logistikoaren bidez *categorical crossentropy*-a hobetu dela ikusten da. Klasifikazio prozesu honekin amaitzeko sare neuronalen erabilpena aztertuko da, hauek hitzen kasuan nola erabili aztertzen delarik.

Laburbiltzeko, orain arte lortu diren emaitzak 5.5 taulan adierazi dira.

		Accuracy	Log Loss
1	**Hitzen agerpen maiztasuna**	0.838	0.589
2	**Oinarrizko bektoreen bidez**	0.454	1.072
3	**UBM ereduarekin**	0.843	0.916
	Erregresio logistikoa probabilitateetan		
4	Entrenamendu datuetan	0.851	0.430
5	Ebaluazio datuetan	0.847	0.428
	Erregresio logistikoa bektoreetan		
6	Entrenamendu datuetan	0.987	0.151
7	Ebaluazio datuetan	0.839	0.450

Taula 5.5: Aurreko emaitzen alderaketa bektoreen gaineko erregresio logistikoarekin.

6 Sare Neuronalak

Sare neuronalak (*Artificial Neural Networks*) gizakion burmuinaren egituran inspiraturiko egitura abstraktuak dira, hauen eginkizuna zeregin konkretu bat burutzea izanik, honetarako bere barne-egitura moldatuz neuronen arteko konexio bitartez [11]. Giza burmuinean gertatzen den antzera, sare neuronala neurona deritzen oinarrizko unitateekin osatzen da. Hauen arteko elkarloturak direla medio, sarrera edo estimulu baten aurrean sareak erantzun bat ematen du.

Sare neuronalen sorrera Warren McCulloch eta Walter Pitts ikerlariek ekarri zuten egitura honen lehen modeloarekin 1943.urtean [12]. 1958.urtean **Mark I Perceptron** makina sortu zuten Frank Rosenblatt eta honen MIT-ko lankideek [13]. Honek potentziometroen bidez, 20×20 tamainako irudietatik abiatuz eskuz idatziriko zifrak irakurtzen zituen [11]. Hala ere, sare neuronalak konputazionalki exigenteak izanik, 80.hamarkada amaieran hasi ziren ospea lortzen.

6.1 Neuronak

Neurona bat sare neuronalean eragiketak egiten dituen funtsezko egitura da. Neurona bakoitzak ondoko egitura du:

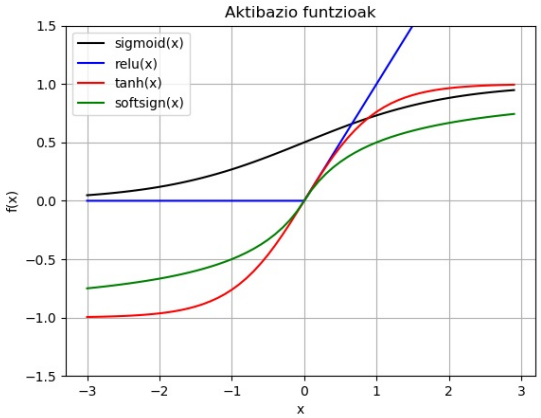
- Sarrerako $\mathbf{X} = [x_1, x_2, ..., x_n]$ seinale bat, orokorrean bektoretzat interpreta daitekeena. Sare neuronalaren sarrerako datuak izan daitezke, edota beste neurona batzuetatik eratorritako seinaleak.
- Propagazio funtzioa. Konbenioz sarrerako seinalearen pisaturiko konbinazio lineala izan ohi da, baina ez du zertan hau izan behar. Neurona honek sarrerako seinale guztiak konbinatzen dira honenbestez neuronaren seinale bakarra bilakatuz.
- Sarrerako seinaleen pisuak: $\mathbf{W} = [w_1, w_2, ..., w_n]$ bektorea. Konbinazio linealean sarrerako seinaleetako bakoitzak duen pisua zehazten du. Honela, propagazio funtzioa 6.1.1 ekuazioan adierazirikoa da.

$$o_i = b_i + \sum_{j=1}^{n} w_j x_j = b_i + \mathbf{W} \cdot \mathbf{X} \tag{6.1.1}$$

  Funtzio honetako b_i gaiari *"bias"* edo *"threshold"* gaia deritzo, eta neurona batek aktibatua edo desaktibatua izateko duen joera adierazten du. Gai hau neuronaren gai askea da eta, pisuen modura, entrenamendu prozesuan zehar egokitzen da. Propagazio funtzioak ez du nahitaez pisaturiko konbinazio lineal hau izan behar eta posible litzake beste funtzio baten bidez adieraztea. Hala ere, funtzio hau hedatuena da konputazionalki merkea izatearen ondorioz.

- Aktibazio funtzioa. Propagazio funtzioa aktibazio funtzio honetatik igarotzen da, neuronaren irteerako seinalea bihurtuz. Aktibazio funtzio hau ez-lineala da. Izan ere, konbinazio linealak diren hainbat o_i funtzio izanik hauen konbinazio lineala beste konbinazio lineal bat da. Hortaz, aktibazio funtzioaren faltan neurona ugari izanik ere, ezingo lirateke portaera ez-linealak deskribatu.

Aktibazio funtzio hauek ugariak badira ere, ohikoenak hauek dira: sigmoid, tangente hiperbolikoa, relu, exponentziala... Ondorengo 6.1.1 irudian agertzen dira hauek.



Irudia 6.1.1: Aktibazio funtzio ezberdinen irudikapena.

$$sigmoid(x) = \frac{1}{1+e^{-x}} \tag{6.1.2}$$

$$relu(x) = \begin{cases} 0 & , x < 0 \\ x & , x \geq 0 \end{cases} \tag{6.1.3}$$

$$tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \tag{6.1.4}$$

$$softsign(x) = \frac{x}{|x|+1} \tag{6.1.5}$$

$$softmax(\mathbf{X})_j = \sigma(\mathbf{X})_j = \frac{e^{x_j}}{\sum_{k=1}^{n} e^{x_k}} \tag{6.1.6}$$

Softmax funtzioa *sigmoid* funtzioaren orokorpena da n irteera ezberdineko sistemaren kasurako. Askotan, neurona baten irteerako balioari interpretazio probabilistikoa eman nahi bazaio funtzio hau erabili ohi da irteerako balioa $[0, 1]$ izateaz gain, funtzioaren definizioa beragatik normalizatua egotearen ondorioz. Honenbestez, sare neuronalaren irteerako neuronen geruzan funtzio hau erabiltzea ohikoa da irteeratzat probabilitateak nahi diren kasuetan.

Aurreko guztiaren ondorioz, $f(x)$ aktibazio funtzioa duen neurona baten irteera:

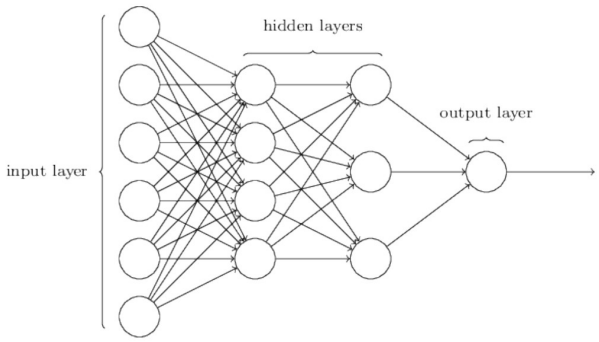
$$o(\mathbf{X}) = f\left(b_i + \sum_{j=1}^{n} w_j x_j\right) \tag{6.1.7}$$

6.2 Sare neuronalen egitura

Behin neuronen barneko egitura aztertu ondoren neuronen arteko konexioak nola egiten diren aztertuko da atal honetan, sare neuronalen egiturak osatzeko helburuarekin.

6.2.1 *Feedforward* sareak

Sare neuronalak hiru geruza mota nagusietan oinarritzen dira: sarrera, geruza ezkutuak eta irteera. Sarrerako geruza sare nueronalak jasotzen duen kanpoko estimulua da, kasuan kasuko ebatzi nahi den problemari buruzko sareak jasotzen duen informazioa. Hauek neurona bezala interpreta daitezke, neurona hauen irteerak $\mathbf{X}$ bektoreko gaiak ezarrita. Ondoren, ezkutuko geruzak kokatzen dira. *Feedforward* egituren kasuan, geruza bakoitzak sarreratzat aurreko geruzako neuronen irteerako balioak ditu. Egitura honen adibide bat 6.2.3 irudian azaltzen da. Adibide honetan sarrerako geruzak 6 neurona ditu; ondoren, lau eta hiru neuronako geruza ezkutuak azaltzen dira; eta, azkenik, irteerako geruza, kasu honetan neurona bakarraren bidez adierazi dena.



Irudia 6.2.1: *Feed Forward* sareen egitura eskematikoa [14].

Honela, neurona bakoitzeko irteerako balioa 6.1.7 ekuazioaren bidez kalkula daiteke zuzenean. *Feedforward* sareen kasuan, beraz, neuronen irteera geruzaz-geruza kalkulatzen da era sekuentzialean, irteerako geruzako balioak kalkulatu arte. Sareko l. geruzako j. neuronaren propagazio funtzioa ondorengo ekuazioak adierazten du [11]:

$$o_j^l = b_j^l + \sum_{i=1}^{n} w_{ij} x_i^{(l-1)} \tag{6.2.1}$$

non $x_i^{(l-1)}$ gaiak $(l-1)$. geruzako i. neuronaren irteera adierazten duen. Propagazio funtzio hauek, ondoren, aukeraturiko aktibazio funtzioan ebaluatzen dira ondorengo geruzarako sarrerarako. Azkenik, l geruzako neuronaren irteera aktibazio funtziotik igarotzen da ondorengo geruzari pasatzeko.

$$x_j^l = f\left(b_j^l + \sum_{i=1}^{n} w_{ij} x_i^{(l-1)}\right) \tag{6.2.2}$$

6.2.2 Sare rekurrenteak

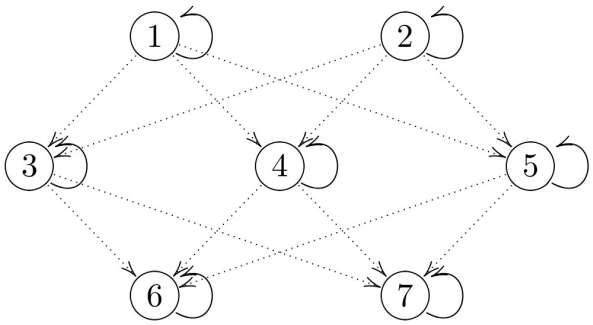
Aurreko ataleko *Feedforward* sareetan, geruza bakoitzeko neuronaren balioak aurreko geruzako neuronen menpekotasuna zuen soilik. Posiblea da, ordea, beste geruza batzuetako neuronen menpekotasuna izatea. Posiblea da, izatez, geruza berbereko beste neurona batzuen irteeraren menpekotasuna izatea neurona batek. Sare rekurrente mota ugari daude, baina azken mota honetan zentratuko gara.

Feedforward sareen arazoetariko bat sekuentzien detekziorako diseinaturik ez egotea da. Ondorioz, denboraren menpekotasuna duten problemen modelizaziorako ez dira egitura egokienak. Esaldiak sekuentzialak izanik, denboraren menpekotasun hau ezaugarritu dezaketen egiturak izatea interesatzen zaigu.

Sare neuronal rekurrenteak (*Recurrent Neural Networks*, RNN) denboraren menpekotasuna modelizatzen duten sareak dira. Hauek, orokorrean, $(t-1)$ aldiunerarteko informazioa jasotzen dute t aldiuneko irteera kalkulatzeko. Kasu honetan, t aldiuneko neurona baten propagazio funtzioaren kalkulua ondorengo 6.2.3 ekuazioan adierazirikoa da.

$$o_j^l(t) = b_j^l + \sum_{i=1}^{n^{(l-1)}} w_{ij} x_i^{l-1}(t) + \sum_{j=1}^{n^{(l)}} v_j x_j^l(t-1) \tag{6.2.3}$$

non $n^{(l-1)}$ eta $n^{(l)}$ aurreko geruzaren eta geruza honen neurona kopurua diren hurrenez hurren. Adierazpen honetako beste terminoak: $a_i^{l-1}(t)$ gaia $(l-1)$. geruzako i. neuronaren irteera t aldiunean; a_j^l gaia l. geruzako j. neuronaren irteera $(t-1)$ aldiunean; N gaia $(l-1)$ geruzako neurona kopurua; M gaia l. geruzako neurona kopurua; w_{ij} gaia $(l-1)$ geruzako i neurona eta l geruzako j neuronaren loturaren pisua; v_j gaia l garren geruzako j neuronaren $(t-1)$ aldiuneko irteeraren balioak t aldiuneko balioa kalkulatzeko duen pisua. Honenbestez, geruza bat errekurtsiboa bada, $(t-1)$ iterazio burutu behar dira geruzaren t aldiuneko irteera kalkulatu arte, ondorengo geruzaren kalkulura igarotzeko.

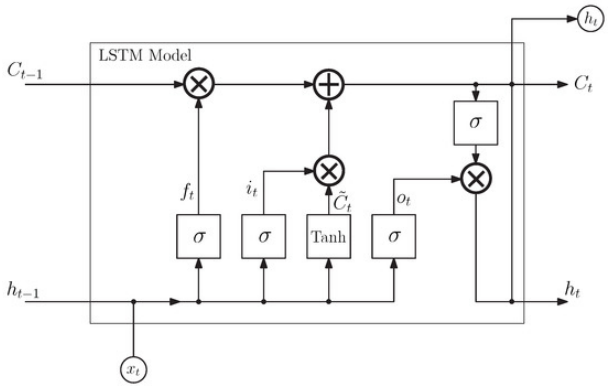


Irudia 6.2.2: Sare errekurrente baten eredua [11].

6.2.2 irudian sare errekurrente baten adibidea azaltzen da, neurona bakoitzak bere irteerako balioa sarrera gisa izanez. Adibide honetan neurona bakoitzak bere buruarekiko errekurrentzia badu ere, hau ez da zertan honetara mugatu behar, geruza berbereko beste neuronen irteerak sarrera gisa izatea posible izanez.

6.2.3 LSTM egitura

LSTM egitura (*Long Short-Term Memory*) problema sekuentzialetarako diseinaturiko egitura da, 1997. urtean proposaturikoa [15]. Egitura honek denbora tarte luzeetarako memoria ahalbidetzen du.



Irudia 6.2.3: LSTM egituraren irudikapena [16].

Ondorengo ekuazioetan LSTM egiturako funtzio bakoitzaren kalkulua nola burutzen den azaltzen da [17].

$$\begin{cases} i(t) = \sigma\left(W^{(i)}x^{(l-1)}(t) + U^{(i)}h^{(l)}(t-1) + b^{(i)}\right) \\ f(t) = \sigma\left(W^{(f)}x^{(l-1)}(t) + U^{(f)}h^{(l)}(t-1) + b^{(f)}\right) \\ o(t) = \sigma\left(W^{(o)}x^{(l-1)}(t) + U^{(o)}h^{(l)}(t-1) + b^{(o)}\right) \\ \tilde{c}(t) = tanh\left(W^{(c)}x^{(l-1)}(t) + U^{(c)}h^{(l)}(t-1) + b^{(c)}\right) \\ c(t) = f(t) \cdot c(t-1) + i(t) \cdot \tilde{c}(t) \\ h(t) = o(t) \cdot tanh(c(t)) \end{cases} \tag{6.2.4}$$

Egitura honen seinale ezberdinak azter ditzakegu, egituraren nondik norakoak ulertzeko.

- $i(t)$ gaiari *input gate* deritzo eta honek LSTM neuronaren egoera eguneratzea du helburu. Honetarako, $(t-1)$ aldiuneko neuronaren irteerarekin berrelikatzen da: $h(t-1)$. Irteera honi $U^{(i)}$ pisuak aplikatzen zaizkio. Pisu hauen eginkizuna aurreko egoeretatik aukeraketa bat burutzea da, eguneraketa prozesuan zein informazio den garrantzizkoa aukeratuz. Horrez, gain, $x(t)$ seinalearen bidez sekuentziaren t aldiunean jasotzen den informazio berria gehitzen da, hauetan ere $W^{(i)}$ pisuak aplikatuz.

- $f(t)$ gaiari *forget gate* deritzo eta honek neuronaren aurreko gelaxka egoeratik zein informazio mantendu eta zein ahaztu zehazten du. Sarrerako atean bezala, ate honetan ere sarrerako terminoei pisuak atxikitzen zaizkie ondoren σ funtziotik pasatuz, irteera $[0, 1]$ tartean bornaturik egon dadin.

- $\tilde{c}(t)$ gaia sare neuronalen ohiko egituraren antza gehien duen gaia da, honen irteerako balioak tangente hiperbolikoaren bidez $[-1, 1]$ tartean bornaturik daudelarik. Sarrerako seinaleetatik gelaxkaren egoeran zein terminok ekarpen positiboa egin eta zein terminok ekarpen negatiboa zehazten du.

- $c(t)$ gaiari *cell state* deritzo, hau da, gelaxka edo neuronaren egoera gordetzen du balio honek. t aldiune jakin bateko gelaxkaren egoera kalkulatzeko, gelaxkaren $(t-1)$ aldiuneko egoerari *forget* gaia aplikatzen zaio, honela aurreko egoeretatik zeintzuk mantendu eta zeintzuk gorde erabakiz. Hau ondoren *input*-eko t aldiuneko informazio berriarekin eguneratzen da, honela aldiune honetako gelaxkaren egoera berria definituz.

- $o(t)$ gaiari *output gate* deritzo, honek LSTM neuronen geruzaren irteerako $h(t)$ gaiak kalkulatzen dituelarik. Azken hau kalkulatzeko, 6.2.4 ekuazioetako azken ekuazioan azaltzen denez, neuronaren *cell state*-a eta *output gate* gaia biderkatzen dira. Iterazioei dagokienez, t amaierako aldiuneko iteraziora iritsi arte, sarea berrelikatu egiten da, hurrengo aldiuneko irteera kalkulatuz, eta iterazio hauen amaieran lortzen diren azken balioak dira sare neuronaleko ondorengo geruzak jasotzen dituen balioak.

Egitura hau neurona arrunta baino konplexuagoa da egitura aldetik. Hala ere, sistemako termino bakoitzaren agerpena justifikatzea posible da. Neurona mota honen abantaila nagusia, luzerako memoria izatea da. Egitura aldetik, ordea, ohiko neuronak baino konplexuagoak dira.

6.3 Sare neuronalen entrenamendua

Orain arte sare neuronalak aztertzean neuronen barne egitura eta neuronen arteko konexioa aztertu da. Halaber, sare neuronalak erabilgarri izan daitezen eta hauen entrenamendua posible izan dadin beharrezkoa da egitura ikasketa prozesuan zehar nola moldatzen den aztertzea.

6.3.1 Kostu funtzioak

Azterketa honetan gainbegiraturiko entrenamendua erabili da (*Supervised Learning*). Entrenamendu mota honen ezaugarri nagusia, sare neuronalaren irteera ideala ezaguna izatea da, hau da, jakina da kasu bakoitzean zein den sare neuronalaren irteera egokiena, klasifikazio prozesua kategorikoa izatearen ondorioz. Honenbestez, entrenamendu prozesuan jakinik esaldi baten egiazko autorea a_j dela, irteera ideala $Z = (z_1 = 0, z_2 = 0, ..., z_{j-1} = 0, z_j = 1, z_{j+1} = 0, ..., z_n = 0)$ bektoreak adierazten du; irteera lortuz gero idazle egokiari %100-eko probabilitatea emango genioke eta nulua beste guztiei.

Sare neuronalaren irteera ideal hau ezaguna izanik, sarearen funtzionamenduaren egokitasuna edo ez-egokitasuna definitzen duen funtzioa zehaztea posiblea da; beste era batean esanda, zein urruti dagoen sistema bere portaera idealetik. Kostu funtzio hau sarearen diseinuan erabakitzen da. Prozedura honen kasuan, 2.4 atalean definituriko *log loss* funtzioa erabili da. Honenbestez, kostu funtzio honek ematen duen informazioaz baliatuz neuronen arteko konexioen aldaketa burutzen da, funtzio hau minimizatzeko asmotan.

6.3.2 Ikasketa prozesua: gradiente jaitsiera

Aurreko 6.1 atalean adierazi den bezala, neuronaren irteera aldatzeko W pisuen bektoreko balioak aldatzea beharrezkoa da, hau da, neurona bakoitzak sarrean dituen datuetako bakoitzari ematen zaion pisua aldatzea. Pisu hauek, kualitatiboki, neuronen arteko lotura bakoitzak duen garrantzia adierazten du. Ondorioz, pisu hauen balioak optimizatu behar dira emaitza egokia lortzeko.

Pisuen optimizazio prozesu honi gradiente jaitsiera edo *Gradient Descent* prozesua deritzo.

Funtzio baten x puntu jakin bateko gradienteak aldaketa handieneko norabidea adierazten du, funtzio honen aldagaiekiko. Honela, kostu funtzioaren gradientea kalkulatuz gero neuronen W sarrerako pisuen funtzioan, pisuen egokitzapena nola egin kalkulatzea posiblea da, kostu funtzioa txikiagoa izan dadin. Prozedura hau iteratiboki erabiltzen da, honenbestez, kostu funtzioaren minimo lokal bat bilatzeko [1].

- Hasteko, pisuei hasierako balio bat ematen zaie, era aleatorio batean adibidez. Entrenamendurako datu sorta bat ebaluatu ostean, hauen kostu funtzio totala kalkulatzen da.

- Ondoren, kostu funtzio honen gradientea kalkulatzen da [18].

$$\nabla f(\mathbf{W}) = \left(\frac{\partial f}{\partial w_1}, \frac{\partial f}{\partial w_2}, ... \right) \Big|_{\mathbf{W}} \tag{6.3.1}$$

  Honekin, ondorengo ikasketako prozesuan erabiliko den pisuen balio berria kalkulatzen da.

$$\mathbf{W}_{t+1} = \mathbf{W}_t - \eta \nabla f\left(\mathbf{W}_t\right) \tag{6.3.2}$$

  Adierazpen honetako η parametroari *learning rate* edo ikasketa abiadura deritzo, eta ikasketa prozesu bakoitzean pisuei ematen zaien aldaketa zenbatekoa den neurtzen du. Sare neuronalen parametro bat da. Orokorrean parametroak balio numeriko txikia du, ikasketaren egonkortasuna bermatzeko. Parametro hau txikia izanez gero gradientearen definizioagatik jakina da $\mathbf{W}_{t+1}$ pisuekin kalkulaturiko kostu funtzioa txikiagoa dela $\mathbf{W}_t$ pisuekin lortzen dena baino.

- Azkenik, lortutako kostu funtzioaren balioa nahi bezain txikia lortu bada edo iterazio kopuru bat burutu ostean, prozedura bukatutzat ematen da, sarea entrenatuta dagoela kontsideratuz. Entrenamendu prozesu bukaeran lortzen diren $\mathbf{W}$ pisuen balio hauek gorde egiten dira. Ondoren, ebaluaketa prozesuan sare neuronalari emandako datuei pisu hauek aplikatzen zaizkie; honela, irteerako geruzan lortzen diren balioak sareak klase bakoitzari ematen dizkion balioak dira.

Prozedura honek mugapenak ditu. Alde batetik, sarearen entrenamendua entrenamendu sortako datuetan oinarritzen da soilik, hau da, entrenamendu prozesuan sare neuronalaren egiteko bakarra, izatez, entrenamendu sortara ahal bezain hobekien egokitzea da *log loss* funtzioa minimizatuz. Hortaz, erregresio logistikoan gertatzen den bezala, gehiegizko entrenamendurako arriskua dago. Honekin loturik egokitzapenerako parametro kopurua dugu. Izan ere, entrenamendu prozesurako datu kopurua handia izanik ere, sare neuronalak duen parametro aske kopurua handia bada, beti izango da posible neuronen arteko pisuak aldatuz entrenamendu datu guztietara egokitzea. Hortaz, sarearen diseinuan beti aukeraketa bat egin behar dugu sarearen egituran, gutxiegizko edo gehiegizko entrenamenduak saihesteko.

6.4 Inplementazioa: Emaitza numerikoak

Sare neuronal hauen inplementaziorako *Keras* [19] eta *Tensorflow* [20] liburutegiak erabili dira. Ondoren, inplementazio hau nola burutu den azaltzen da, atalez atal sarea aztertuz. Kodearen inplementazio osoa eranskinetan adierazita dago.

- Testuak klasifikatzeko, 2 atalean azaldu den bezala, lehenik aurre prozesaketa bat aplikatu behar zaio sarrerako datu sortari. Kasu honetan *Keras*-ek inplementatua duen *Tokenizer* aurre prozesaketako klasea erabili da. Klase honek, hasteko, puntuazio zeinu guztiak ezabatzen ditu eta hitz guztiak letra xeheetara itzuli. Hau egin ostean, entrenamenduko hitzetako bakoitzari zenbaki oso bat esleitzen dio $[1, Z]$ tartean [21]. Aukeraketa hau eginda, x_j entrenamenduko esaldietako bakoitza zenbaki oso segida baten bidez adierazten da.

- Aurre prozesaketarekin bukatzeko, esaldi guztien luzeera unifikatu egiten da. Horretarako esaldi luzeenaren tamaina kalkulatzen da. Esaldi laburrak luzatzeko esaldiaren hasieran 0 zifra eransten da behar haina aldiz esaldi guztiek luzeera berbera izan arte. Hau egitea beharrezkoa da, sare neuronalen sarrerako datuen tamaina beti berbera izatea beharrezkoa izatearen ondorioz. Aurre prozesaketa honen amaieran lortzen diren bektoreak dira sare neuronalaren sarrerako $\mathbf{W}$ bektoreak hain zuzen.

- Bektore hauek, ondoren *Embedding* geruzara sartzen dira. Geruza honek sarrerako hitz bakoitza dimentsio jakin bateko bektore gisa adierazten du. Bektore honen tamaina erabiltzaileak erabakitzen du. Geruza hau entreantu egiten da, sare neuronaletako beste geruzak bezala. Entrenamendu honen helburua hitzen antzekotasuna karakterizatzen da *cosine similarity*-aren bitartez. Hau da, bi hitzen bektoreen arteko biderkadura eskalarra zenbat eta handiagoa izan, hitz hauen arteko antzekotasuna handiagoa da [22].

- *Embedding* geruza honen ondoren, LSTM neuronak kokatzen dira. Hauek esaldi bateko hitzei dagozkien bektoreak jasotzen dituzte era sekuentzialean, 6.2.3 atalean azaldu den moduan. Guztira 64 LSTM neurona ezarri dira. Esaldiko hitz guztien bektoreak sartu ondoren geruza hauetara, honen irteera *Feedforward* sare batean sartzen da.

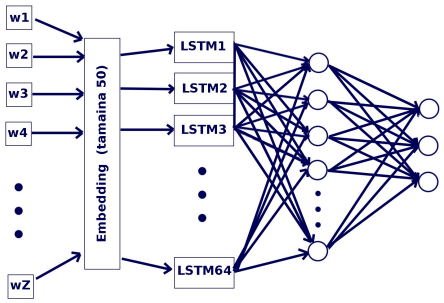
- Ondorengo sare neuronaleko geruza ohiko neurona egituraz osatzen da. Guztira 16 neurona jarri dira honetan, hauen aktibazio funtzioa *relu* aukeratuz. Azkenik, 3 neuronako irteerako geruza kokatzen da. Hauetako bakoitza datu baseko autore baten probabilitatearen adierazle izango da. Azken neurona hauen aktibazio funtzioa *softmax* funtzioa ezarri da, irteeraren interpretazio probabilistikoa egiten laguntzen baitu.

- Azkenik, sare neuronalen geruza ezberdinen artean *dropout*-a ezarri da. Honek entrenamendu prozesuan zehar neurona batzuk ausazko eran desaktibatzen ditu. Honek entrenamendu prozesua hobetzea ahalbidetzen du. Izan ere, *dropout*-ak entrenamendu prozesuan amaierarako informazioa neurona ezberdinetatik iristea ahalbidetzen du. Hau da, hausazko desaktibazio hau gabe, gerta liteke neurona batek garrantzia handia hartzea amaierako erabakia hartzean. Neuronak tarteka desaktibatzearen ondorioz, sareko neurona ezberdinak entrenatzea ahalbidetzen da, honela klasifikazio prozesua hobetuz.

Sare neuronal honen egitura osoa 6.4.1 irudian adierazi da. Hau aplikatuz lortzen diren emaitzak 6.1 taulan azaltzen dira.

	Accuracy	Log Loss
Entrenamendu datuetan	0.944	0.181
Ebaluazio datuetan	0.830	0.459

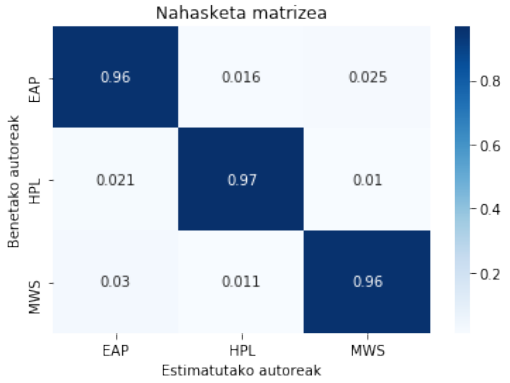
Taula 6.1: Erregresio logistikoaren emaitzak Bayes-en oinarrituriko metodoan.



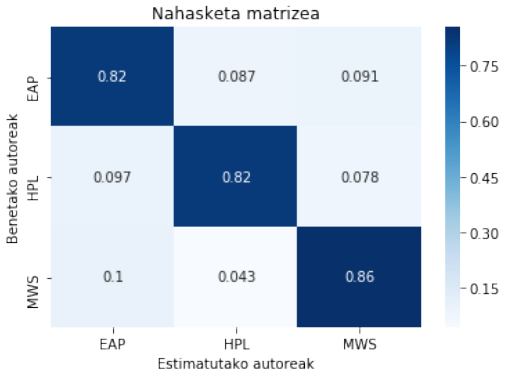
Irudia 6.4.1: Erabili den sare neuronalen egitura irudikapena.

6.1 taulako emaitzetan ikus daitekeenez, kasu honetan ere gehiegizko entrenamendua gertatu da. Entrenamenduko zehaztasuna %94.4 izan bada ere ebaluazio datuetan %83 baliokoa izan da. Honenbestez, sareak datuen kasu handi batean klasifikazio egokia egiten badu ere, sare neuronalen bidezko inplementazio hau erabiliz ez dira aurreko emaitzak era nabarmenean hobetu. *Log loss*-ari dagokionez. Entrenamendu prozesuan benetako autoreei eman zaien batezbesteko probabilitatea $< P(a_{true}|y_i) >_g$= %83.4-koa izan da. Ebaluazio datuetan, berriz, $< P(a_{true}|y_i) >_g$= %63.1.

Azkenik sare neuronalekin lortu diren nahasketa matrizeak irudikatu dira entrenamendu zein ebaluazio klaseetan.



(a) Sare neuronalak erabiliz lorturiko nahasketa matrizea entrenamendu datuetan.



(b) Sare neuronalak erabiliz lorturiko nahasketa matrizea ebaluazio datuetan.

Irudia 6.4.2: Entrenamendu eta ebaluazio datuetarako nahasketa matrizeak LSTM neuronadun sare neuronalak erabiliz.

Nahasketa matrizeetan argi ikusten da gehiegizko entrenamendua burutu dela. Entrenamenduko datu sortara moldaketa ona da, idazleetako kasu bakoitzean %96 inguruko zehaztasuna lortuz. Ebaluazio datuekin ordea, testuen kasu handi batean idazleak identifikatzea lortu bada ere, zehaztasuna %82 ingurura jaitsi da. Beraz, ez da lortu aurreko ataletako emaitzak hobetzea, antzeko balioak lortu badira ere.

Ondorioz, sare neuronalean gehiegizko entrenamendua burutzea erraza dela ikusi da. Honen arrazoia, erregresio logistikoan ikusi den antzera, sare neuronalak optimizatzeko duen parametro kopurua da. Honen ondorioz, sare neuronala erraz molda daiteke entrenamendu prozesuko datu guztietan emaitza egokiak ematera irteeran, baina honek ez du ebaluazio datuetan emaitza ona izatea ziurtatzen. Hala eta guztiz ere, sare neuronalak tresna oso boteretsuak dira problema baten aurrean informazioa eskuratzeko. Lan honetan egiaztatu denez, kode lerro gutxi idatzita, klasifikazio sistema egoki bat sortzea posiblea da, eskuragarri dauden sare neuronalak sortzeko liburutegiekin.

Ondorengo 6.2 taulan lortu diren emaitza guztiak adierazi dira.

		Accuracy	Log Loss
1	**Hitzen agerpen maiztasuna**	0.838	0.589
2	**Oinarrizko bektoreen bidez**	0.454	1.072
3	**UBM ereduarekin**	0.843	0.916
	Erregresio logistikoa probabilitateetan		
4	Entrenamendu datuetan	0.851	0.430
5	Ebaluazio datuetan	0.847	0.428
	Erregresio logistikoa bektoreetan		
6	Entrenamendu datuetan	0.987	0.151
7	Ebaluazio datuetan	0.839	0.450
	Sare neuronalak		
8	Entrenamendu datuetan	0.944	0.181
9	Ebaluazio datuetan	0.830	0.459

Taula 6.2: Aurreko emaitzen alderaketa bektoreen gaineko erregresio logistikoarekin.

7 Ondorioak

Lan honetan testuan oinarrituriko klasifikazio problemen oinarriak aztertu dira, teknika ezberdinak erabiliz eta hauen arteko alderaketa eginaz. Honela, hitzen agerpen maiztasuna, UBM eredua, erregresio logistikoa eta sare neuronalak aztertu dira.

Lehenik, testuaren aurre prozesaketa nola egin aztertu da. Testuen kasuan hau berehalakoa da, hutsunez bananduriko edozein lemma edo hitz kontsideratzen delarik sarrerako datu bat. Hau da, unitate hauetako bakoitza da klasifikazio ereduaren sarrerako datua. Honek, ordea, arazo nabarmen bat dakar. Familia bereko hitzen kasuan, hitzaren edozein deklinazio izanik, hitz ezberdintzat kontsideratzen dira. Honek "*address*"eta "*addresses*"hitzak ezberdintzat hartzen ditu hauen artean erlazio zuzena egonik ere. Argi dago, hortaz, teknika hau erabiltzeko hizkuntzaren arabera emaitzak aldatu daitezkeela, hizkuntza honetan erabiltzen diren deklinabide moten arabera, besteak beste.

Aurre prozesaketa honez baliatuz lehenik hitzen agerpen maiztasunen bidez egin da klasifikazioa. Honetan ikusi denez, Bayes-en teoreman oniarrituz eta hitzen agerpen probabilitateak onartuz klasifikazioa egitea posiblea da, kasu honetan %83.8-ko doitasuna lortuz. Prozedura honen arazo nagusia probabilitateen balio txikiaren ondoriozko errore numerikoan dago. Hori kontuan izanik, bektore bidezko klasifikazioa eta *Universal Background Model*-a aztertu dira. Metodo hauen kasuan probabilitateen kalkulua bektoreen arteko biderketa eskalar moduan egitea posible da. Lehenengoaren kasuan sailkapena ez da egokia izan, orokorrean idazle konkretu bat aurresateko joera izanez sistemak. Bigarrenean, aldiz, idazleen independente den hitzen agerpen maiztasuna erabiliz hitz bakoitzari pisu ezberdina ematen zaio probabilitatearen kalkuluan, honek emaitzak egokiak lortzea ahalbidetuz. Bestalde, bektoreak erabiliz egin diren klasifikazioetan abstrakzio maila handiagoa lortzen da. Esaldi bakoitza bektore gisa karakterizatzen da, eta autore bakoitza beste bektore batez. Espazio bektorial abstraktu honetan bektore hauen arteko biderkadura eskalarra zenbat eta handiagoa izan, orduan eta probabilitate handiagoa definitzen da. Dena den, UBM-aren kasuan ez da lortu galera logaritmikoa hobetzea. Honenbestez, hau optimizatzeko beste teknika batzuk bilatu dira.

Ondoren, erregresio logistikoa aplikatu da, klasifikazioan idazle egokiari ematen zaion probabilitatea handitzeko asmoz. Emaitzetan ikusi denez, erregresio logistikoaren bidez *log loss* ebaluazioko funtzioa nabarmenki hobetzea lortu da. Ondoren, UBM-eko hitzen agerpenetan burutu da erregresio logistikoa, emaitza antzekoak direlarik. Bi metodo hauen arteko ezberdintasun nagusia optimizatzeko eskuragarri dagoen parametro kopurua da. Lehenengo kasuan, hiru autore izanik, 12 parametro zeuden optimizatzeko; UBM-eko bektoreen kasuan, aldiz, entrenamendu prozesuko hitz kopuruarekin proportzionalki hazten da. Honen ondorioz, UBM-ko bektoreen erregresio logistikoa egitean gehiegizko entrenamendua eman da, lehenengo kasuan ez bezala.

Hurrenez, sare neuronalen egitura eta hauen arteko loturak aztertu dira. Hauetan, garrantzia berezia dute LSTM neuronek datu mota sekuentzialen kasuan, memoria duten neurona egiturak izanik eta neurona hauen egituraren atal ezberdinak aztertu dira. Bestalde, sare neuronalen egiturak osatzea erraza da egun eskuragarri dauden liburutegi ezberdinez baliatuz. Ikusi den moduan, sare neuronalak inplementatuz entrenamendu prozesuko datuetara egokitzapena burutzen dute, baina ebaluazio datuekin lortzen diren emaitzak aurreko metodoen antzekoak dira.

Ebaluazio prozesuko emaitzetan erreparatuz zehaztasun handiko eredua lortzea konplexua dela ikusten da; metodo ezberdinekin lortu da zehaztasunak $\sim$ %84 inguruko balioa izatea, baina ezin izan da hau hobetu. Erregresio lineala zein sare neuronalen bidez lorturiko probabilitateen bataz besteko geometrikoa $\sim$ %64 ingurukoa da, UBM edo Bayes-en metodoekin lortzen dena hobetuz. Emaitza hauek beste klasifikazio prozesu batzuetan hobeak izatea espero da (*spam* klasifikazioa, testuak gaika klasifikatzea...). Izan ere, kasu hauetan hitz jakin batzuen agerpenak klasifikazioa burutzeko informazio karga handia du. Testua idatzi duen autorea klasifikatzean, aldiz, informazio karga hau ez da hain handia. Honenbestez, erregresio logistikoa zein sare neuronalen bitartez hitz hauen agerpen maiztasunaren bidez klasifikazioa burutzean emaitza lan honetako emaitzak hobetzea espero da.

Beste aukera bat prozedura hauez baliatuz klasifikazioaren eraginkortasuna hizkuntza ezberdinetan aztertzea litzake. Prozedura honetan zehar hutsunez bananduriko edozein elementu hartzen da esaldiaren unitate gisa. Hortaz, azterketa honetan ingelesezko hiru autoreren esaldiak aztertu badira ere printzipioz ia edozein hizkuntzatan hau egitea posiblea da. Jakina da, gainera, hizkuntza ezberdinetan hitz bakoitzak gordetzen duen informazio kopurua ezberdina dela. Hortaz, hizkuntza ezberdinetaran teknika hauen eraginkortasunean aldaketa txikiak ematea espero daiteke.

Bibliografia

[1] F. Chollet, *Deep Learning with Python*. Manning, Nov. 2017.

[2] F. Sebastiani, "Machine learning in automated text categorization," *ACM computing surveys (CSUR)*, vol. 34, no. 1, pp. 1–47, 2002.

[3] Unknown, "Kaggle - spooky author identification." https://www.kaggle.com/c/spooky-author-identification/data. Azken sarrera: 2019-05-25.

[4] Cross Entropy, "Cross entropy — Wikipedia, the free encyclopedia," 2019. Azken sarrera: 2019-06-19.

[5] L. Buitinck, G. Louppe, M. Blondel, F. Pedregosa, A. Mueller, O. Grisel, V.Ñiculae, P. Prettenhofer, A. Gramfort, J. Grobler, R. Layton, J. VanderPlas, A. Joly, B. Holt, and G. Varoquaux, "API design for machine learning software: experiences from the scikit-learn project," in *ECML PKDD Workshop: Languages for Data Mining and Machine Learning*, pp. 108–122, 2013.

[6] Bayes's theorem, "Bayes's theorem -Wikipedia, the free encyclopedia." Azken sarrera: 2019-06-16.

[7] M. Suzuki and S. Hirasawa, "Text categorization based on the ratio of word frequency in each categories," in *2007 IEEE International Conference on Systems, Man and Cybernetics*, pp. 3535–3540, IEEE, 2007.

[8] W. M. Campbell, J. Campbell, D. A. Reynolds, D. A. Jones, and T. R. Leek, "High-level speaker verification with support vector machines," in *2004 IEEE International Conference on Acoustics, Speech, and Signal Processing*, vol. 1, pp. I–73, IEEE, 2004.

[9] A. Agresti and M. Kateri, *Categorical data analysis*. Springer, 2 ed., 2011.

[10] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[11] D. Kriesel, "A brief introduction to neural networks," *http://www.dkriesel.com*, 2007.

[12] W. S. McCulloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity," *The bulletin of mathematical biophysics*, vol. 5, no. 4, pp. 115–133, 1943.

[13] F. Rosenblatt, *The perceptron, a perceiving and recognizing automaton Project Para*. Cornell Aeronautical Laboratory, 1957.

[14] M. A. Nielsen, *Neural networks and deep learning*, vol. 25. Determination press San Francisco, CA, USA:, 2015.

[15] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[16] K. Yan, X. Wang, Y. Du, N. Jin, H. Huang, and H. Zhou, "Multi-step short-term power consumption forecasting with a hybrid deep learning strategy," *Energies*, vol. 11, no. 11, p. 3089, 2018.

[17] C. Qian, T. He, and R. Zhang, "Deep learning based authorship identification," *Department of Electrical Engineering, Stanford, CA*, 2017.

[18] S. Haynkin, *Neural Networks and Learning Machines*. Pearson Printice Hall, 2009.

[19] F. Chollet *et al.*, "Keras: The python deep learning library," *Astrophysics Source Code Library*, 2018.

[20] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, *et al.*, "Tensorflow: A system for large-scale machine learning," in *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)*, pp. 265–283, 2016.

[21] F. Chollet *et al.*, "Keras documentation: text preprocessing." https://keras.io/preprocessing/text/. Azken sarrera: 2019-06-16.

[22] W. Koehrsen, "Neural network embeddings explained." https://towardsdatascience.com/neural-network-embeddings-explained-4d028e6f0526. Azken sarrera: 2019-06-16.

Eranskinak

A Klase eta metodoen definizioa

A.1 *Prozesaketa* klasea

```
import pandas as pd
import numpy as np

class Prozesaketa:

    spechar = ".,;:"

    def process(self,dataframe,KeepPunctuationCharacters=False,KeepUppercase=False)
    :
        df=dataframe.copy()
        for index,row in df.iterrows():
            row['text']=self.phrase_clean(row['text'],KeepPunctuationCharacters,
    KeepUppercase)
        return df

    def set_special_characters(text):
        self.spechar = text


    def phrase_clean(self,phrase,KeepPunctuationCharacters,KeepUppercase):
        if not KeepUppercase:
            phrase=phrase.lower()
        if KeepPunctuationCharacters:
            for elem in Prozesaketa.spechar:
                phrase=phrase.replace(elem,' '+elem+' ')
            return phrase
        else:
            for elem  in Prozesaketa.spechar:
                phrase=phrase.replace(elem,' ')
            return phrase
```

A.2 *Histograma* klasea

```
from collections import Counter
import pandas as pd
import numpy as np
import math

class Histograma:

# METODO ERAIKITZAILEA

    alpha=1

    def __init__(self,name,text_series):
        self.name = name
        self.author_counter = self.create_counter(text_series)
        self.normalize_histogram()


# ENTRENAMENDURAKO KLASEAK

    def get_counter(self):
        return self.author_counter

    def word_frequency(self,word):
        return self.author_counter.get(word,0) #

    def word_frequency_weighted(self,word,model):
        return self.author_counter.get(word,0)/math.sqrt(model.word_frequency(word)
    ) #

    def create_counter(self,texts):
        inner = Counter(" ".join(texts).split())
        return inner

    def normalize_histogram(self):
        total=sum(self.author_counter.values())
        self.alpha=min([self.alpha,1/total])*0.5
        for key in self.author_counter:
            self.author_counter[key]/=total
        return None

# KLASIFIKAZIORAKO KLASEAK

    def probability(self,phrase,model,ubmboolean=False):
        if ubmboolean:
            ans=1
            for word in phrase.split():
                if (model.word_frequency(word)!=0):
                    ans*=(self.word_frequency_weighted(word,model)+self.alpha)/math
    .sqrt(model.word_frequency(word)+self.alpha)
            return ans
        else:
            ans=1
            for word in phrase.split():
                ans*=(self.word_frequency(word)+self.alpha)
            return ans
```

A.3 *Generator* klasea

```
import numpy as np
import pandas as pd
from Histograma import Histograma
from collections import Counter

class Generator:

# METODO ERAIKITZAILEA

    def __init__(self,dataframe,UseUniversalBackgroundModel=False):
        self.dataframe  = dataframe
        self.UseUniversalBackgroundModel = UseUniversalBackgroundModel
        self.histograms = self.create_histograms(dataframe)
        self.UBM        = Histograma('UBM',dataframe.text)
        #self.KPC = dataframe[1]
        #self.KU  = dataframe[2]

# ENTRENAMENDURAKO KLASEAK

    def create_histograms(self,dataframe):
        inner = {}
        for auth in np.unique(dataframe.author.values):
            df = dataframe.loc[dataframe["author"] == auth]
            inner[auth] = Histograma(auth,df.text)
        return inner

    def set_UseUniversalBackgroundModel(self,UBM):
        self.UseUniversalBackgroundModel = UBM
        return None


# KLASIFIKAZIORAKO KLASEAK


    def get_authors(self):
        v = list(self.histograms.keys())
        v.sort()
        return v

    def probabilities(self,phrase):
        prob={}
        for auth,hist in self.histograms.items():
            prob[auth] = hist.probability(phrase,model=self.UBM,ubmboolean=self.
    UseUniversalBackgroundModel)
        prob = self.normalize_prob(prob)
        return list(prob[a] for a in self.get_authors())

    def normalize_prob(self,prob):
        total=sum(prob.values())
        if (total == 0):
            return prob
        else:
            for auth,hist in prob.items():
                prob[auth]/= total
            return prob

    def predictions(self,test,UBM=False):
        self.UseUniversalBackgroundModel=UBM
        authors = self.get_authors()
        matrix = test.values
        idx = list(test.id)
        result = np.zeros((len(matrix[:,0]),len(authors)))
```

```
        i = -1
        for phrase in test.text:
            i += 1
            result[i,:] = self.probabilities(phrase)
        df = pd.DataFrame(result)
        df.columns = authors
        df['id']=idx
        df = df[['id']+authors]
        return df
```

A.4 *Vector* klasea

```

from Histograma import Histograma
from Generator import Generator
from Prozesaketa import Prozesaketa
from Analisia import testrainsplit , accuracy , logloss
from sklearn.preprocessing import normalize

from sklearn import preprocessing
import pandas as pd
import numpy as np
import math
import time

class Vector:

# METODO ERAIKITZAILEA

    def __init__(self,dataframe,UseUniversalBackgroundModel=True):
        self.UseUniversalBackgroundModel = UseUniversalBackgroundModel
        self.generator = Generator(dataframe,self.UseUniversalBackgroundModel)
        self.vector , self.location = self.get_words(self.generator.histograms)
        self.UBM    = self.ubm_vector(self.generator.UBM)
        self.hist   = self.get_dictvectors(self.generator.histograms)

# ENTRENAMENDURAKO KLASEAK


    def get_words(self,dictionary):
        hist={}
        v = []
        for h in dictionary.values():
            for word in h.get_counter():
                if word not in hist:
                    hist[word] = len(v)
                    v.append(word)
        return v , hist

    def get_dictvectors(self,dictionary):
        h = {}
        for key,histogram in dictionary.items():
            h[key]=self.author_vector(histogram)
        return h

    def author_vector(self,histogram):
        if self.UseUniversalBackgroundModel:
            v = [histogram.word_frequency(self.vector[i])/math.sqrt(self.UBM[i])
```

```
    for i in range(len(self.vector))]
        else:
            v = [histogram.word_frequency(self.vector[i]) for i in range(len(self.
    vector))]
        return np.array(v)

    def ubm_vector(self,histogram):
        v = [histogram.word_frequency(word) for word in self.vector]
        return np.array(v)


# KLASIFIKAZIORAKO KLASEAK

    def get_authors(self):
        v = list(self.hist.keys())
        v.sort()
        return v

    def vectorize_phrase(self,phrase):
        v = np.zeros(len(self.vector))
        for elem in phrase.split():
            if elem in self.location:
                if (self.UseUniversalBackgroundModel):
                    v[self.location.get(elem)]+=1/(math.sqrt(self.UBM[self.location
    [elem]]))
                else:
                    v[self.location.get(elem)]+=1
        v = np.array(v)
        return v

    def phrases_to_matrix(self,dataframe):
        phrases = list((self.vectorize_phrase(ph) for ph in dataframe.values[:,1]))
        return np.matrix(phrases)


    def normalize_predictions(self,matrix):
        return preprocessing.normalize(matrix,axis=1,norm='l1')


    def predictions(self,dataframe):
        data = dataframe.values
        ids = data[:,0]
        authors = np.empty((0,len(self.vector)))
        for author in self.get_authors():
            authors = np.vstack([authors,self.hist[author]])
        authors = np.transpose(authors)
        phrases = self.phrases_to_matrix(dataframe)
        df = pd.DataFrame(self.normalize_predictions(np.dot(phrases,authors)))
        df.columns = self.get_authors()
        df['id'] = ids
        df = df[['id']+self.get_authors()]
        return pd.DataFrame(df)
```

A.5 Aurre prozesaketa eta ebaluaziorako klaseak

```
from collections import Counter
import pandas as pd
import numpy as np
```

```
import math
import time


def logloss(pred_df,real_author_df):
    ans =0
    maxi = 1 - 1e-15
    mini = 1e-15
    authors = list(pred_df.columns)
    authors.remove('id')
    pred = pred_df.sort_values(by=['id'])[authors].values
    real = real_author_df.sort_values(by=['id'])[['author']].values
    for i in range(len(real)):
        value = pred[i,authors.index(real[i])]
        if (value< mini):
            value = mini
        elif(value>maxi):
            value=maxi
        ans+=math.log(value)
    return -ans/len(real)


def accuracy(pred_df,real_author_df):
    authors = list(pred_df.columns)
    authors.remove('id')
    pred = pred_df.sort_values(by=['id'])[authors].values
    real = real_author_df.sort_values(by=['id'])[['author']].values
    v = list((authors[np.argmax(line)] for line in pred))
    total_guessed= sum(v[i]==real[i] for i in range(len(real)))
    return float(total_guessed/len(real))


def testrainsplit(dataframe,test_size,evaluation_size=0.0):
    columns=list(dataframe.keys())
    index=[]
    train = pd.DataFrame(index=index, columns=columns)
    test  = pd.DataFrame(index=index, columns=columns)
    evaluation = pd.DataFrame(index=index, columns=columns)
    test_number=int(min(dataframe.author.value_counts())*test_size)
    evaluation_number = int(min(dataframe.author.value_counts())*evaluation_size)
    for auth in np.unique(dataframe.author.values):
        au = dataframe.loc[dataframe["author"] == auth]
        test=test.append(au.sample(frac=test_size,random_state=0)) #tail
        au = au.drop(au.sample(frac=test_size,random_state=0).index)
        evaluation = evaluation.append((au.sample(frac=evaluation_size/(1-test_size
    ),random_state=0)))
        train=train.append(au.drop(au.sample(frac=evaluation_size/(1-test_size),
    random_state=0).index))
    if (test.shape[0]!=0): test=test.sample(n=test.shape[0]).reset_index(drop=True)
          # Random shuffle
    if (train.shape[0]!=0): train=train.sample(n=train.shape[0]).reset_index(drop=
    True)    # Random shuffle
    if (evaluation.shape[0]!=0): evaluation=evaluation.sample(n=evaluation.shape
    [0]).reset_index(drop=True)
    if (evaluation_size==0.0):
        return (train,test)
    else:
        return (train,test,evaluation)

def results(predictions,real,PunctuationArray,UppercaseArray):
    column3=[accuracy(p,real) for p in predictions]
    column4 = [logloss(p,real) for p in predictions]
    df = pd.DataFrame(index=[i for i in range(len(PunctuationArray))], columns=["
    Keep Punctuation Marks","Keep Uppercase","Accuracy","Logarithmic Loss"])
    df = df.fillna(0.0)
```

```
63      df["Keep Punctuation Marks"]= PunctuationArray
64      df["Keep Uppercase"]= UppercaseArray
65      df["Accuracy"] = column3
66      df["Logarithmic Loss"] = column4
67      return df
```

A.6 Nahasketa matrizeak irudikatzeko klasea

```
1  import numpy as np
2  import pandas as pd
3  import matplotlib.pyplot as plt
4  from sklearn.metrics import confusion_matrix
5  import seaborn as sns
6  from sklearn.preprocessing import normalize
7
8  def plot_confusion_matrix(ytest,ypred, title='Confusion matrix', cmap=plt.cm.Blues)
       :
9      cm=confusion_matrix(ytest,ypred)
10     cm = normalize(cm, axis=1, norm='l1')
11     ax= plt.subplot()
12     sns.heatmap(cm, annot=True, ax = ax,cmap='Blues');
13     ax.set_xlabel('Estimatutako autoreak');ax.set_ylabel('Benetako autoreak');
14     ax.set_title('Nahasketa matrizea');
15     ax.xaxis.set_ticklabels(np.unique(ytest)); ax.yaxis.set_ticklabels(np.unique(
       ytest));
16     plt.show()
```

B Kodearen exekuzioa

Ondorengo ataletan lanean lortu diren emaitzak lortzeko erabili den prozedura adierazi da. Lanean lortutako emaitza eta irudiak kode hau exekutatuz lortzea posible da.

B.1 Datuen analisia

Lehenik, beharrezko klaseen inportazioa egin behar da.

[1]:
```
import matplotlib.pyplot as plt
import pandas as pd
from Histograma import Histograma

train = pd.read_csv('train.csv')
```

Ondoren eskuragarri dugun datu matrizearen tamaina eta honen hasierako lerroak inprimatu dira.

[2]:
```
print(train.shape)
train.head(8)
```

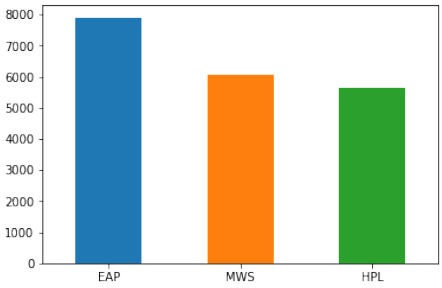
(19579, 3)

	id	text	author
0	id26305	This process, however, afforded me no means of...	EAP
1	id17569	It never once occurred to me that the fumbling...	HPL
2	id11008	In his left hand was a gold snuff box, from wh...	EAP
3	id27763	How lovely is spring As we looked from Windsor...	MWS
4	id12958	Finding nothing else, not even gold, the Super...	HPL
5	id22965	A youth passed in solitude, my best years spen...	MWS
6	id09674	The astronomer, perhaps, at this point, took r...	EAP
7	id13515	The surcingle hung in ribands from my body.	EAP

Taula B.1: Datu basearen egituraren analisia.

Ondoren, idazle bakoitzeko zenbat esaldi ditugun kontatu da.

[3]:
```
authors = train["author"].value_counts()
authors.plot.bar()
plt.xticks(rotation=0)
```

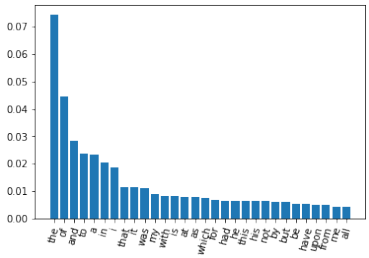


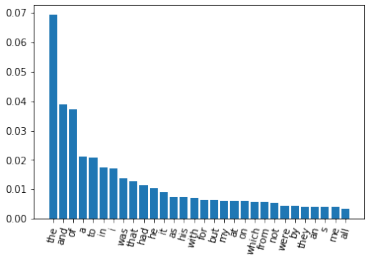
Ondorengo atalean autore bakoitzeko honen hitz histograma bat sortzen da, eta gehien agertzen diren hitzak irudikatzen dira ordenean.

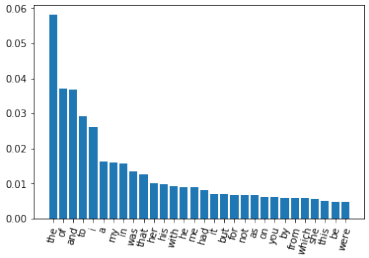
[5]:
```
from Prozesaketa import Prozesaketa


authors = train.author.unique()
hist =[]
for auth in authors:
    p = Prozesaketa()
    sub_df = p.process(train.loc[train.author ==␣
 ↪auth],KeepPunctuationCharacters=False,KeepUppercase=False)
    hist.append(Histograma(auth,sub_df.text))

for words in hist:
    freq = words.get_counter()
    df = pd.DataFrame.from_dict(freq, orient='index',columns=['freq'])
    df= df.sort_values(by=['freq'],ascending=False)
    plt.bar(df.head(30).index,df.freq.head(30))
    plt.xticks(rotation=75)
    plt.show()
```



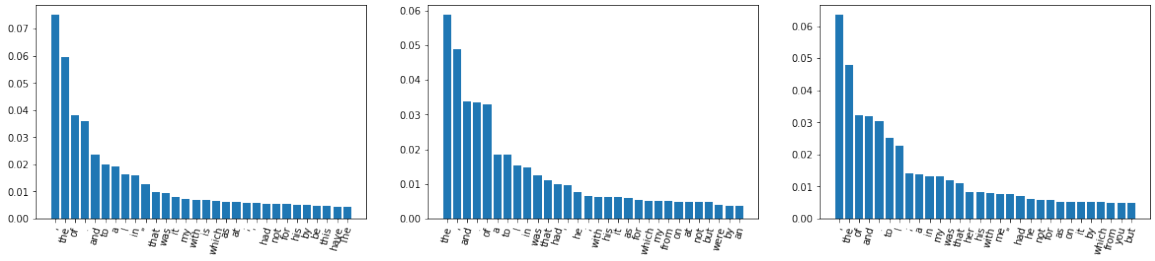




Azkenik, hitz histograma berberak adierazi dira. Oraingoan, ordea, puntuazio zeinu eta karaktere larriak kontuan hartuz.

[6]:
```
authors = train.author.unique()
hist =[]
for auth in authors:
    p = Prozesaketa()
    sub_df = p.process(train.loc[train.author ==␣
 ↪auth],KeepPunctuationCharacters=True,KeepUppercase=True)
    hist.append(Histograma(auth,sub_df.text))
```

```
for words in hist:
    freq = words.get_counter()
    df = pd.DataFrame.from_dict(freq, orient='index',columns=['freq'])
    df= df.sort_values(by=['freq'],ascending=False)
    plt.bar(df.head(30).index,df.freq.head(30))
    plt.xticks(rotation=75)
    plt.show()
```



B.2 Bayes-en bidezko probabilitateen kalkulua

Lehenik, atal honetan behar diren modulu eta klaseen inportazioa egin da.

[1]:
```
from Histograma  import Histograma
from Generator   import Generator
from Prozesaketa import Prozesaketa
from Vector      import Vector
from Analisia    import results,testrainsplit,accuracy,logloss
from ConfusionMatrix import plot_confusion_matrix

import pandas as pd
import numpy as np
import time
```

Ondoren, lan egiteko datuak biltzen dira.

[2]:
```
data = pd.read_csv('train.csv')
train , test = testrainsplit(data,0.15)
ptrain = Prozesaketa()
ptest = Prozesaketa()
```

Atal honetan esaldiei aurreprozesaketa egiten zaie. Lau *dataframe* definitzen dira, bakoitzean entrenamendu eta ebaluazio datuei egiten zaien aurreprozesaketa ezberdina izanez.

[3]:
```
train_data1=ptrain.process(train,
KeepPunctuationCharacters=True,KeepUppercase=True)
train_data2=ptrain.process(train,
```

```
KeepPunctuationCharacters=True,KeepUppercase=False)
train_data3=ptrain.process(train,
KeepPunctuationCharacters=False,KeepUppercase=True)
train_data4=ptrain.process(train,
KeepPunctuationCharacters=False,KeepUppercase=False)

test_data1=ptrain.process(test,
KeepPunctuationCharacters=True,KeepUppercase=True)
test_data2=ptrain.process(test,
KeepPunctuationCharacters=True,KeepUppercase=False)
test_data3=ptrain.process(test,
KeepPunctuationCharacters=False,KeepUppercase=True)
test_data4=ptrain.process(test,
KeepPunctuationCharacters=False,KeepUppercase=False)
```

Ondoren entrenamendu prozesua exekutatzen da, autore bakoitzaren hitz agerpen maiztasunak kalkulatzeko.

[4]:
```
g1 = Generator(train_data1,UseUniversalBackgroundModel=False)
g2 = Generator(train_data2,UseUniversalBackgroundModel=False)
g3 = Generator(train_data3,UseUniversalBackgroundModel=False)
g4 = Generator(train_data4,UseUniversalBackgroundModel=False)
```

Honen ostean, predikzio prozesua egiten da. Ebaluazio prozesurako gorde diren datuak modeloan sartzen dira, honela sarea ebaluatu ahal izateko.

[5]:
```
pred1 = g1.predictions(test_data1)
pred2 = g2.predictions(test_data2)
pred3 = g3.predictions(test_data3)
pred4 = g4.predictions(test_data4)
```

Emaitza numerikoak kalkulatzen dira, ebaluazio esaldietako predikzioak egiazko autoreekin alderatuz.

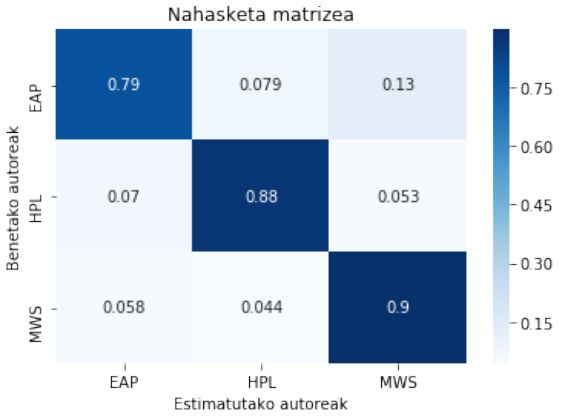
[6]:
```
df=results(
[pred1,pred2,pred3,pred4],
test,
[True,True,False,False],
[True,False,True,False])
df
```

	Keep Punctuation Marks	Keep Uppercase	Accuracy	Log Loss
1	True	True	0.847	0.657
2	True	False	0.840	0.657
3	False	True	0.843	0.599
4	False	False	0.838	0.589

Taula B.2: Hitzen agerpen bidez lortzen diren emaitzak, puntuazio zeinu eta letra larrien eragina alderatuz.

Azkenik, nahasketa matrizea kalkulatu da.

[7]:
```
predicted = pred1.drop(['id'],axis=1).idxmax(axis=1)
aim = test.author
plot_confusion_matrix(aim,predicted)
```



B.3 Bektoreen bidezko probabilitateen kalkulua

Lehenik, betiko lez, beharrezko metodo eta klaseak inportatzen dira.

[1]:
```
from Histograma  import Histograma
from Prozesaketa import Prozesaketa
from Vector      import Vector
from Analisia    import results,testrainsplit,accuracy,logloss
from ConfusionMatrix import plot_confusion_matrix

import pandas as pd
import numpy as np
import time
```

Ondoren, datuak irakurri eta entrenamendu zein ebaluazio datuetan banatzen dira.

[2]:
```
data = pd.read_csv('train.csv')
train , test = testrainsplit(data,0.20)
ptrain = Prozesaketa()
ptest = Prozesaketa()
```

Honen ostean, aurre prozesaketa exekutatzen da. Kasu honetan ere, *dataframe* ezberdinak sortzen dira aurreprozesaketaren arabera.

[3]:
```
train_data1=ptrain.process(train,
KeepPunctuationCharacters=True,KeepUppercase=True)
train_data2=ptrain.process(train,
KeepPunctuationCharacters=True,KeepUppercase=False)
train_data3=ptrain.process(train,
KeepPunctuationCharacters=False,KeepUppercase=True)
train_data4=ptrain.process(train,
KeepPunctuationCharacters=False,KeepUppercase=False)

test_data1=ptest.process(test,
KeepPunctuationCharacters=True,KeepUppercase=True)
test_data2=ptest.process(test,
KeepPunctuationCharacters=True,KeepUppercase=False)
test_data3=ptest.process(test,
KeepPunctuationCharacters=False,KeepUppercase=True)
test_data4=ptest.process(test,
KeepPunctuationCharacters=False,KeepUppercase=False)
```

Honen ostean, entrenamendu prozesua egiten da, bektoreak definituz eta autoreen bektore karakteristikoak lortuz.

[4]:
```
v1 = Vector(train_data1,UseUniversalBackgroundModel=False)
v2 = Vector(train_data2,UseUniversalBackgroundModel=False)
v3 = Vector(train_data3,UseUniversalBackgroundModel=False)
v4 = Vector(train_data4,UseUniversalBackgroundModel=False)

v5 = Vector(train_data1,UseUniversalBackgroundModel=True)
v6 = Vector(train_data2,UseUniversalBackgroundModel=True)
v7 = Vector(train_data3,UseUniversalBackgroundModel=True)
v8 = Vector(train_data4,UseUniversalBackgroundModel=True)
```

Ondoren, predikzio prozesua exekutatzen da, modeloak ebaluazio datuetan aurresaten dituen probabilitateak lortzeko.

[5]:
```
prob1 = v1.predictions(test_data1)
prob2 = v2.predictions(test_data2)
prob3 = v3.predictions(test_data3)
prob4 = v4.predictions(test_data4)

prob5 = v5.predictions(test_data1)
prob6 = v6.predictions(test_data2)
prob7 = v7.predictions(test_data3)
```

```
prob8 = v8.predictions(test_data4)
```

Biderkadura eskalarra erabiliz lortzen diren emaitza numerikoak kalkulatzen dira, zehaztasuna eta galera logaritmikoa kalkulatuz kasu bakoitzean.

[6]:
```
df=results(
[prob1,prob2,prob3,prob4],
test,
[True,True,False,False],
[True,False,True,False])
df
```

	Keep Punctuation Marks	Keep Uppercase	Accuracy	Log Loss
0	True	True	0.454	1.072
1	True	False	0.448	1.072
2	False	True	0.498	1.078
3	False	False	0.493	1.078

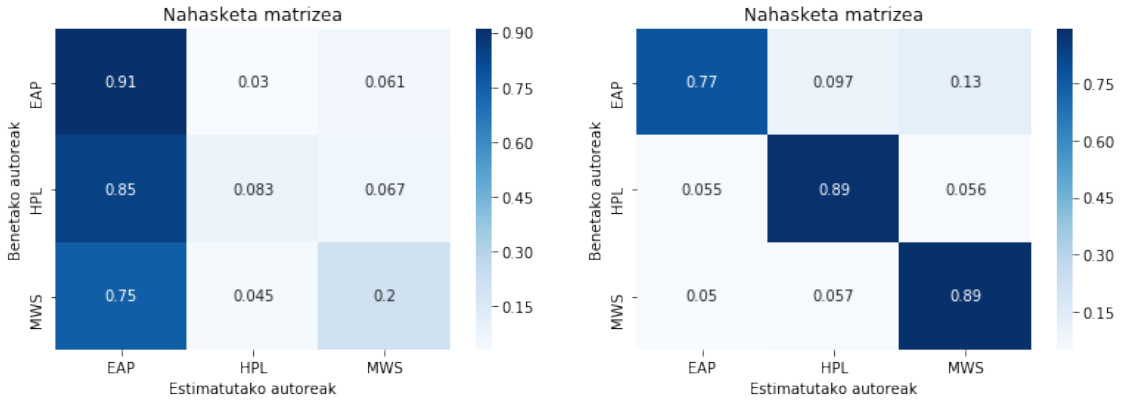
Honen ostean, ebaluaketa prozesu berbera egin da, kasu honetan UBM metodoa erabiliz.

[7]:
```
df=results(
[prob5,prob6,prob7,prob8],
test,
[True,True,False,False],
[True,False,True,False])
df
```

	Keep Punctuation Marks	Keep Uppercase	Accuracy	Log Loss
0	True	True	0.843	0.916
1	True	False	0.839	0.921
2	False	True	0.843	0.906
3	False	False	0.836	0.912

Azkenik, bai biderkadura eskalar zuzena eta UBM kasuan lortzen diren nahasketa matrizeak irudikatzen dira.

[8]:
```
predicted = prob1.drop(['id'],axis=1).idxmax(axis=1)
aim = test.author
plot_confusion_matrix(aim,predicted)
predicted2 = prob5.drop(['id'],axis=1).idxmax(axis=1)
aim = test.author
plot_confusion_matrix(aim,predicted2)
```



B.4 Erregresio logistikoa

Beharreezko klaseen eta funtzioen inportazioa:

[1]:
```
from Prozesaketa import Prozesaketa
from Vector      import Vector
from Generator   import Generator
from Analisia    import results,testrainsplit,accuracy,logloss
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score,log_loss
from ConfusionMatrix import plot_confusion_matrix
import matplotlib.pyplot as plt

import pandas as pd
import numpy as np
import time
```

Lehenengo, Bayesen probabilitateetan oinarritu gara erregresio logistikoa egiteko. Kasu honetan hiru datu multzo behar dira: lehenengoa Bayes-en modeloa entrenatzeko; bigarrena erregresio logistikoa entrenatzeko; eta hirugarrena, erregresio logistikoaren ebaluaketa egiteko. Hortaz, hiru datu multzotan banatzen dira hasierako datuak.

[2]:
```
data = pd.read_csv('train.csv')
train , test , evaluation = testrainsplit(
data,test_size=0.2,evaluation_size=0.05)
ptrain = Prozesaketa()
ptest = Prozesaketa()
pevaluation= Prozesaketa()
```

Ondoren aurre prozesaketa aplikatzen zaie datu multzo ezberdinei.

[3]:
```
train_data1=ptrain.process(train,
KeepPunctuationCharacters=True,KeepUppercase=True)
test_data1=ptest.process(test,
KeepPunctuationCharacters=True,KeepUppercase=True)
eval_data1=pevaluation.process(evaluation,
```

```
KeepPunctuationCharacters=True,KeepUppercase=True)
g1 = Generator(train_data1)
```

Honen ostean, *generator*-a entrenatzen da, honi ondoren erregresio logistikoa aplikatzeko. Aldi berean, datuak egokitzen dira *sklearn*-eko *LogisticRegression* klaseari eman ahal izateko.

[4]:
```
fit_matrix1, y1 = g1.predictions(test_data1), test_data1.author
fit_matrix1 = fit_matrix1.drop(["id"],axis=1)
```

Honela, erregresio logistikoa egiten da datu horientzat. Horrez gain, erregresio logistikoaren ebaluazioa egiteko datuak prestatzen dira.

[5]:
```
model1 = LogisticRegression(
solver="lbfgs",multi_class="auto",random_state=0,C=1).fit(fit_matrix1,y1)
eval_matrix1, aim1 = g1.predictions(eval_data1), eval_data1.author
eval_matrix1 = eval_matrix1.drop(["id"],axis=1)
```

Hau izanda, lehenengo entrenamendu datuekin izandako eraginkortasuna aztertzen da, gehiegizko entrenamendurik egon ote den aztertzeko.

[6]:
```
output=model1.predict_proba(fit_matrix1)
df = pd.DataFrame(output,columns=['EAP','HPL','MWS'])
df['id']=test_data1.id
results([df],test,[True],[True])
```

	Keep Punctuation Marks	Keep Uppercase	Accuracy	Log Loss
0	True	True	0.851	0.430

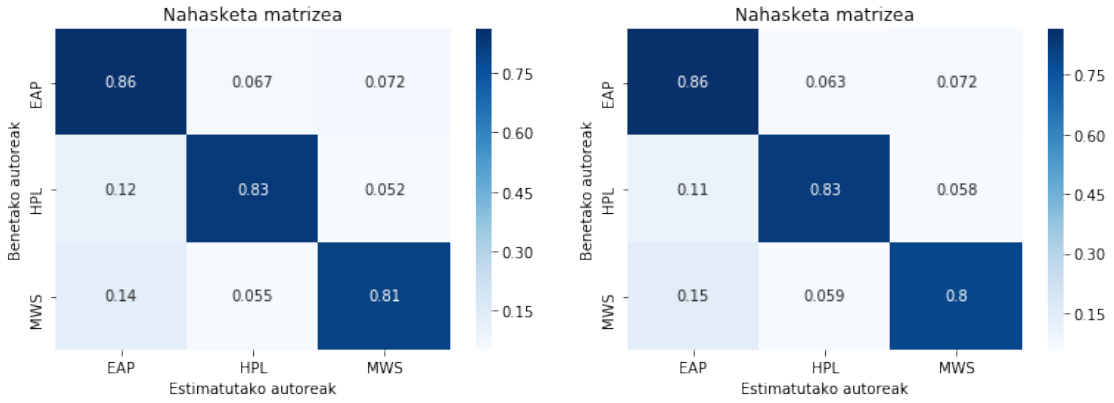
Ondoren, ebaluazio datuetan aztertzen da sarearen egiazko eraginkortasuna.

[7]:
```
y2=eval_data1.author
output=model1.predict_proba(eval_matrix1)
df = pd.DataFrame(output,columns=['EAP','HPL','MWS'])
df['id']=eval_data1.id
results([df],evaluation,[True],[True])
```

	Keep Punctuation Marks	Keep Uppercase	Accuracy	Log Loss
0	True	True	0.847	0.428

Azkenik, entrenamendu zein ebaluazio datuetako nahasketa matrizeak irudikatu dira.

[8]:
```
plot_confusion_matrix(model1.predict(fit_matrix1),y1)
plot_confusion_matrix(model1.predict(eval_matrix1),y2)
```



Erregresio logistikoaren bigarren atal honetan, teknika berbera aplikatu da, kasu honetan UBM ereduarekin definitzen diren bektoreengan. Lehenengo datuak bi multzotan banatzen dira (jada ez da beharrezkoa hiru multzo ezberdinetan banatzea).

[9]:
```
train , test = testrainsplit(data,0.2)
ptrain = Prozesaketa()
ptest = Prozesaketa()
```

Aurre prozesaketa burutzen da.

[10]:
```
train_data=ptrain.process(train,
KeepPunctuationCharacters=True,KeepUppercase=True)
test_data=ptest.process(test,
KeepPunctuationCharacters=True,KeepUppercase=True)
```

Ondoren, UBM ereduan definitu diren bektoreak kalkulatzen dira. Matrize moduan jasotzen dira hauek, azaldu den moduan.

[11]:
```
v1 = Vector(train_data,UseUniversalBackgroundModel=True)
```

Erregresio logistikorako datuak tratatzen dira.

[12]:
```
fit_matrix=v1.phrases_to_matrix(train_data)
y = train_data.author
```

Erregresio logistikoa entrenatzen da, entenamenduko datu multzoetara parametroak egokituz.

[13]:
```
model1 = LogisticRegression(
solver="lbfgs",multi_class="auto",random_state=0,C=2e-5).fit(fit_matrix,y)
```

Entrenamendu datuetara egokitzeko izan duen gaitasuna kalkulatzen da, entrenamendu datuekin ebaluatuz.

[14]:
```
output2=model1.predict_proba(fit_matrix)
df = pd.DataFrame(output2,columns=['EAP','HPL','MWS'])
df['id']=train_data.id
results([df],train_data,[True],[True])
```

	Keep Punctuation Marks	Keep Uppercase	Accuracy	Log Loss
0	True	True	0.987	0.151

Ondoren, ebaluazio datuak sartzen dira sistemaren egiazko ebaluaketa egiteko.

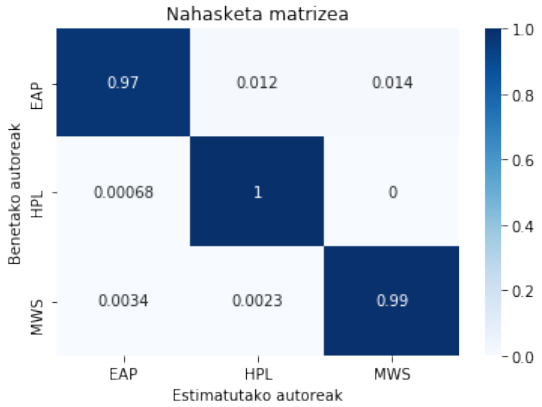
[15]:
```
test_matrix, aim = v1.phrases_to_matrix(test_data), test_data.author
output=model1.predict_proba(test_matrix)
df = pd.DataFrame(output,columns=['EAP','HPL','MWS'])
df['id']=test_data.id
results([df],test,[True],[True])
```

	Keep Punctuation Marks	Keep Uppercase	Accuracy	Log Loss
0	True	True	0.839	0.450

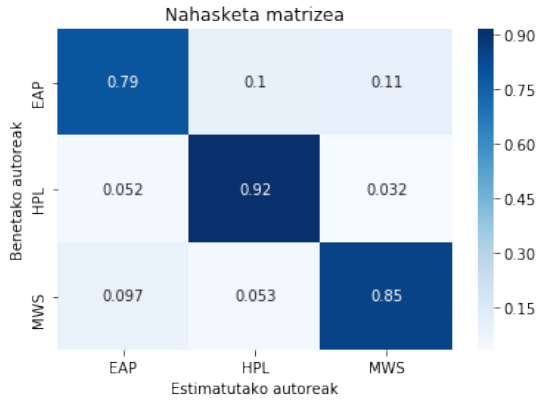
Azkenik, entrenamendu zein ebaluazio datuen nahasketa matrizeak kalkulatzen dira.

[16]:
```
plot_confusion_matrix(model1.predict(fit_matrix),y)
```

[17]:
```
plot_confusion_matrix(model1.predict(test_matrix),aim)
```



(a) Bektoreekiko erregresio logistikoa erabiliz nahasketa matrizea entrenamendu datuetan.



(b) Bektoreekiko erregresio logistikoa erabiliz nahasketa matrizea ebaluazio datuetan.

Irudia B.4.1: Entrenamendu eta ebaluazio datuetarako nahasketa matrizeak erregresio logistikoa bektoreei aplikatuz.

B.5 Sare Neuronalak

Lehenik, kasu guztietan bezala, beharrezko metodo guztiak inportatzen dira.

[1]:
```
import numpy as np
import pandas as pd

from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
from keras.layers import Dropout
from keras.layers.embeddings import Embedding
from keras.preprocessing import sequence
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
from keras.optimizers import Adam

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelBinarizer
from ConfusionMatrix import plot_confusion_matrix
```

```
Using TensorFlow backend.
```

Entrenamendu zein ebaluazio prozesuetarako datu sortak eskuratzen dira.

[2]:
```
train_set = pd.read_csv('train.csv', index_col=False)
```

Ondoren, esaldi guztiak bata bestearen segidan jartzen dira *Tokenizer*-ari pasatzeko.

[3]:
```
all_sentences =pd.concat([train_set['text']])
```

[4]:
```
labelbinarizer = LabelBinarizer()
labelbinarizer.fit(train_set['author'])
y = labelbinarizer.fit_transform(train_set['author'])
```

Emaitza zuzeneko klaseak *one-hot-encoding*-era bilakatzen dira; hau da, Z multzoan ezartzen dira (balio guztiak nuluak esaldiari dagokion autorearen posizioa izan ezik). Aldi berean bi parametro definitzen dira: *max_words* parametroak sare neuronalean erabiliko duen hitz kopurua zehazten du. Kasu honetan, gehien agertzen diren lehen 15000 hitzak. Bestetik *max_len* parametroak esaldi batek izan dezakeen luzeera maximoa zehazten da. Hauek jakinik tokenizerrak esaldietako hitz bakoitzari zenbaki oso bat atxikiko dio. Bestalde, entrenamendu eta ebaluazio datuak banatzen dira.

[5]:
```
max_words = 15000
max_len = 200


tokenizer = Tokenizer(num_words=max_words)
tokenizer.fit_on_texts(all_sentences)
X = tokenizer.texts_to_sequences(train_set['text'])
X=pad_sequences(X,maxlen=max_len)
```

```
X_train, X_eval, y_train, y_eval =␣
 ↪train_test_split(X,y,random_state=0,test_size=0.15)
dict_size = len(tokenizer.word_index)
```

Ondoren, sare neuronalaren egitura orokorra deskribatzen da, eta entrenamendu prozesua exekutatu.

[6]:
```
model = Sequential()
model.add(Embedding(max_words,50,input_length=max_len))
model.add(Dropout(0.5))
model.add(LSTM(64))
model.add(Dropout(0.5))
model.add(Dense(16,activation='relu'))
model.add(Dense(3, activation='softmax'))
model.compile(optimizer=Adam(lr=0.
 ↪0005),loss='categorical_crossentropy',metrics=['acc'])
model.fit(X_train, y_train, epochs=4,␣
 ↪batch_size=64,validation_data=(X_eval,y_eval))
```

```
Train on 16642 samples, validate on 2937 samples
Epoch 1/4
16642/16642 [==============================] - 52s 3ms/step - loss: 1.0312 -
acc: 0.4495 - val_loss: 0.8700 - val_acc: 0.5655
Epoch 2/4
16642/16642 [==============================] - 51s 3ms/step - loss: 0.7143 -
acc: 0.6833 - val_loss: 0.6019 - val_acc: 0.7566
Epoch 3/4
16642/16642 [==============================] - 51s 3ms/step - loss: 0.4309 -
acc: 0.8417 - val_loss: 0.5278 - val_acc: 0.8114
Epoch 4/4
16642/16642 [==============================] - 50s 3ms/step - loss: 0.2983 -
acc: 0.8962 - val_loss: 0.4586 - val_acc: 0.8301
```

Azkeneko emaitzetan sare neuronalak pixkanaka nola ikasten duen adierazten da, entrenamendu prozesua burutu ahala. Ondoren, entrenamendu zein ebaluazio datuak erabiliz sarearen eraginkortasuna kalkuatzen da.

[7]:
```
eval_train=model.evaluate(X_train,y_train,batch_size=256)
print("Log_loss entrenamendu datuetan : "+str(eval_train[0]))
print("Accuracy entrenamendu datuetan : "+str(eval_train[1]))
evaluation=model.evaluate(X_eval,y_eval,batch_size=256)
print("Log_loss ebaluazio datuetan    : "+str(evaluation[0]))
print("Accuracy ebaluazio datuetan    : "+str(evaluation[1]))
```

```
16642/16642 [==============================] - 6s 354us/step
Log_loss entrenamendu datuetan : 0.181
Accuracy entrenamendu datuetan : 0.944
2937/2937 [==============================] - 1s 358us/step
Log_loss ebaluazio datuetan    : 0.459
Accuracy ebaluazio datuetan    : 0.830
```

Azkenik, entrenamendu eta ebaluazio datu hauen nahasketa matrizeak irudikatzen dira.

[8]:
```
from ConfusionMatrix import plot_confusion_matrix
train_pred=list(np.argmax(row) for row in model.predict(X_train))
train_pred=list('EAP' if (elem==0) else ('HPL' if (elem==1) else 'MWS') for␣
 ↪elem in train_pred)
train_true=list(np.argmax(row) for row in y_train)
train_true=list('EAP' if (elem==0) else ('HPL' if (elem==1) else 'MWS') for␣
 ↪elem in train_true)
plot_confusion_matrix(train_true,train_pred)
eval_pred=list(np.argmax(row) for row in model.predict(X_eval))
eval_pred=list('EAP' if (elem==0) else ('HPL' if (elem==1) else 'MWS') for␣
 ↪elem in eval_pred)
eval_true=list(np.argmax(row) for row in y_eval)
eval_true=list('EAP' if (elem==0) else ('HPL' if (elem==1) else 'MWS') for␣
 ↪elem in eval_true)
plot_confusion_matrix(eval_true,eval_pred)
```