Facultade de Informática
UNIVERSIDADE DA CORUÑA

TRABALLO FIN DE GRAO
GRAO EN ENXEÑARÍA INFORMÁTICA
MENCIÓN EN COMPUTACIÓN

Detección temperá da depresión mediante *temporal word embeddings*

Estudante: Manuel Couto Pintos
Dirección: Javier Parapar López, Alfonso Landín Piñeiro

A Coruña, June de 2021.

Adicado aos que nunca se renden

Agradecementos

Moitas grazas ós membros do iRlab, ó profesor Álvaro Barreiro por terme dado a oportunidade de facer este traballo de fin de grao. Igualmente agradecido ós meus titores Javier Parapar e Alfonso Landín, sen o seu apoio e dirección non tería sido posible realizar este proxecto con éxito. Por suposto tamén quero agradecer o apoio constante e a paciencia dos meus pais que sempre estiveron ahí. Grazas a todos os que me deixo no tinteiro, pero que saben que son igualmente importantes para min, moitas grazas pola vosa axuda e apoio.

Resumo

A depresión é un trastorno mental habitual que afecta a máis de 300 millons de persoas no mundo. Constitúe a principal causa de discapacidade según a OMS e no peor dos casos pode levar ao suicidio. Ainda que existen tratamentos eficaces o seu difícil diagnóstico xunto coa falla de recursos fan que o tratamento non chegue a máis da metade dos afectados [1]. Neste contexto a iniciativa eRisk establece unha serie de retos para tratar de detectar de forma temperá o perigo de sufrir transtornos persoais. Entre estes retos atópanse a detección temprá da depresión. Esta tarefa desempeñouse no 2017 [2] e 2018 [3] contando con metodoloxías propias de evaluación.

Neste proxecto imos crear un sistema de alertas que axude a detectala depresión de forma temperá. Estudaremos as publicacións de diferentes usuarios na rede social Reddit para extraer os patróns propios da linguaxe das persoas que desenrolaran depresión. Teremos en conta cómo evoluíu o significado das palabras no tempo para poder determinar cándo o nivel da depresión traspasa un umbral. Para esta tarefa empregaremos representacións temporáis densas de palabras (*temporal word embedings*). Estas representacións demostraron súa utilidade para captala evolución dos sentimentos asociados as palabras ó longo do tempo [4]. En concreto usaremos o algoritmo TWEC proposto no paper *Training Temporal Word Embeddings With a Compass* [5] e compararemos os resultados deste algoritmo cos resultados dos presentados nas edicións do CLEF eRisk 2017 [2] e CLEF eRisk 2018 [3].

Palabras chave:

- eRisk
- Depresión
- NLP
- Temporal word embeddings
- Word2Vec
- Redes neuronais recurrentes
- Maquinas de soporte vectorial
- CBOW
- TWEC
- Scrum
- Compass
- Git

Índice Xeral

Índice de Figuras

Índice de Táboas

Capítulo 1

Introdución

Neste proxecto búscase aplicar técnicas de procesamento da linguaxe á tarefa de detección temperá da depresión. Escóllese esta tarefa xa que as características na linguaxe que presenta unha persoa con depresión están documentadas en diversos estudos, coma queda recollido no traballo *linguistic features in depression:a meta-analysis* [4]. Ou o estudo da frecuencia do uso dos pronomes en primeira persoa do singular, que sinala un maior uso deste na xente con depresión [10]. Tamén están os estudos que asocian a depresión co uso de palabras con componentes emocionáis, os cales apuntan a que a detonación dun esquema de depresión pode chegar a apreciarse na linguaxe dun deprimido, xa que ó expresar a súa visión de si mesmos, do mundo e do futuro adoptan unha visión pesimista. Por conseguinte usará máis palabras negativas que positivas [4]. Ó longo deste traballo empregamos diferentes tecnoloxías para estudar a variación na linguaxe de usuarios da rede social Reddit. Dispomos de datos etiquetados para entrenar e testar os modelos xunto con resultados doutros algoritmos que foron presentamdos na competición eRisk nas edicións do 2017 [2] e 2018 [3], e que empregaremos como referencia para comparar cos resultados obtidos.

As tecnoloxías escollidas para esta tarefa xiran entorno ao modelo Word2Vec, este foi publicado por Thomas Mikolov e os membros do seu equipo en Google no ano 2013. Esta técnica permite captar o significado das palabras e o seu contexto mediante o uso de redes neuronais. Estes modelos permiten codificar vectores dispersos cunha gran dimensionalidade a vectores densos, sobre os que se poden realizar operacións alxebraicas que nos serán moi útiles. Referirémonos a esta representación densa como *embedding*. Este modelo presentouse coma alternativa con algunhas ventaxas con respecto ás máis usadas do momento como eran as LSA(Latent Semantic Analysis) ou as LDA(Latent Dirichlet Allocation) [11]. Este modelo na súa forma estandar presenta algúns problemas para representar a evolución da linguaxe xa que pola nureza estocástica das redes neuronais e a varición entre os contextos dun documento a outro, os *embeddings* de cada texto están representados con diferente aliñamento dentro do espazo vectorial [5]. Se procesamos dous documentos nos que unha palabra en concreto

ten o mesmo significado nada nos garante que o seu *embedding* ocupe o mesmo lugar no espacio vectorial. Existen diversas tecnoloxías que afrontan este problema, pero estas presentan algunhas desvantaxes, como a de precisar grándes cantidades de datos para funcionar ben ou ser computacionalmente moi custosas. por outra banda temos o modelo TWEC (*Temporal Word Embeddings with a Compass*) [5], o cal plantexa unha solución a estas problemáticas. Esta proposta plantexa usar un modelo atemporal como brúxula para inicializar outros modelos temporais e así aliñar os seus *embeddings*. Desta forma podese captar os cambios no lugar que ocupan os *embeddings* das palabras no espacio vectorial ó longo do tempo.
Ó longo deste proxecto exploraremos as posiblidades destas tecnoloxías para a tarefa obxetivo. O que faremos será empregar o historial de publicacións na rede social Reddit de usuario deprimidos e non deprimidos para entrenar modelos temporáis. Estes modelos estarán aliñados cun mesmo compass dentro dun mesmo espacio vectorial. A continuación calcularemos canto se moveu cada palabra con respecto ao compass en cada momento e para cada usuario de training. Esta variación medida para cada palabra computarase mediante a distancia de coseno. Esta medida para todas as palabras nun momento do tempo comformará o que chamaremos vector delta para o momento t. Finalmente empregaremos diferentes clasificadores para alimentalos cos vectores delta dos usuarios de training a modo de exemplos de entrenamento. Como paso final testaremos no conxunto de testing e avaliaremos o funcionamento do *pipeline*.

1.1 Motivación

Como xa se veu contando a depresión é unha das principales causas de discapacidade no mundo e contribue de forma importante á carga mundial xeral de morbilidad. No peor dos casos a depresión pode levar ó suicidio, acontecendo cada ano perto de 800.000 suicidios no mundo, dos cales 3.500 son no noso país. O impacto desta efermidade é evidente endemáis constitúe a segunda causa de morte entre os grupos etarios de entre 15 e 29 anos. [1]

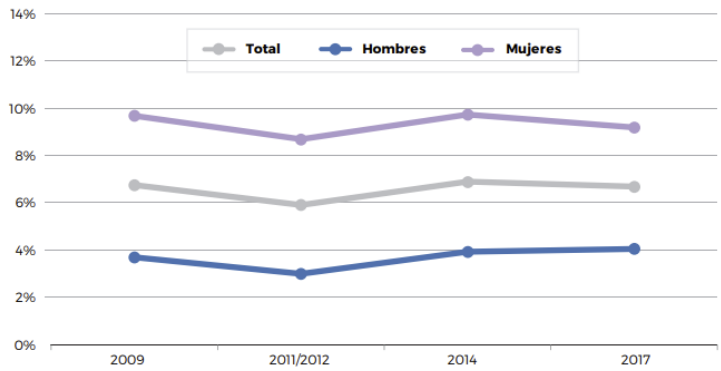


Figura 1.1: Prevalencia de depresión diagnosticada. [6]

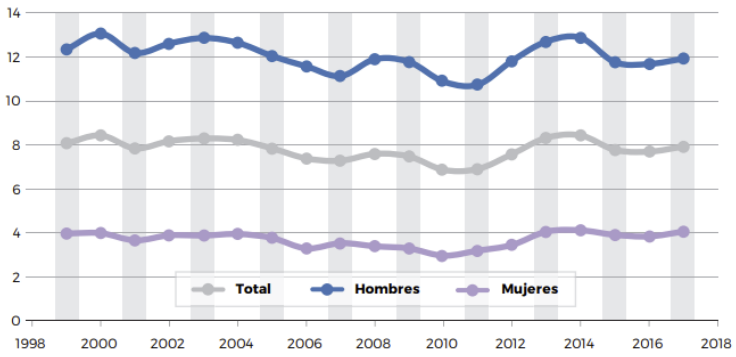
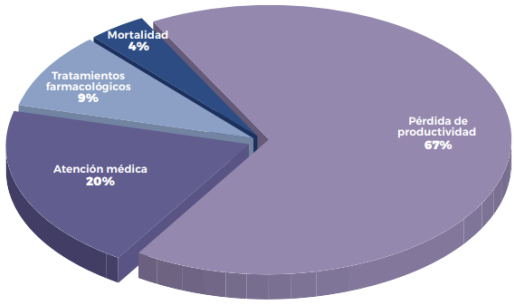


Figura 1.2: Tasa de mortalidade por suicidio 100m/h. [6]

A depresión non é un problema estrictamente sanitario, debe enmarcarse nun contexto máis amplio xa que afecta a nivel social, familiar, laboral e económico. Según un estudo realizado en 28 países europeos, esta patoloxía supuso en 2004 en todo Europa un coste de 118.000 millons de euros, representa o 1% do PIB. En concreto en España, os custos da depresión cifraronse en 5.005 millons de euros dos cales 985 corresponderon a custos médicos, 449 millons ao consumo de fármacos 3.385 aos custos pola perda de productividade e 187 millons aos custos da mortandade atribuida a depresión. Véxanse figuras 1.3a e 1.3b . [6]

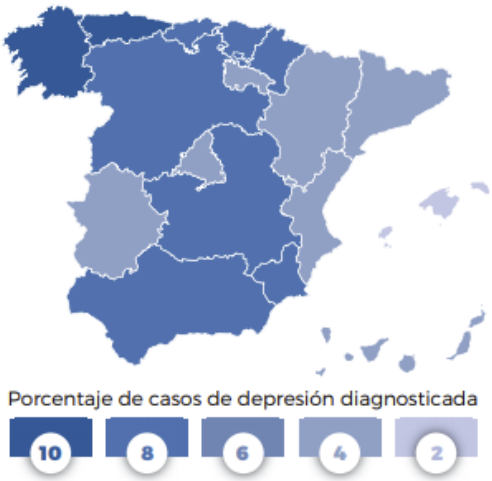


(a) Custos asociados. [6]

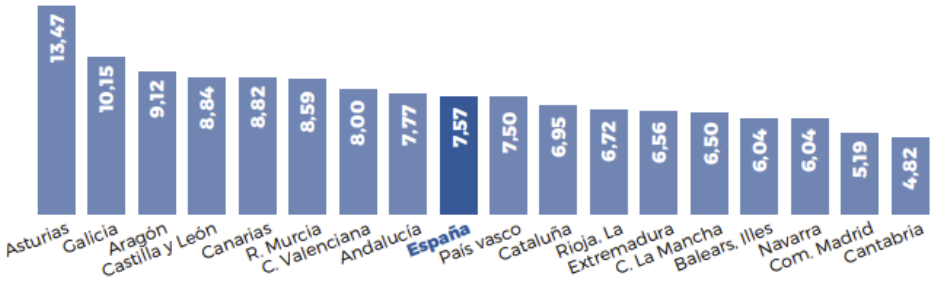
Ambos sexos	Cuadro depresivo mayor	%	Otros cuadros depresivos	%	Total	%
Trabajando	230,8	17%	354,6	24%	585,4	21%
En desempleo	300,4	22%	292,9	20%	593,3	21%
Jubilado/a o prejubilado/a	460,7	34%	466,4	31%	927,1	33%
Estudiando	10,8	1%	43,6	3%	54,4	2%
Incapacitado/a para trabajar	153,3	11%	82,0	6%	235,3	8%
Labores del hogar	205,2	15%	238,6	16%	443,8	16%
Otros	1,9	0%	6,8	0%	8,7	0%
Total	1.363,1		1.484,9		2.848,0	

(b) Prevalencia por actividade. [6]

En relación co resto de comunidades autónomas Galicia é unha das que ten peores datos, tanto en número de cadros con depresión coma número de suicidios. Véxanse figuras 1.4a e 1.4b. [6] Existen tratamentos eficaces para convatir a depresión máis non é doado chegar a todos os afectados, tanto pola falla de recursos coma pola estigmatización das efermedades mentáis. No contexto actual de crise sanitaria e coa situación de illamento que se está a sufrir, en especial as persoas maiores, prevese un incremento nos casos de depresión a nivel global. Endemáis o saturamento dos servizos sanitarios pola pandemia, afondan o problema da falla de recursos nos sistemas de diagnostico temperán e prevención. En base ó anteriormente exposto, a principal motivación é dar apoio dende as tecnoloxías da información aos traballos de detección temperá e prevención da depresión no ámbito das redes sociais e os riscos en internet.



(a) Depresión por CA. [6]



(b) Tasa suicidio por 100m/h CA. [6]

1.2 Obxetivos

En base as características das ferramentas a empregar e do problema a abordar, os obxetivos deste proxecto desgránanse en:

- Probar a utilidade do algoritmo TWEC para a detección temperá da depresión
- Optimizar os procesos que envolven o uso da ferramenta
- Cuantificar a eficacia con respecto a otros algoritmos
- Explorar a combinación do algoritmo TWEC con outros algoritmos intelixentes

1.3 Estrutura da memoria

Neste apartado exporase a estructura da memoria xunto cun breve resumo do contido de cada capítulo.

- **Introducción**: este é o capítulo introdutorio, onde se presenta o problema, o contexto e razóns que motivan a realización deste proxecto. Neste capítulo tamén se atopa a estrutura da memoria e o plan de traballo.
- **Conceptos**: este capítulo introduce unha serie de conceptos base para comprender o que se está tratando de facer en cada momento, e algunhas decisións tomadas.
- **Tecnoloxías**: neste capitulo explicase cales foron as ferramentas, entornos de desenvolvemento e outras tecnoloxías empregadas durante a realización deste proxecto.
- **Proposta e Evolución**: este é o capítulo no que se explica o plantexamento inicial do proxecto, nel tamen se explica unha serie de modificacións que se realizaron a medida que o traballo foi avanzando.
- **Metodoloxía e Desenvolvemento**: neste capítulo explícase que metodoloxía se empregou para o desenvolvemento do proxecto e como se adaptou para amoldarse as características do proxecto. Tamén neste capitulo comentarase a planificación e os traballos feitos en cada etapa do proxecto.
- **Deseño**: nesta sección mostrase o deseño da ferramenta, explicado mediante diagramas de clases.
- **Resultados**: neste capítulo recollense os resultados dos experimentos realizados o longo do proxecto e comparanse entre eles o cos presentadados no CLEF eRisk 2017 e 2018.

- **Conclusións**: este capitulo faise unha reflexión en torno as dificultades do proxecto e aos resultados obtidos.

1.4 Plan de traballo

A realización deste traballo estivo dividido en varias etapas. Unha etapa de documentación, unha etapa de investigación e desenrolo, e a etapa final de avaliación de resultados e redacción da memoria.

- Na etapa de documentación leronse artigos relacionados co ámbito do Procesamento da Linguaxe Natural, da Recuperación da Información e dos Algoritmos Intelixentes. Esta etapa non estivo suxeta a planificación.
- A etapa de investigación e desenvolvemento, na que se partiu dun plantexamento inicial. Este plantexamento consiste nun *pipeline* completo capaz de realizar unha clasificación dos usuarios e facer unha avaliación desta clasificación. O longo de cada semana faranse propostas de ideas para a mellora dos resultados do *pipeline*. Anotaranse as modificacións que se levaron a cabo.
- Etapa de avaliación de resultados e redacción da memoria. Esta etapa adicouse á recopilación de resultados e á redacción da memoria e ás conclusións.

Capítulo 2

Conceptos

NESTE capítulo explicaranse algúns conceptos que van ser recurrentes ó longo do proxecto ou que son necesarios para comprender algunhas das suas partes.

2.1 Vector Space Model

Os modelos de espazos vectoriales son pilares fundamentáis do álxebra lineal. Neles os vectores representanse mediante secuencias ordeadas de números ou coordeadas nun espazo vectorial. Estas coordeadas representan unha posición ou podense usar para identificar unha dirección e magnitude nese espazo. Entón unha secuencia de n números reáis sería un vector nun espazo n dimensional $\mathbf{R^n}$.

Os modelos de espazos vectoriais serviron de base para moitas das investigacións dos anos 1960s e 1970s no campo da Recuperación da Información. Históricamente a súa incorporación foi moi importante e ó longo dos anos fóronse desenvolvendo técnicas efectivas. Típicamente asúmese que os documentos son vectores dun espazo vectorial n-dimensional, onde n é o número de palabras diferentes do seu vocabulario. Un documento D_i está representado por un vector de n terminos 2.1.

$$D_i = (d_{i,1}, d_{i,2}, \times\times\times, d_{i,n})$$

Figura 2.1: Representación vectorial do documento D_i

Se dispomos dun corpus ou colección de documentos, e queremos facer unha representación do mesmo, tendo n palabras no vocabulario e m documentos difrentes poderíamos representar esta información mediante a seguinte matriz 2.2.

$$M = \begin{pmatrix} d_{1,1} & d_{1,2} & \times\times\times & d_{1,n} \\ d_{2,1} & d_{2,2} & \times\times\times & d_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ d_{m,1} & d_{m,2} & \times\times\times & d_{m,n} \end{pmatrix}$$

Figura 2.2: Matriz dunha colección de documentos [7]

Onde cada elemento $d_{i,j}$ toma o seu valor da iformación estadística dunha palabra. Típicamente as dúas representacións máis habituais son o TF que representa a frecuencia dunha palabra nun documento é o TF-IDF 2.3 que representa canto aparece unha palabra nun documento en relación á aparición desa palabra no resto dos documentos, xa que o DF representa o número de documentos nos que aparece unha palabra.

$$\text{TF-IDF}_{m,n} = \text{TF}_{m,n} * \log \frac{N}{\text{DF}_n}$$

Figura 2.3: fórmula do TF-IDF [7]

Estas representacións adoecen de ser moi dispersas (moitas posicións son ceros). Ó estar traballando con vocabularios de miles de palabras, os cales en ocasións conforman conxuntos disxuntos entre os documentos, fai que ó empregar matrices densas, a maioría das posicións sexan ceros. Para estes casos é máis apropiado empregar matrices dispersas, as cales son moito máis eficientes en termos de complexidade espacial, ainda que un pouco máis lentas á hora de acceder á información que conteñen. O beneficio nótase especialmente cando, coma no caso deste traballo, temos documentos con poucas palabras e con vocabularios moi diferentes entre sí. Ó ter moitísimos máis valores cero que valores que non son cero, compénsanos gardar so os valores non son cero, xa que son os que aportan información. Nestas matrices dispersas este tipo de información almacénase de xeito eficiente. Existen varios formatos destas matrices, un exemplo son as que mapean pares (row, column) con valores x de forma que sendo M a matiz densa temos que $M[row, column] = x \implies x \neq 0$.

Co que temos ata o momento xa podemos empezar calcular o parecidos que son 2 documentos. Isto pódese facer xa que ó estar traballando nun espazo euclídeo podemos calcular a distancia de coseno entre dous vectores. [7]

$$\cos(\varphi) = \frac{D_x \times D_y}{D_x \quad D_y}$$

Figura 2.4: Fórmula distancia de cosemo [8]

2.2 Redes Neuronáis

A primeira aproximación ao funcionamento das redes de neuronás artificiáis remóntase ós 1950s nos que Fran Rosenblatt propón o percetrón [12]. Este foi un algoritmo novedoso para atopar patróns nos datos, é unha abstracción matemática do funcionamento dunha neurona. A idea básica é a de ter un conxunto de entradas representadas nun vector $\overrightarrow{X} = [x_1, x_2, \ldots, x_i, \ldots, x_n]$ e que se corresponden a unha serie de características xunto cun parámetro de sesgo. Todos elas asociadas a un valor de peso que inicialmente teñen valores aleatorios e que se representan como o vector $\overrightarrow{W} = [w_1, w_2, \ldots, w_i, \ldots, w_n]$. Logo o percetrón combina estes valores multiplicando a entrada polo seu valor correspondente e suma os resultados $(x_1 \subseteq w_1) + (x_2 \subseteq w_2) + \times\times\times + (x_i \subseteq w_i) + \ldots$ logo aplicaselle unha función threshold, ou función de activación. Todo isto pódese representar matemáticamente tal que $if\ (\sum_{i=1}^{n} x_i \subseteq w_i > threshold)\ then\ f(\overrightarrow{x}) = 1\ else\ f(\overrightarrow{x}) = 0$ xunto cunha función de error $err(x) = \|y \quad f(x)\|$ O obxectivo do algoritmo e minimizar o error en cada un dos pesos $J(x) = min \sum_{i=1}^{n} err(x_i)$. [8]

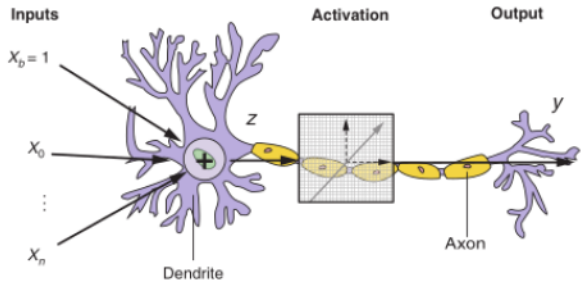


Figura 2.5: Percetrón e neurona real superpostas. [8]

O Precetrón por si mesmo non funcionaba ben para problemas linealmente non separables. Para poder solucionar problemas deste tipo foi necesario aliñar varias capas de neuronas, de tal forma que as saídas dunha capa conforman as entradas da seguinte e onde antes tiñamos unha función simple de erro agora temos a función de back-propagation. Como o seu nome indica, esta función propaga cara atrás o erro cometido ás entradas de cada neurona da rede, e poder así facer o reaxuste dos seus pesos. Para definir o algoritmo back-propagation pártese da suma total dos erros cuadráticos de cada un dos pesos.

$$Error = \sum_{i=1}^{n} \frac{1}{2}\ err(x_i)^{\ 2}$$

Figura 2.6: Error medio cuadrado

Logo usando a regla da cadea, podemos calcular a derivada da función de activación con respecto a cada neurona da rede, e podemos obter canto influiu ese peso no erro cometido e axustalo. Se estamos na última capa o cálculo do erro é directo, A derivada do erro con respecto o $j \quad th$ saida que o conforma é:

$$\Delta w_{i,j} = \quad \alpha \frac{\partial Error}{w_{i,j}} = \quad \alpha y_i (y_j \quad f(x_j)) y_j (1 \quad y_j)$$

Figura 2.7: Derivada do error na última capa

Para actualizar os pesos dunha capa oculta a ecuación é un pouco máis complexa:

$$\Delta w_{i,j} = \quad \alpha \frac{\partial Error}{w_{i,j}} = \quad \alpha y_i (\sum_{l \varepsilon L} \delta_j w_{j,l}) y_j (1 \quad y_j)$$

Figura 2.8: Derivada do error nas capas ocultas

O parámetro α das ecuacións definen a ratio de aprendizaxece. Este parámetro se colle valores altos fai que a corrección sexa grande, se o valor é demasiado grande pode levar unha corrección ineficiente e o seguinte error será ainda maior. Se pola contra este parámetro colle valores moi pequenos o algoritmo podería quedar estancado nun mínimo local.

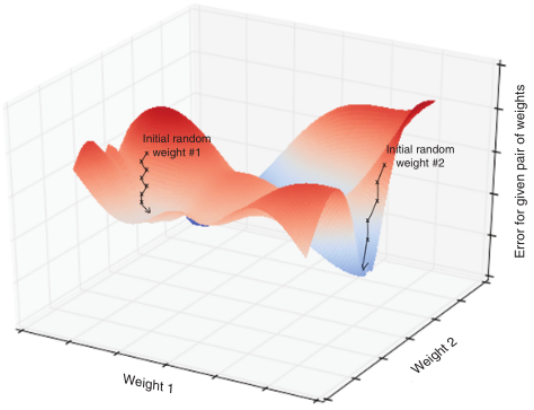


Figura 2.9: Superficie do erro

Temos varias formas de entrenar unha rede neuronal, a primeira *batch learning* na que o erro e os novos pesos calculase ó final do *batch*. É un método rápido pero pode quedar atascado máis facilmente nun minimo local. Outra opción é mediante o *stochastic gradient descent*, neste caso os pesos da rede neuronal actualízanse despóis de cada patrón de entrenamento, o que o fái unha opción máis lenta pero resiliente. *Mini-batch learning* pola banda colle o mellor dos dous métodos anteriores, a rapidez do *batch learning* e a resiliencia do *stochastic learning*.

2.3 Word2Vec

Word2Vec é un método de construcción de representacións densas de palabras, foi presentado por Thomas Mikolov xunto cos seus compañeiros de google no ano 2013 [11]. Estes compararon varios modelos, tratando de maximizar o acerto e minimizar a complexidade computacional. O que descubriron no seu estudo é que se poden lograr representacións densas de palabras dunha gran calidade sen necesidad de usar modelos moi complexos. Este modelo emprega redes neuronáis nun esquema *encoder decoder* para lograr a representación densa de palabras.

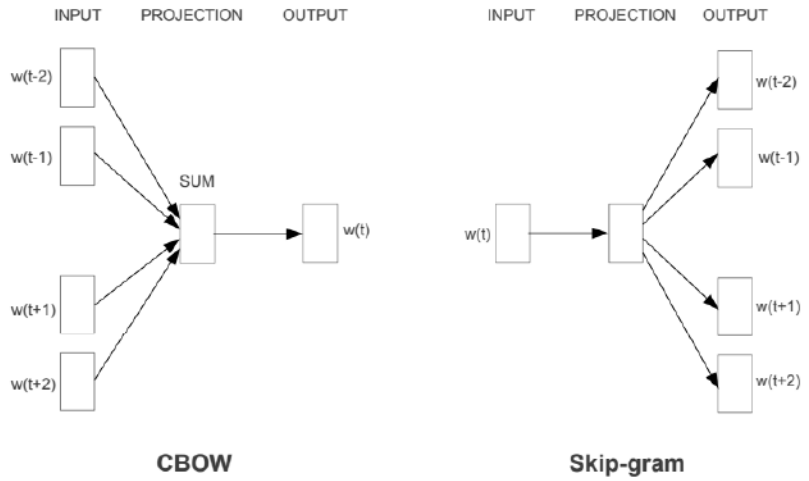


Figura 2.10: Esquena encode-decoder

No seu traballo, definen a complexidade computacional como na figura 2.11 sendo E o número de *epochs* de entrenamento, T o número de palabras no conxunto de entrenamento e Q coma complexidade específica de cada aproximación.

$$O = E * T * Q$$

Figura 2.11: Complexidade computacional

CBOW é similar a unha *Neural Net Language Model* alimentado hacia adiante. En base ós seus experimentos decidiron eliminar a capa oculta non lineal, e fixeron que a capa de proxeción se compartise co resto das capas. Todas as palabras proxéctanse sobre a mesma posición e os seus vectores son promediados. Incluiron un clasificador *log-lineal* que recibe como input unha ventá de tamaño n collendo $n/2$ palabras que representan ás predecesoras e $n/2$ palabras que representan ás sucesoras dunha palabra central que o modelo pode predecir. O valor Q para o cálculo da complexidade deste algoritmo defínese 2.12 sendo N o

número de palabras previamente codificadas, V e o tamaño do vocabulario e D e o tamaño da representación de cada palabra.

$$Q = N * \ D + D * \ \log_2(V)$$

Figura 2.12: Complexidade de CBOW

Skip-gram é similar a CBOW solo que en vez de predecir unha palabra en función dun contexto, trata de maximizar a clasificación dunha palabra en base a outra palabra da mesma frase. Esta aproximación é capaz predecir palabras anteriores e posteriores dentro dun rango respecto a unha palabra dada. Dado que as palabras máis cercanas están máis relacionadas que as lonxanas con respecto á palabra actual danlle máis pesos ás primeiras collendo máis exemplos deste tipo do conxunto de entrenamiento. Neste caso o valor Q calcúlase coa fórmula 2.13 sendo C o rango de palabras que é capaz de predecir, notese que canto máior sexa o rango máior será complexidade computacional.

$$Q = C * \ (D + D * \ \log_2(V))$$

Figura 2.13: Complexidade de Skip-gram

2.4 Temporal Word Embeddings With a Compass TWEC

TWEC explota a evolución dos embeddings de Word2Vec no tempo. Outras aproximacións anteriores teñen a desvantaxe de ser pouco eficientes no proceso de entrenamento ou ser algoritmos complexos e difíciles de entender para xente que non sexa experta no tema. TWEC por outra banda propón unha heurística para entrear modelos Word2Vec [5]. A heurística consiste en empregar un aliñador atemporal (*compass*) como referencia para entrenar representacións en momentos determinados.

Os desenvolvedores de TWEC decidiron empregar un modelo CBOW por funcionarlles mellor con pequenos *datasests* que os modelos Skip-gram. No modelo CBOW os *embeddings* do context $\overrightarrow{\overrightarrow{c_j}}$ son codificados na matriz de pesos de entrada C da rede neuronal mentras que os *embeddings* obxetivo $\overrightarrow{\overrightarrow{u_k}}$ son codificados na matriz de pesos de saida U. Supoñendo un corpus diacrónico D dividido en n porcións temporáis D^{t_i} e sendo $1 \geq i \geq n$. O entrenamento de TWEC divídese en dúas fases ilustradas na figura 2.14. Primeiro constrúense as dúas matrices atemporáis C e U aplicando o modelo CBOW orixinal a todo o corpus D. A continuación para cada porción temporal D^{t_i} primeiro inicialízanse a matriz de pesos de saida U da rede neuronal cos embeddings obxectivo previamente entrenados da matriz U. A continuación executase CBOW actualizando os *embeddings* de contexto na matriz de entrada C^{t_i}. Este

proceso repítese para cada porción temporal.

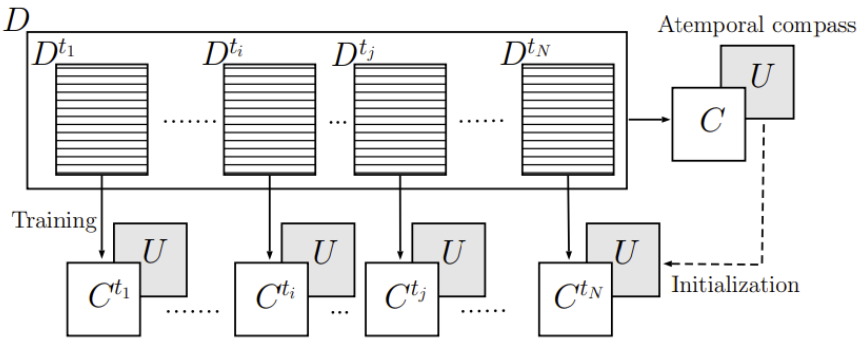


Figura 2.14: Esquema do modelo TWEC

Dada unha porción temporal D^{t_i} a segunda fase do proceso de entrenamento pódese formalizar para un único exemplo de entrenamento $\rangle w_k, \gamma(w_k)|$ / D^t como probema de optimización 2.15

$$\max_{C^t} \log P(w_k \| \gamma(w_k)) = \sigma(\overrightarrow{\overrightarrow{u_k}} \times \overrightarrow{\overrightarrow{c_\gamma}}^t(w_k))$$

Figura 2.15: Problema optimización

onde $\sigma(w_k) = \rangle w_{j_i}, \ldots, w_{j_M}|$ representa as M palabras do contexto W_k que aparecen en D^t ($\frac{M}{2}$ e o tamaño da ventá do contexto), $\overrightarrow{\overrightarrow{u_k}}$ / U e o ***embedding*** atemporal obxectivo da palabra w_k e 2.16 é a media dos ***embedding*** temporales do contexto $\overrightarrow{\overrightarrow{c}}^t_{j_m}$ da palabra contextual w_{j_m}.

$$\overrightarrow{\overrightarrow{c_\gamma}}^t(w_k)) = \frac{M}{2}(\overrightarrow{\overrightarrow{c}}^t_{j_1} + \times\times\times + \overrightarrow{\overrightarrow{c}}^t_{j_M})^T$$

Figura 2.16: media dos ***embeddings*** para unha palabra nun contexto

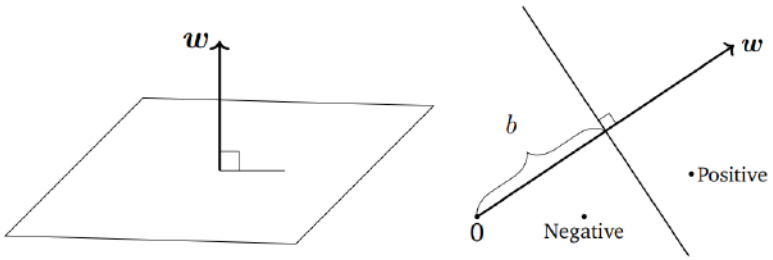
A única diferencia con respecto ao CBOW clasico é a optimización da matiz de pesos C^t. O proceso de entrenamento maximiza a probabilidade de, dado contexto dunha palabra nun fragmento temporal particular, poder predecir a palabra usando a matriz atemporal obxetivo U.

2.5 Outros algoritmos intelixentes

Alen dos conceptos dos que se falou anteriormente, os cales son esenciáis para entender o que se quere facer neste traballo, compre falar dalgúns algoritmos intelixentes que se empregaron durante a realización do traballo para a tarefa de clasificación. Os algoritmos que se empregaron foron.

2.5.1 Suport Vector Machine

As máquinas de soporte vectorial o SVM son un tipo de algoritmos intelixentes, de aprendizaxe supervisado e de comportamento non estocástico. O seu funcionamento razóase xeométricamente, susténtase en conceptos como o producto interior e a proxección. Estos algoritmos poden aproximar problemas de clasificación binaria. Sendo o vector D de numeros reáis $\mathbb{R}$ que conforman as características dun exemplo. Definimos o predictor da forma $f : \mathbb{R}^D \Rightarrow \}{+}1, \quad 1\langle$. Sendo $x_n \ / \ \mathbb{R}^D$ coas súas respectivas etiquetas $y_n \ / \ \}\ \ 1, {+}1\langle$ o conxunto de exemplos de entrenamento, cos que o algorítmo tratará de minimizar o erro na sua predicción. Como vemos, o plantexamento inicial a nivel conceptual é semellante o das redes neuronáis, pero veremos que a forma de plantexar e resolver as ecuacións é diferente.



(a) Hiperplano de separación 3D (b) Proxeción do hiperplano

Figura 2.17: Problema simplificado

Nun problema simplificado, Os SVM tratan de definir un hiperplano capaz de separar dúas clases a partir dunha serie de exemplos de entrenamento. Un hiperplano é un subespacio affin de dimensión $D \quad 1$ sendo D a dimensión do correspondente espazo vectorial. Por exemplo para realizar una separación lineal, definimos unha función para o noso hiperplano, ver figura 2.1, no que $x \ / \ \mathbb{R}^D$ é un elemento do espacio dos datos e $w \ / \ \mathbb{R}^D$ é un parámetro da función.

$$f : \mathbb{R}^D \Rightarrow \mathbb{R} \tag{2.1a}$$

$$x \Leftrightarrow \rangle w, x| + b \tag{2.1b}$$

Esta función define un vector con dirección ortogonal ó hiperplano e con magnitude igual ó seu punto de intersección, de tal forma que mediante esta función podemos determinar a posición do hiperplano no espazo. Agora definimos una función que lle permite separ as dúas clases no problema de clasificación binaria.

$$\}x \ / \mathbb{R}^R : f(x) = 0\langle \tag{2.2}$$

Como vemos na figura 2.17b o vector w é un vector normal ó hiperplano e b é a intersección. Podemos derivar que w é normal ó plano collendo calquera par de exemplos x_a e x_b do hiperplano e ver se o vector que definen é ortogonal o vector w .

$$f(x_a) \quad f(x_b) = \rangle w, w_a| + b(\rangle w, w_b| + b) \tag{2.3a}$$

$$= \rangle w, x_a \quad x_b| \tag{2.3b}$$

Dado que x_a e x_b son puntos do hiperplano esto quere decir que $f(x_a) = 0$ e $f(x_b) = 0$ e isto implica que $\rangle w, x_a \quad x_b| = 0$. Dado que dous vectores son ortogonais cando o seu producto interior é 0 queda demostrado que w é normal ó hiperplano. Clasificamos os exemplos dependendo de que lado do hiperplano se sitúan. Por exemplo, definindo un exemplo de test x_{test} calculamos o valor de $f(x_{test})$ e clasificamos o exemplo como $+1$ se $f(x_{test}) \rightarrow 0$ e 1 noutro caso. Durante o entrenamento queremos que os patróns positivos queden no lado positivo do hiperplano 2.4 e que os patrons negativos queden no lado negativo do hiperplano 2.5.

$$\rangle w, x_n| + b \rightarrow 0 \ \ when \ \ y_n = +1 \tag{2.4}$$

$$\rangle w, x_n| + b < 0 \ \ when \ \ y_n = \quad 1 \tag{2.5}$$

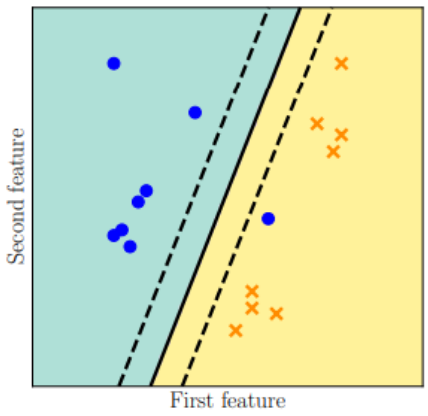
Frecuentemente, as dúas condicións acostuman a vir nunha soa ecuación.

$$y_n(\rangle w, x_n| + b) \rightarrow 0 \tag{2.6}$$

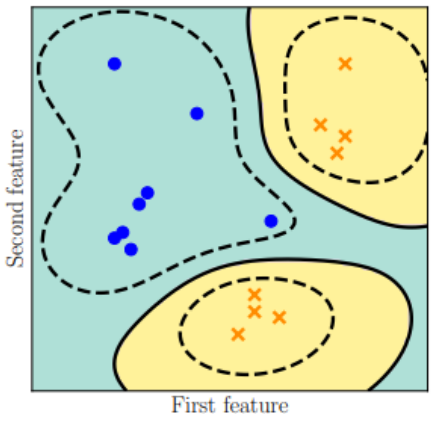
Partindo destes conceptos, realízanse unha serie de asuncións e plantéxase unha solución preliminar chamada *primal SVM*. Esta aproximación é computacionalmente moi ineficiente, crecendo esta linealmente co número de características dos exemplos. A aproximación *dual SVM* dá solución a este problema xa que habilita o uso de Kernels. Os kernesls son computacionalmente moi eficientes, son definidos como funcións $k : \mathbb{X}x\mathbb{X} \quad > \mathbb{R}$ para os cales existen un espacio de Hilbert $\mathbb{H}$ e $\phi : \mathbb{X} \quad > \mathbb{H}$ de características tal que:

$$k(x_i, x_j) = \rangle \phi(x_i), \phi(x_j)|_{\mathbb{H}} \tag{2.7}$$

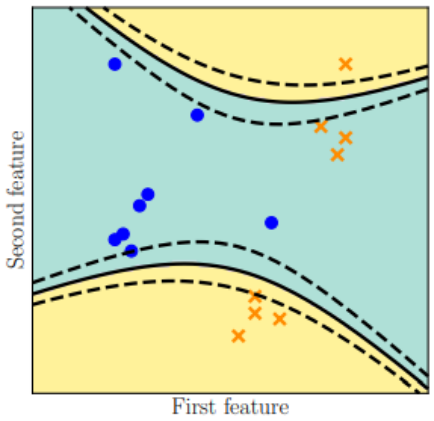
A continuación móstranse as funcións de decisión de varios kernels, nótese que ainda que a función de decisión no é lineal o problema subxacente ó que dan solución é para un hiperplano que fai unha separación lineal.



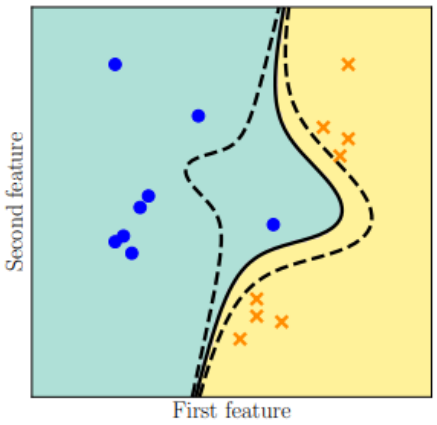
(a) kernel lineal



(b) kernel rbf



(a) kernel polynomico grao 2



(b) kernel polinomico grao 3

2.5.2 LSTM Recurent neural network model

As redes recurrentes son un tipo de redes neuronáis as cales introducen redundancia na estrutura das conexións da rede, desta forma logran recordar pasos previos a un patrón introducido.

No diagrama 2.20 pódese ver á esquerda un fragmento enrolado e á dereita un fragmento desenrolado dunha rede neuronal recurrente, no que A mira por uns valores x_t de entrada e devolve un valor h_t de saida, a información pode fluir dun paso t_i ao seguinte paso t_{i+1}.

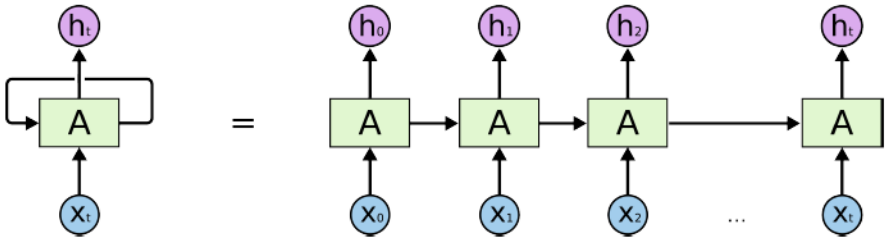


Figura 2.20: Esquema desenrolado dunha RNN

As RNN pretenden conectar o suceso presente cos sucesos previamente acontecidos, ás veces soamente precisamos coñecer a información máis recente e neste caso funcionan con éxito. Pero hai casos nos que necesitamos máis contexto, non chega coa aprendizaxe a curto prazo, precisamos que as redes poidan aprender a información a longo prazo. En teoría as RNN poden lograr este obxectivo pero na práctica non o fan.

As redes *Long Short Term Memory* ou simplemente as chamadas redes LSTM teñen éxito nesta tarefa. Estas son un tipo especial de RNN que foron introducidas por Hochreiter e Schmidhuber no 1997 [13]. Este tipo de redes foron especialmente deseñadas para eludir o problema da memoria a longo prazo. Lembrar a longo prazo é practicamente o seu comportamento por defecto.

Para presentar a estructura das redes LSTM empregaremos a seguinte notación 2.21.

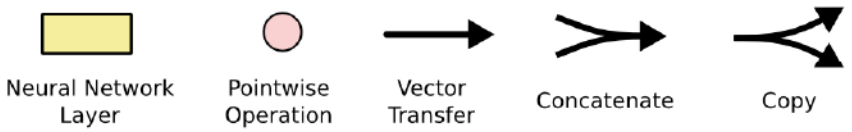


Figura 2.21: Notación

Todas as RNN teñen a estructura dunha cadea de módulos repetidos. Nas RNN estandar, este módulo ten unha estructura tan sinxela coma a capa dunha rede neuronal cunha tanxente hiperbólica como función de activación.

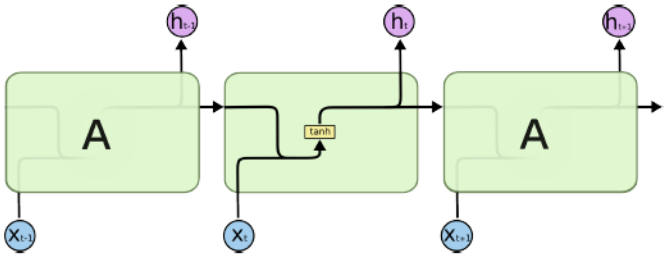


Figura 2.22: RNN Simple

As LSTMs tamén teñen esta estructura en cadea, pero no lugar da tanxente hiperbólica, o módulo repetido ten unha estructura diferente, algo máis complexa. En vez dunha única capa esta rede terá catro capas que interacturán entre si nunha forma especial.

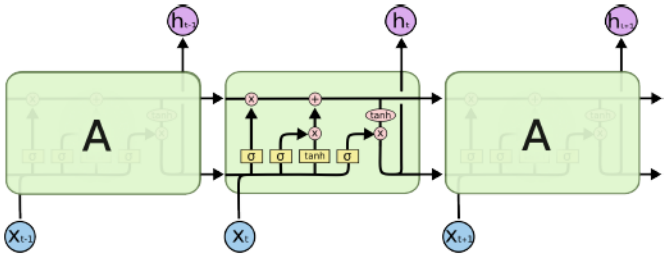
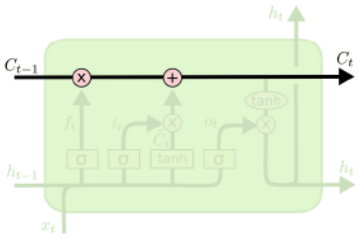
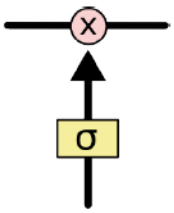


Figura 2.23: LSTM cadea

O funcionamento clave das redes LSTM ven dado pola liña superiór do diagrama 2.24a, a cal actúa como canle do fluxo da información. A rede ten a capacidade de engadir ou retirar información ao estado da célula a cal regúlase mediante portas. As portas son unha forma de permitir pasar a información de forma opcional, están compostas por capas de redes neuronáis con función de activación sigmoide seguidas dunha operación de multiplicación.

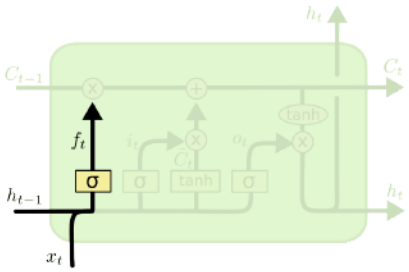


(a) Celda de estado



(b) Gate

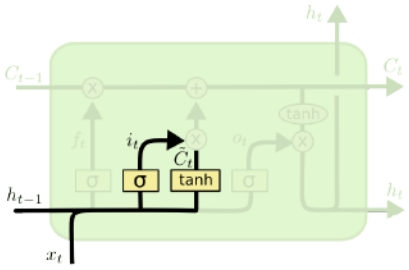
O primeriro paso da rede LSTM é decidir que información vaise deixar ir do estado da célula. Esta decisión ten lugar nunha capa cunha sigmoide chamada "capa da porta de olvido" f_t. Esta mira os valores recibidos h_{t-1} e x_t e devolve un valor entre 0 e 1 para cada número do estado da célula C_{t-1}. O valor 1 representa o manter por completo ese valor e o 0 representa olvidar por completo ese elemento.



$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

Figura 2.25: Capa da celula de olvido

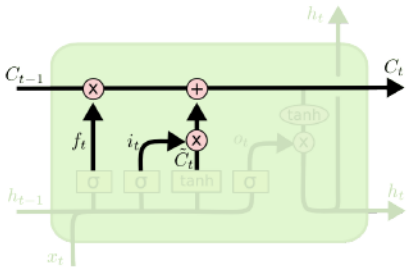
O seguinte paso é o de decidir con qué información nos ímos quedar no estado da célula, isto faise en dúas partes. Primeiro está a "capa da porta de entrada" i_t a cal decide que valor se vai a actualizar. Despóis unha capa taxente hiperbólica crea un vector candidato de novos valores, $\tilde{C}_t$, que podería engadirse ó estado. No seguinte paso combinaranse ambos para crear unha actualización para o estado.



$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$

Figura 2.26: Capa células de entrada

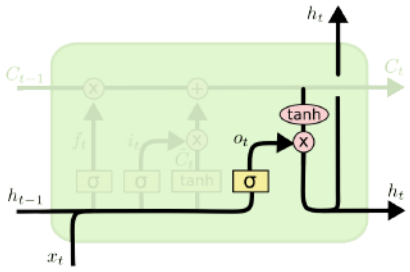
Agora é o momento de actualizar o antigo valor do estado da célula C_{t-1} co novo valor $\tilde{C}_t$. O paso anterior xa decidiu qué facer, e so precisamos actualizar o valor. Isto se fai multiplicando o antigo valor do estado por f_t olvidando as cousas que anteriormente decidiuse olvidar. Logo súmase o valor de operar $i_t \subseteq \tilde{C}_t$. Estes son os novos valores candidatos, escalados por canto se decidiu actualizar cada valor do estado.



$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

Figura 2.27: Actualización dos valores

Finalmente decídese o que se devolve á saída. Esta saída devolverase en función do estado da célula, pero filtrándoo. Primeiro pásase pór unha capa sigmoide a cal decide que partes do estado da célula vanse devolver. Logo pásase ó estado da célula a través dunha tanxente hiperbólica, para forzar que a saída fluctúe entre -1 e 1, e a multiplicamos pola saída da capa sigmoide de forma que so se devolven as partes que se decidiron devolver.



$$o_t = \sigma\left(W_o\left[h_{t-1}, x_t\right] + b_o\right)$$
$$h_t = o_t * \tanh\left(C_t\right)$$

Figura 2.28: Saída do modelo

Capítulo 3

Tecnoloxías

Neste capítulo falarase das tecnoloxías e ferramentas que se empregaron durante a realización do proxecto. Estas ferramentas e tecnoloxías permitiron a correcta xestión e desenvolvemento de cada unha das etapas que conforman o proxecto. Algunhas delas empregáronse durante as etapas de experimentación, outras na codificación e outras durante a execución dos tests e avaliación dos resultados.

3.1 Plataforma

A decisión que se tomou en canto a qué plataforma empregar durante a realización deste proxecto veu motivada polas características do mesmo. Ó tratarse dun proxecto de investigación e ó empregar algoritmos xa implementados nunha plataforma concreta, reducíusenos o abano de posiblidades, ainda así a plataforma escollida é a adecuada xa que conta con todas as librerías necesarias para o correcto desenvolvemento do traballo.

3.1.1 Python

Python é unha linguaxe de programación interpretada cun sistema de tipado dinámico, isto permítelle ser unha linguaxe moi flexible e aos desarrolladores permítenos acurtar os ciclos de codificación e execución, xa que non temos que compilar o código coma nos pasa por exemplo con C ou Java. Endemáis é unha linguaxe que inclúe todo o necesario para a programación orientada a obxetos e, a diferencia doutras linguaxes como Java, permite a herdanza múltiple. Tamén inclúe un recolector de lixo que realiza a xestión da memoria por nós, ainda que dado o algoritmo que implementa pode obligarnos a facer nós a eliminación explícita de variables. Python inclúe todo o necesario para traballar con espazos vectoriáis. A súa idoneidade para este proxecto adquire outra dimensión ó ter en conta a cantidade de librerías específicas as que temos acceso como desenvolvedores, *numpy*, *scipy*, *astropy*...etc.

Unha das limitacións que nos encontramos con Python é ó traballar con multiprocesos, o *global interpreter lock* ou GIL bloquea o uso de máis dun *thread* por instancia de Python á vez. Este problema non é insalvable xa que algunhas librerías como *numpy* sáltanse o GIL ó estar directamente implementadas en C. Tamén dispoñemos dunha librería para o aproveitamento do multiproceso como é a librería *multiprocessing* que levanta unha instancia de Python para cada *thread* co que se quere traballar.

3.2 Ferramentas de Soporte

Usáronse diferentes ferramentas que serviron de apoio durante o desenvolvemento do traballo.

3.2.1 PyCharm

Pycharm é un IDE, *Integrated Development Environment*, comercializado por JetBrains. Esta empresa é a mesma que fixo Android Studio. Este IDE integra unha serie de utilidades como son a análise estático do código, a provisión ó usuario dunha interfaz gráfica para o control de versións ou a asistencia durante o traballo de codificación dunha serie de recomendacións para segirmos boas prácticas. Tamén tén funcións de autocompletado, depuración e *profiling*. Este IDE non é gratuito pero dispon de licencias para estudantes e profesores que nos dan acceso gratuito a toda a *suit* de ferramentas que ofrece JetBrains.

3.2.2 Visual Studio Code

Esta ferramenta creada por Microsoft conta cunha ampla comunidade de usuarios que desenrolan plugins e extensións. Unha desas extensións é *LatexWorkshop*, e foi empregada para a redacción da memoria deste proxecto.

3.2.3 Jupyter Notebooks

Jupyter é unha organización sen ánimo de lucro creada para desenvolver software de código aberto, servizos para computación e estándares abertos. Esta ferramenta é moi util especialmente cando se quere experimentar, visualizar gráficas e imaxes ou cando se quere probar algunha librería. Esta ferramenta funciona a modo de inteface gráfica entre o usuario e o que chaman un *kernel* que é unha peza de código encargada de procesar varios tipos de solicitude a través da rede. Neste sentido o *kernel* de Python é o principal, ainda que tamén dan soporte a *kernels* doutras linguaxes como lidia ou R, as principales ás que da soporte xunto a Python, aida que soporta moitas outras. O que se presenta nesa interface é una serie de celdas nas que se poden inxerir fragmentos de código ou texto en formato Markdown, logo

o kernel executara o contenido da celda e debuxará a resposta, en texto plano, imaxes ou o Markdown compilado. O formato de Jupyter notebook é un documento Json que xeralmente termina cunha extensión .ipynb. Pódense convertir a varios formatos de saída entre os que están LaTeX, PDF, html, Markdown... etc.

3.2.4 Git

Git é unha ferramenta de control de versións de código aberto, e distribuida DVCS (*Descentralized Version Control System*), pensado na eficiencia, a confiabilidade e a compatibilidade no mantemento de proxectos cun gran número de arquivos. A día de hoxe é unha das ferramentas de control de versións máis extendidas. Isto é debido a que as características básicas, como a confirmación de novos cambios, a ramificación, a fusión e a comparación de versións anteriores, están moito máis optimizadas que nas outras alternativas. Git almacena os arquivos baseándose no seu contido en vez do seu nome, que podería cambiar ó longo do proxecto. O formato de obxetos dos arquivos do repositorio Git emplea unha mestura de codificación delta (que almacena a diferencia de contidos entre versións) e compresión.

3.2.5 Taiga

Taiga é unha ferramenta de xestión de proxectos áxiles, en concreto neste traballo séguese a metodoloxía SCRUM. Taiga dispón de diferentes módulos que se poden activar e desactivar en función das necesidades do proxecto. Unha vez creado un proxecto, pódense engadir historias de usuario, ó xerar estas historias a estimación realízase por puntos ainda que pode facerse por roles. Seguindo a metodoloxía Scrum os ciclos de desenvolvemento do proxecto agrúpanse en Sprints ó final dos cales deberemos ter un producto funcional, e isto se fará dentro dun período de tempo determinado. Taiga permítenos ver o estado das tarefas nun taboeiro estilo Kanban, tamén proporciónanos unha gráfica na que se pode visualizar o progreso global do sprint.

3.2.6 Docker

A plataforma Docker é unha ferramenta de creación de contenedores. Estos contenedores son como pequenas máquinas virtuais nas que podemos configurar entornos illados e lixeiros dunha forma moi sinxela. A función da aplicación desta tecnoloxía ao proxecto é parecido á función dos entornos virtuais de Python pero extendido a todo o sistema operativo. Empregando Docker temos a capacidade de executar procesos de forma separada do resto do sistema. Docker ofrece un modelo de implementación mediante imáxes, desta forma pódense compartir fácilmente as aplicacións. Ó mesmo tempo Docker dispón dun repositorio chamado DockerHub no que pódense subir imaxes preconfiguradas. Mediante un Dockerfile pódese

definir unha serie de pasos par que se executen ó construir a imaxe ou arrancar a máquina.

3.3 Google News word2vec model

Esta ferramenta proporciona unha implementación eficiente das arquitecturas CBOW e skip-gram para computar representacións de palabras mediante vectores. En concreto estamos a falar dun modelo preentrenado cunha colección de noticias coas que conta Google. Este modelo pode ser empregado tanto para investigar as relacións entre palabras como para continuar cun desenvolvemento. Neste proxecto empregaremos este modelo para facer *fine-tuning* dos modelos word2vec que usa TWEC.

Capítulo 4

Proposta e Evolucións

Neste capítulo definirase a idea inicial deste proxecto. Explicarase con esquemas e fórmulas a idea inicial do proxecto e a investigación realizada. A continuación explicaranse as evolucións que se fixeron sobre a idea inicial e a súa motivación

4.1 Proposta

Este traballo ten por obxetivo estudar a evolución no tempo do significado das palabras de usuarios deprimidos e non deprimidos para así poder detectar de forma temperá cando un novo usuario está a desenvolver depresión. Para poder captar esta evolución, proponse empregar a ferramenta TWEC xa que demostrou poder captar a evolución do significado das palabras dun xeito máis eficiente que outras propostas. Ainda que o algoritmo TWEC é un elemento destacado do proxecto, so é unha peza máis do *pipeline* do clasificador. Para poder traballar na tarefa, houbo que implementar una serie de elementos funcionáis que van dende o preprocesamento dos datos do *dataset*, pasando polo entrenamento dos modelos word2vec, ó cálculo das deltas, á clasificación e á avaliación de resultados. Os diferentes elementos desde *pipeline* descrébense na sección 4.2, pero antes compre falar das características do *dataset* empregado.

4.1.1 Descripción do dataset

Primeiramente explicaremos o *dataset* de partida e de qué maneira traballamos con el. Este *dataset* é o mesmo que o presentado no concurso CLEF eRisk 2017 [2]. O dataset consta de textos escritos na rede social Reddit datados e asociados a usuarios anonimizados. Dispomos dun conxunto de exemplos de entrenamento, estos están divididos por usuario e por *chunks*, sendo un *chunks* un conxunto de publicacións dun usuario nun período de tempo. Chamaremos D ao total dos textos do conxunto de entrenamento, D^+ ao subconxunto etiquetado como positivo e D^- ao etiquetado coma negativo. Cabe destacar que o conxunto de

entrenamento está desbalanceado, sendo $\|D^-\| > \|D^+\|$

Estes documentos atópanse en carpetas. Hay unha carpeta para exemplos positivos e outra para exemplos negativos. Descendendo un nivel na xerarquía do sistema de arquivos, vemos que no seguinte nivel atópanse as carpetas dos *chunks*. Dentro desas carpetas, por exemplo na carpeta do *chunk1* atópanse os XML do fragmento temporal un de cada usuario. Cada un destes XMLs teñen unha estructura determinada vease a figura 4.1.

Figura 4.1: Estrutura dos XMLs

Xunto co conxunto de entrenamento contamos cun conxunto de exemplos de test, que tamén están datados e asociados a usuarios anonimizados. Igual que no caso de training os de test encóntranse dentro de carpetas pero non están divididos por positivos e negativos, neste caso dispomos dun csv cos valores verdadeiros de cada usuario. Hai unha carpeta por *chunk*, na cal hay un XML por cada usuario. Neste caso interésanos referirnos a estes textos por usuario, polo que empregaremos D^u para facer referencia ao conxunto de textos que dispomos dun usuario. Inicialmente ó haber un formato de presentación único para o conxunto de training e o de testing, non se desacoplou a lectura dos datos do resto do preprocesado, posteriormente ó ampliar o dataset veremos que houbo que desacoplar esta funcionalidade para que poidera afrontar diferentes estratexias de lectura.

4.2 Pipeline

Descompoñendo o experimento inicial podemos definir un *pipeline* coas fases de preprocesado, entrenamento dos modelos word2vec, o calculo das deltas e a avaliación, como se ilustra na figura 4.2. Estas fases explicaranse en detalle a continuación. Máis adiante engadiranse algunhas fases e modificaranse o funcionamento doutras.

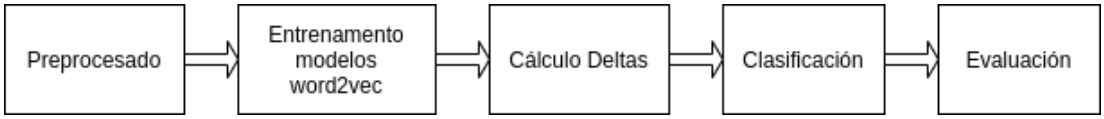


Figura 4.2: Pipeline do experimento

4.2.1 Preprocesado

O *dataset* anterior 4.1.1 presenta a información en documentos XML con etiquetas que introducen ruído no proceso de análise de textos. Endemáis as publicacións non se atopan completamente ordenadas por data. Esto fixo necesario unha etapa de preprocesado para poder adaptalos a un formato adecuado para a seguinte etapa do *pipeline* 4.2.2. Para esta tarefa, en primeiro lugar empregamos unha librería de *parsing* de documentos XML a JSON. A continuación para poder manexar e visualizar os datos con maior flexibilidade empregouse a librería Pandas, a través dos pandas.DataFrame, que ademáis permitiu ordear os textos pola súa data dunha forma moi sinxela. Por último xuntáronse as publicacións dos positivos nun .txt e dos negativos noutro .txt. Finalmente para cada subconxunto, troceáronse noutros .txt que conforman os *chunks* temporáis de cada clase. Na imaxe 4.3 vemos o exemplo de como sería o procesamento dos textos dos usuarios negativos.

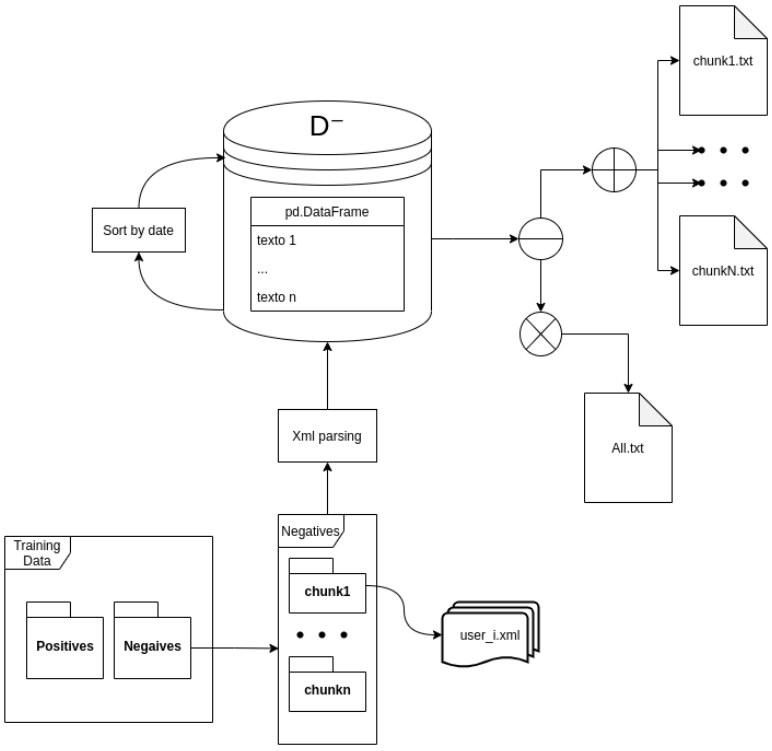


Figura 4.3: Esquema preprocesado subconxunto training D^-

4.2.2 Entrenamento modelos word2vec

Táboa 4.1: Notacións enpregadas

Notacións	
u	Refírese a un usuario do conxunto de test
w_k	Refírese a unha palabra do vocabulario
t_i	Chunk temporal i
C^+	Matriz compass dos exemplos de entrenamento positivos
C^-	matriz compass dos exemplos de entrenamento negativos
$C^+_{t_i}$	matriz chunk t_i dos exemplos de entrenamento positivos
$C^-_{t_i}$	matriz chunk t_i dos exemplos de entrenamento negativos
${}_uC^-_{t_i}$	matriz chunk t_i dun usuario de test
${}_+\overrightarrow{c_{w_k}}^{c}$	Vector do embeding dunha palabra w_k do contexto C^+
${}_-\overrightarrow{c_{w_k}}^{c}$	Vector do embeding dunha palabra w_k do contexto C^-
${}_+\overrightarrow{c_{w_k}}^{t_i}$	Vector do embeding dunha palabra w_k do contexto $C^+_{t_i}$
${}_-\overrightarrow{c_{w_k}}^{t_i}$	Vector do embeding dunha palabra w_k do contexto $C^-_{t_i}$
${}_u\overrightarrow{c_{w_k}}^{t_i}$	Vector do embeding dunha palabra w_k do contexto ${}_uC^-_{t_i}$

Na primeira aproximación, o entrenamento plantexouse da seguinte forma. Empregando a ferramenta TWEC, considéranse por separado os casos positivos e os casos negativos do conxunto de entrenamento, entrenandose un aliñador para cada un dos dous conxuntos. No contexto dos exemplos positivos temos a matriz $C^+ = [{}_+\overrightarrow{c_{w_1}}^{c}, \times\times\times, {}_+\overrightarrow{c_{w_k}}^{c}, \times\times\times, {}_+\overrightarrow{c_{w_n}}^{c}]$. Onde cada ${}_+\overrightarrow{c_{w_n}}^{c}$ correspóndese co ***embedding*** dunha palabra w_k do vocabulario C^+_w do contexto do ***compass*** positivo, o cal ten tamaño n.

Unha vez tense entrenado o aliñador, entrénanse os modelos temporáis de cada un dos ***chunks***, de forma que a matriz $C^+_{t_i}$ dos ***embeddings*** dun chunk temporal t_i definiríase como $C^+_{t_i} = [{}_+\overrightarrow{c_{w_1}}^{t_i}, \times\times\times, {}_+\overrightarrow{c_{w_k}}^{t_i}, \times\times\times, {}_+\overrightarrow{c_{w_m}}^{t_i}]$. Onde cada ${}_+\overrightarrow{c_{w_k}}^{t_i}$ correspóndese co ***embedding*** da palabra w_k do vocabulario $C^{t_i^+}_w$ do contexto dun chunk t_i dos exemplos positivos. Para o conxunto dos negativos faríase de forma equivalente sustituindo C^+, $C^+_{t_i}$ e $C^{t_i^+}_w$ polos seus respectivos C^-, $C^-_{t_i}$ e $C^{t_i^-}_w$.

Para o conxunto de test empregouse o aliñador C^- do subconxunto de exemplos negativos, xa que se estimou que este modelo permitiría que os ***embeddings*** temporáis entrenados a partir del tiveran un comportamento menos nervioso. Entón para cada usuario u teremos un conxunto de modelos, coas súas respectivas matrices de ***embeddings*** de palabras tal que para cada momento t_i cada usuario terá unha matriz $C^u_{t_i} = [{}_u\overrightarrow{c_{w_1}}^{t_i}, \times\times\times, {}_u\overrightarrow{c_{w_k}}^{t_i}, \times\times\times, {}_u\overrightarrow{c_{w'_m}}^{t_i}]$ sendo ${}_u\overrightarrow{c_{w_k}}^{t_i}$ o ***embedding*** da palabra w_k do vocabulario dos textos do usuario u no momento t_i.

A matriz $C_{t_i}^{u}$ ten m' columnas, que se corresponden co tamaño do seu vocabulario, compárandoo co tamaño do *compass* a relación é $m' \leq n^{-}$. Estas diferencias entre o tamaño dos vocabularios resulta relevante para entender algúns problemas cos que nos atopamos ó longo do proxecto. Na seguinte fase mediremos a distancia entre as palabras e crearemos así un novo conxunto de vectores chamados (deltas) con esta información.

4.2.3 Cálculo das deltas

Táboa 4.2: Extensión das notacións empregadas.

Notacións	
$_{+}\delta_{w_k}^{t_i}$	Distancia entre o *embedding* dunha palabra w_k na matriz do *compass* C^{+} e do *chunk* $C_{t_i}^{+}$
$_{-}\delta_{w_k}^{t_i}$	Distancia entre o *embedding* dunha palabra w_k na matriz do *compass* C^{-} e do *chunk* $C_{t_i}^{-}$
$\Delta_{t_i}^{+}$	E a matriz dos deltas do vocabulario dos usuarios positivos no instante t_i
$\Delta_{t_i}^{-}$	E a matriz dos deltas do vocabulario dos usuarios negativos no instante t_i
$\Delta_{w_k}^{+}$	E a matriz dos deltas dunha palabra w_k para os usuarios positivo
$\Delta_{w_k}^{-}$	E a matriz dos deltas dunha palabra w_k para os usuarios negativos

Unha vez tense cada un dos modelos entrenados, teremos a codificación dos vectores das palabras para cada instante de tempo, de cada un dos subconxuntos do conxunto de training e do conxunto de testing aliñados por separado. A continuación houbo que atopar unha maneria de comparar as evolucións do significado das palabras dos usuario de test coa evolución do significado das palabras dos usuarios positivos e negativos do conxunto de training. Para poder decidir quén e cándo está a desenrolar unha depresión. Para definir esta similitud empregamos a variable $\delta_{w_k}^{t_i}$ que representa cánto se moveu o vector dunha palabra w_k nun momento t_i determinado con respecto do vector da palabra k no aliñador. Na primeira aproximación empregamos a distancia de coseno 4.4 como medida de distancia.

$$ {}_{+}\delta_{w_k}^{t_i} = cos({}_{+}\overrightarrow{c_{w_k}^{c}}, {}_{+}\overrightarrow{c_{w_k}^{t_i}}) = 1 \quad \frac{{}_{+}\overrightarrow{c_{w_k}^{c}} \times {}_{+}\overrightarrow{c_{w_k}^{t_i}}}{{}_{+}\overrightarrow{c_{w_k}^{c}}_{\ 2} \times {}_{+}\overrightarrow{c_{w_k}^{t_i}}_{\ 2}} $$

Figura 4.4: Delta da palabra w_k para os usuarios positivos no momento t_i

Computando esta medida de distancia para cada ***chunks*** temos a matiz Δ. Se vemos a matriz como un conxunto de vectores columnas obtemos vectores $\Delta_{w_k}/w_k \ / \ \}1, \times\times\times, n\langle$ que codifica a evolución dunha palabra ó longo do tempo, por outra banda se vemos a matriz como un conxunto de vectores filas temos os vectores $\Delta_{ti}/i \ / \ \}1, \times\times\times, m\langle$ que codifica unha fotografía estática no momento t_i da evolución de todas as palabras da linguaxe. Para poder

traballar dentro dun espacio vectorial común, sempre coa mesma dimensionalidade, empregamos o vocabulario do *compass*. Así poderemos facer operacións alxebraicas cos vectores Δ_{ti} sen problemas. Para poder lograr isto, cando unha palabra w_k do vocabulario do *compass* non existe no vocabulario do *chunk* t_i dun usuario u, asúmese que ${}_u\delta^{t_i}_{w_k} = 0$ e dicir, o significado non mutou. Na figura 4.1 pódese ver como queda representada a matriz das deltas dos exemplos de training positivos.

$$\Delta^+ = \begin{array}{c} \\ t_1 \\ \vdots \\ t_i \\ \vdots \\ t_i \end{array} \begin{array}{c} \begin{array}{ccccc} w_1 & \cdots & w_k & \cdots & w_n \end{array} \\ \begin{pmatrix} +\delta^{t_1}_{w_1} & \cdots & +\delta^{t_1}_{w_k} & \cdots & +\delta^{t_1}_{w_n} \\ \vdots & & \vdots & & \vdots \\ +\delta^{t_i}_{w_1} & \cdots & +\delta^{t_i}_{w_k} & \cdots & +\delta^{t_i}_{w_n} \\ \vdots & & \vdots & & \vdots \\ +\delta^{t_n}_{w_1} & \cdots & +\delta^{t_n}_{w_k} & \cdots & +\delta^{t_n}_{w_n} \end{pmatrix} \end{array} \tag{4.1}$$

4.2.4 Clasificación

A continuación continúase coa etapa de clasificación na que computamos a distancia entre os deltas de cada usuario u e os deltas dos subconxuntos de entrenamento $+$ e $\quad$. Nun principio asúmese que os usuarios son non deprimidos no instante t_1, a medida que vamos comparando os deltas de cada un dos momentos t_i de cada usuario, se nalgún momento o $\Delta^u_{t_i}$ parécese máis ao $\Delta^+_{t_i}$, decídese a detección do deprimido no momento t_i. Esta función de decisión $f(u, +, \quad)$ formalízase como se indica a continuación.

$$f(\Delta^u_{t_i}, \Delta^+_{t_i}, \Delta^-_{t_i}) = \begin{cases} 1, t_i & \text{when } cos(\Delta^+_{t_i}, \Delta^u_{t_i}) \geq cos(\Delta^-_{t_i}, \Delta^u_{t_i})) \\ 0, t_1 & \text{when } cos(\Delta^+_{t_i}, \Delta^u_{t_i}) > cos(\Delta^-_{t_i}, \Delta^u_{t_i})) \end{cases} \tag{4.2}$$

4.2.5 Avaliación

A continuación lemos o arquivo cos valores de verdade e avaliamos os resultados dos experimentos. Para poder medir a bondade dos experimentos realizados neste traballo empregamos as mesmas métricas que definiron para o *workshop CLECE eRisk 2017* [2]. As medicións básicas *true positives* TP, número de clasificados como positivos sendo realmente positivos, *true negatives* TN, número de clasificados como negativos sendo realmente negativos, *false positives* FP, número de clasificados como positivos sendo realmente negativos e *false negatives* FN, número de clasificados como negativos sendo realmente positivos, empréganse para explicar o resto das métricas.

A **precision** indica o número de elementos correctamente clasificados como positivos en relación a o total de clasificados como positivos.

$$precision = \frac{TP}{TP + FP} \tag{4.3}$$

O **recall** indica o número de elementos correctamente clasificados como positivos en relación o total de verdadeiros positivos.

$$recall = \frac{TP}{TP + FN} \tag{4.4}$$

a **F-measure** ou a media armónica e unha métrica que combina *precision* e *recall*. É práctica para comparar o rendemento global dos algoritmos, xa que agrupa nun valor o resultado das dúas anteriores métricas.

$$F_\beta = (1 + \beta^2) \times \frac{precision \times recall}{(\beta^2 \times precision) + recall} \tag{4.5}$$

Cando a métrica F-measure preséntase con valor $\beta = 1$ chámaselle **F1**. Nesta configuración o que se asume é que importa por igual a precisión que a exhaustividade na avaliación dos resultados. Quizáis no caso deste proxecto tivera sido mellor unha métrica **F2** con $\beta = 2$ pero os datos da avaliación dos resultados do CLECE eRisk 2017 foron presentados coa métrica **F1**.

$$F_1 = 2 \times \frac{precision \times recall}{precision + recall} \tag{4.6}$$

A seguinte das métricas é a $ERDE$ foi definida ad-hoc para o problema de detección temperá da depresión. Ten en conta a rapidez ou lentitude coa que foi detectado o caso positivo para indicar unha medida de penalización. O parámetro k indica o número de exemplos introducidos antes de tomar a decisión en o parámetro d o que se decidiu. A función defínese da seguinte forma:

$$ERDE_o(d,k) = \begin{cases} c_{fp} & if\ \ d = positives\ \ AND\ \ groundtruth = negatives\ \ (FP) \\ c_{fn} & if\ \ d = negatives\ \ AND\ \ groundtruth = negatives\ \ (FN) \\ lc_o(k) \times c_{tp} & if\ \ d = positives\ \ AND\ \ groundtruth = negatives\ \ (TP) \\ 0 & if\ \ d = negatives\ \ AND\ \ groundtruth = negatives\ \ (TN) \end{cases} \tag{4.7}$$

A decisión do valor empregado para c_{fp} e c_{fn} depende do campo de aplicación dos FP e FN. Dado que temos un número de exemplos negativos varios ordes de magnitud superior a o número de exemplos positivos, para previr clasificadores que sempre clasifiquen como negativo temos que ter $c_{fn} >> c_{fp}$. Neste caso o arreglamos para que $c_{fn} = 1$ e poñemos c_{fp} dacordo á proporción de casos positivos no conxunto de test. o factor $lc_o(k) \times c_{tp}(/\ [0,1])$ codifica un coste asociado á tardanza na detección dos TP.

$$lc_o(k) \times c_{tp} = 1 \quad \frac{1}{1 + e^{k-o}} \tag{4.8}$$

Dado que a excesiva tardanza na detección é equivalente a non detectar o positivo, a función 4.8 é monótona crecente. O parámetro o regula o lugar no eixo das x no que a función de penalización medra máis.

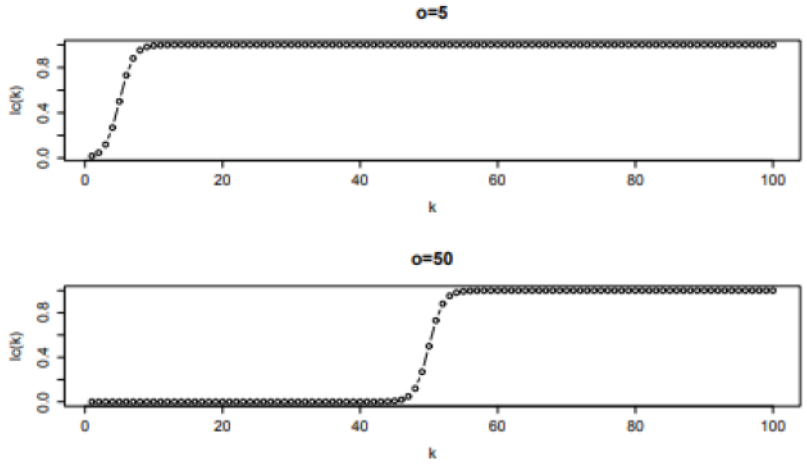


Figura 4.5: función de coste da detección tardía

4.3 Evolucións e melloras

A medida que avanzou a experimentación fomos mudando a idea inicial para tratar de mellorar os resultados, esto concretouse en modificacións que afectaron a diferentes partes do pipeline. A continuación describiremos cales foron e porqué se produciron cada unha destas modificacións.

4.3.1 Empregar un único compass

Coma xa se explicou, a idea inicial pasaba por empregar un aliñador para os exemplos positivos e outro para os exemplos negativos, e logo empregar o aliñador de negativos no entrenamento dos modelos do conxunto de *test*. O problema que ten esta aproximación é que para poder computar a medida de similaridade de coseno, os vectores delta teñen que ter a mesma dimensionalidade. Esta dimensionalidade ven dada polo vocabulario correspondente ó contexto de cada *compass*, o cal e diferente para o caso dos positivos e negativos. Tras discutilo nunha reunión decidimos que habría que empregar un único aliñador. Ademáis ó estar ambos subconxuntos de training no mesmo espazo aliñado debería captar mellor as diferencias entre a evolución na linguaxe de cada un deles. Isto traduciuse na modificación de varias etapas, a primeira a de preprocesado, xa que agora haberá un único documento que aglutinará todos os textos do conxunto de *training*, en vez dos dous .txt que había antes. Tamén afectará ó proceso de entrenamento de modelos word2vec, en vez de ter 2 aliñadores agora solo haberá un. Por último tamén afectará ó proceso do cálculo das deltas, que agora empregará so un aliñador. A mellora da versión cun aliñador fronte a versión con dous aliñadores faise evidente xa que soluciona o problema da diferencia de vocabulario. Para poder implementar efectivamente a proximación de dous aliñadores, habería que reducir os seus vocabularios ao común entre ambos. Nesta situación, se houbera unha palabra no vocabulario do usuario pero non houbera no vocabulario común dos *compass* teríamos que prescindir desta palabra para o cálculo das deltas e perdíamos esta información. En cambio, xuntando os textos nun único *compass* temos un vocabulario máis extenso. Como contrapartida, esta modificación supón un aumento na complexidade espacial do algoritmo, pero neste momento non supuso un gran problema.

4.3.2 Fine tunning

A segunda das modificacións propúsoxe noutra reunión, na que se discutiu o *fine tuning* coma unha posible solución para mellorar a clasificación. Esta técnica permite obter melloras significativas cando non se dispón de exeplos dabondo para entrenar os modelos de *embeddings*. Esta técnica consiste en, a partir dun modelo preentrenado con textos xenéricos, continuar entrenando o modelo cos datos do contexto expecífico. Para o caso do noso proxecto, empregamos un modelo preentrenado cos textos de *Google News*. As etapas que se

viron afectadas por esta modificación foron, en primeiro lugar a etapa de entrenamento de modelos word2vec, na que agora leerase un modelo preentrenado que se afinarán para crear o modelo do *compass*. A seguinte das etapas que se veron afectadas por esta modificación foi o a do calculo das deltas, xa que agora ó empregar un modelo cun vocabulario tan amplio, o aumento da complexidade espacial do que falábamos antes agora convírtese nun problema serio. Démonos conta de que o aumento da complexidade ven polo uso escesivo de celdas da matriz para aloxar valores cero, e dicir, empregar matrices densas para datos dispersos. A solución foi empregar a librería scipy.sparse, a cal ten varias implementacións de matrices dispersas. Estas implementacións almacenan a información en pares (fila , columna) asociadas a un valor, de forma que o almacenamento é óptimo.

4.3.3 Filtrado do vocabulario

Baseándonos no léxico creado por Saif M. Mohammad e Peter D. Tuney [14] fixemos unha selección das palabras que tiñan algún contido emocional. Desta forma empregamos este novo léxico para filtrar o vocabularo co que traballamos durante a investigación. O obxetivo era obviar características que non aporten información para a tarefa concreta de detección temperá da depresión. Buscábase acadar un beneficio doble, reducir a dimensionalidade dos deltas dos chunks, e seleccionar características máis relevantes. Este filtrado implementase na etapa de cálculo das deltas, xa que si se fixera antes, os modelos de *embeddings* perderían parte da información para crear a representación das palabras.

4.3.4 SVM como clasificador

Ata o momento, na etapa de clasificación empregamos a distancia de coseno para decidir cando as deltas dun usuario do conxunto de test parécense máis os deltas dos usuarios positivos ou negativos nun momento dado. Esta forma de clasificar é moi rudimentaria xa que estamos resumindo a información que tiñamos dos vectores Δ nunha única medida, a cal non ten por que ser decisiva para a classificación. Por estes motivos decidiuse modificar a etapa de clasificación introducindo unha máquina de soporte vectorial. Estes algoritmos permítenos configurar unha tolerancia ó ruído, Ademáis teñen en conta a posición dos patróns no espazo, en contraposición co clasificador por distancia de coseno que só tiña en conta o ángulo que forman os vectores delta con respecto ó aliñador. Poriba destas vantaxes, as máquinas de soporte vectorial son capaces de captar as diferencias entre os patrons de cada clase e definir unha función que maximice a marxe entre a rexión de separación e os vectores de soporte, que marcan os límites das clases. Para acadar os mellores hiperparámetros para o algoritmo fixemos unha validación cruzada, desta forma o algoritmo vai entrenándose e validándose para cada unha das combinacións de parámetros que se queren probar.

4.3.5 LSTM como clasificador

Dende o principio quixemos empregar a información da evolución do vocabulario no tempo, pero ata o momento non empregamos un algoritmo que retivera información temporal. Para esta tarefa as redes neuronáis recurrentes son as apropiadas. LSTM é un modelo de redes neuronais recurrentes que captura información tanto a curto como a longo prazo. Este algoritmo obríganos a presentarlle os exemplos por bloques, de tal forma que os exemplos, os $\Delta_{t_i}^+$ e $\Delta_{t_i}^-$, teñen que presentarse de n en n de forma que un bloque sería $[\Delta_{t_i}, \times\times\times, \Delta_{t_{i+n}}]$. Para implementar este algoritmo soamente tivemos que crear unha nova clase de clasificador seleccionable ó final do *pipeline*.

4.3.6 Inclusión de novos datasets para o entrenamento

Ó empregar os deltas do vocabulario como as características dun exemplo Δ_{t_i} tivemos que lidiar co problema da dimensionalidade, e dicir, tíñamos máis dimensións por cada exemplo que o número total de exemplos. O primeiro paso que demos para tratar combatir este problema foi o de aumentar o tamaño dos exemplos de entrenamento. Isto fixémolo engadindo o *dataset* de entrenamento do eRisk 2018 [3], o cal presentábase co mesmo formato que o dataset de *testing*. Neste punto decidimos modificar a etapa de preprocesamento do *pipeline*, desacoplamos a lectura da etapa de preprocesamento creando dúas clases distintas, unha determinaba a etiqueta de cada exemplo en función da carpeta na que se encontra e a outra en función dun arquivo csv que contén o seu valor de verdade. Desta forma puidemos definir nun arquivo de configuración qué estratexia adoptar para cada *dataset*.

4.3.7 Filtrado máis estricto

Anteriormente fixeramos un filtrado buscando palabras que contiveran significado emocional, o problema é que moitas das palabras con contido emocional non resultaron ser adecuadas para discernir entre deprimidos e non deprimidos. Endemáis disto continuamos a ter moi poucos exemplos comparandoos coa enorme dimensionalidade do problema e necesitamos reducir o noso vocabulario a outro máis específico. Para lograr esto, centrámonos só nas palabras con emocións positivas e negativas, filtrando as demáis na etapa de cálculo das deltas.

Capítulo 5

Metodoloxía e Desenvolvemento

A planificación dentro dun proxecto de software resulta fundamental para poder levalo a cabo con éxito. Este proxecto tivo unhas características especiáis xa que se tratou dunha investigación entorno a unha tecnoloxía en concreto. Polas características do proxecto decidíuse empregar unha metodoloxía áxil, en concreto o *framework* Scrum.

5.1 Metodoloxía Áxil

A metodoloxía áxil conforman un enfoque para a toma de decisións nos proxectos de *software*, que se refiren aos métodos de enxeñería de software baseados no desenvolvemento iterativo e incremental, no que en cada iteración realízanse os traballos de deseño, desenvolvemento e test, desplege e revisión. Neste marco os requisitos e as solucións evolucionan co tempo según as necesidades do proxecto. Así o traballo é realizado mediante a colaboración de equipos autoorganizados e multidisciplinarios, inmersos nun proceso compartido de toma de decisións a curto prazo.

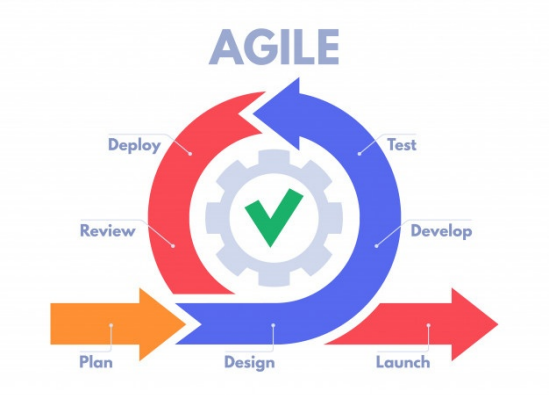


Figura 5.1: Esquema Axil

5.1.1 Scrum

Scrum é un *framework* ou marco de traballo para o desenvolvemento áxil de software. Ten o a sua orixe nun estudo realizado en 1986 por Hirotaka Takeuchi e Ikujiro Nonaka [15], a partir do cal empezou a ser adoptado por diferentes empresas. É un proceso no que se aplican de forma regular un conxunto de boas prácticas para o traballo colaborativo e que aporta unha serie de beneficios.

- **Cumplimento da expectativa**: O cliente establece as súas expectativas e o valor que lle aporta requisito/historia do proxecto.
- **Flexibilidade a cambios**: Alta capacidade de reacción ante os cambios de requerimentos xerados polas necesidades do cliente ou evolución do mercado.
- **Redución do time to market**: O cliente pode empezar a empregar as funcionalidades máis importantes do proxecto ainda que non este completamente rematado
- **Maior calidade de software**: A metódica do traballo e a necesidade de obter unha versión funcional despóis de cada iteración, axuda á obtención dun software de calidade superior.
- **Maior produtividade**: Conséguese entre outras razóns pola eliminación da burocracia e a motivación no equipo que proporciona o feito de que sexan autónomos para organizarse.
- **Maximizar o retorno da inversión**: Prodúcense o software priorizando os requisitos que maximicen a ralación valor custo.
- **Redución de tempos**: Permite coñecer a velocidade media dun equipo por sprint, polo que é posible estimar cando vai a estar lista unha funcionalidade que ainda está no *backlog*.
- **Redución de riscos**: O feito de levar a cabo as funcionalidades de maior valor e de coñecer a velocidade coa que o equipo avanza no proxecto, permite abordar riscos de xeito anticipado.

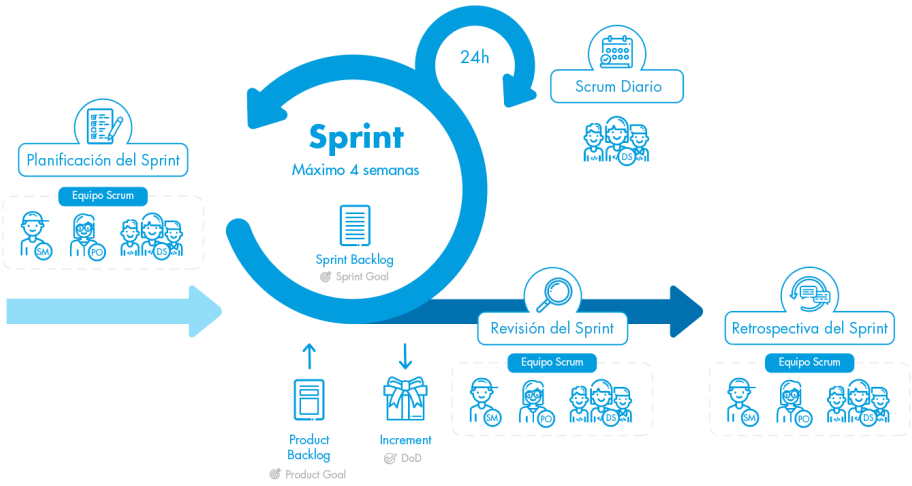


Figura 5.2: Esquema Scrum

Os Roles

A forma de traballar de Scrum é posible grazas a úns roles ben definidos, que coordinan as suas funcións e experiencias ó servizo de cada nova labor. O tres axentes máis importantes do éxito de Scrum son:

1. **Product owner** Quen ocupa este rol ten a responsabilidade de dialogar co cliente e garantir que o equipo esté a traballar según o previsto. Este papel debe asumirse de forma individual, non debe existir loita de poder en torno a esta competencia.

2. **Scrum Master** É o perfil o cal comproba que os integrantes do proxecto acadan os seus obxetivos realizando un proceso de acompañamento constante. Dende a sua posición, axuda ao grupo a adquirir as habilidades necesarias para o perfecto desenvolvemento das súas funcións.

3. **Scrum Team** É a base desta metodoloxía dado que é un equipo capaz de traballar de forma autónoma en beneficio do producto.

Sprint

En Scrum un proxecto execútase en ciclos de tempo curtos de entre 2 e 4 semanas. Cada iteración ten como saída un producto funcional, que pode ser entregado ao cliente cando o solicite. O proceso parte dunha lista de obxetivos/requisitos do producto do que o cliente escolle cales teñen maior prioridade en función de cales lle aportan máis valor respecto o seu custe, e quedan repartidos en iteracións e entregas.

Planificación do Sprint

1. **Selección de requisitos** O cliente presenta ó equipo a lista de requisitos priorizada. o equipo pregunta ó cliente as dúbidas, e selecciona os requisitos máis prioritarios que prevé completar

2. **Planificación da iteración** O equipo elabora a lista de tarefas da iteración necesarias para realizar os requisitos seleccionados. A estimación do esforzo realízase de forma conxunta e son os propios membros do equipo os que se autoasignan as tarefas.

Execución do sprint

Cada día o equipo realiza unha reunión de control ou $Daily$ para informarse uns aos outros cómo vai o desenvolvemento da sua tarefa. Durante o *sprint* o *Scrum Master* encargase de que o equipo poda manter o foco para cumplir cos seus obxetivos.

Inspección e adpatación

O último día da iteración realízase a reunión de revisión. Ten dúas partes:

1. **Revisión (demostración)** O equipo presenta o cliente os requisitos completados durante a iteración en forma de incremento do producto, preparado para ser entregado. En base ao resultado e aos cambios no contexto do proxecto o cliente realizara as adaptacións necesarias de maneira obxetiva, replanificando o proxecto.

2. **Retrospectiva** O equipo revisa a sua forma de traballar e cales son os problemas que lles poden impedir progresar e mellorar a productividade. O facilitador encargarase de eliminar ou escalar os obstáculos identificados que estén mais aló do ámbito do equipo.

Ó longo do proxecto iranse xerando unha serie de elementos que plasman os traballos realizados.

- **Product backlog** É o listado de todas as tarefas que se pretenden facer ó longo do desenvolvemento do proxecto. Todas as tarefas do proxecto deben estar dentro do *product backlog* para que estén visibles polos membros do equipo e se poida ter unha visión de conxunto de todo o que se espera realizar no proxecto.

- **Sprint backlog** Neste caso este elemento consiste na lista de historias de usuario pendentes por realizarse nun sprint. Permite visualizar a lista de tarefas dun sprint.

5.2 Adaptación de Scrum ao proxecto

Para poder adaptar Scrum a este proxecto houbo que simplificalo un pouco. Isto é debido ao equipo limitado a unha única persoa. Esta adaptación realizouse da seguinte forma:

- O rol de **Product owner** foi levado a cabo polos directores do proxecto.
- O rol de **Scrum Master** foi levado a cabo polos directores do proxecto.
- O **Equipo de desenvolvemento** conformao unha única persona Manuel Couto Pintos.
- Os **Sprints** duran unha semana, correspóndenlle 37.5 puntos de historia.

Historias de usuario e puntos de historia

As historias de usuario son a descripción das funcionalidades do proxecto, que permiten levar un control do qué e o porqué do que se está facendo no traballo diario dos membros do equipo. Normalmente formúlanse dende o punto de vista dun dos roles do proxecto. O común é que sigan o formato:

$$\text{Como } [rol] \text{ quero } [funcionalidad] \text{ para } [motivo] \tag{5.1}$$

No caso deste proxecto empregaranse as historias de usuario para recoller os requisitos funcionáis que conforman o *product backlog* que se esperan ter realizados ó finalizar o proxecto. Para designar o esforzo que conleva unha historia de usuario ou canto esforzo pódese realizar ó longo dun sprint, emprégase unha medida abstracta chamada puntos de historia. Esta medida abstracta non debería basearse simplemente nunha estimación das horas que se espera que leve un proxecto, senon que debe representar os riscos e os problemas que poden acontecer durante o desenvolvemento.

5.3 Desenvolvemento

O desenvolvemento do proxecto realizouse seguindo a metodoloxía Scrum, mediante a definición de historias de usuario, estas constitúen os requisitos do proxecto e cando foi preciso, foronse incluindo de forma progresiva ao *product backlog*. Ó comenzo de cada sprint engadíronse ao *product backlog* unha serie de HU, a continuación do *product backlog* escolleronse unha serie de HU a realizar durante a semana.

5.3.1 Historias de usuario

Facendo unha recopilación dos requisitos que se foron definindo ó longo do proxecto, podemos ver na seguinte táboa, na cal presentamos cada un deles, obviando a parte da HU de ”como cliente preciso ...”, xa que sempre se trata do mesmo rol. Desta forma definíronse as seguintes historias de ususario:

Historia de Usuario	codigo	puntuación
... procesar textos xml e visualizalos	HU1	7.5
... poder entrenar os modelos do algoritmo TWEC	HU2	7.5
... calcular as deltas	HU3	7.5
... empregar a distancia de coseno para clasificar	HU4	7.5
... implementar as métricas para comparar o algoritmo	HU5	7.5
... executar o proxecto en entornos Docker	HU6	7.5
... poder entrenar un único *compass* para training	HU7	7.5
... poder executar o algoritmo en paralelo	HU8	7.5
... partir dun modelo preentrenado (fine-tunning)	HU9	15
... reducir o uso de ram	HU10	22.5
... incrementar a velocidade do algoritmo	HU11	22.5
... empregar un único chunk para positivos e outro para negativos	HU12	7
... mellorar o deseño do código	HU13	37.5
... clasificar mediante SVM	HU14	15
... evaluar o clasificador SVM con variacións no pipeline	HU15	22.5
... clasificar mediante LSTM	HU16	15
... evaluar o clasificador LSTM con variacións no pipeline	HU17	22.5
... poder incluir novos datasets	HU18	7.5
... filtrar de forma máis estricta	HU19	7.5
... facer validación cruzada do SVM	HU20	15
... evaluar os novos hiperparámetros	HU21	22.5
... facer validación cruzada do LSTM	HU22	15
... evaluar os novos hiperparámetros	HU23	22.5
... adaptar a validación cruzada o proxecto	HU24	15

5.4 Sprints

Os sprints agruparon os traballos realizados cada semana, ao fin das cales tívose listo un producto funcional. A continuación describiranse os traballos realizados en cada un dos sprints.

Sprint 1

Neste sprint planificouse realizar as tarefas HU1, HU2, HU3, HU4 e HU5. Que se corresponden co *pipeline* da primeira aproximación do proxecto. As tarefas correspóndense coa implementación do preprocesado dos textos, do entrenamento dos modelos word2vec de TWEC, do cálculo das deltas, da clasificación e da avaliación do pipeline. Durante a semana detectouse un problema ao tratar de comparar os deltas positivos cos deltas creados co aliñador negativo e decidiuse cambiar a aproximación a un único aliñador. Ó final da semana so se puideron completar as tarefas HU1, HU2, HU3 e HU4. Quedando pendente a tarefa de avaliación do *pipeline*. Ó final desta semana comentouse a posiblidade de xuntar os aliñadores do algoritmo TWEC nun único algoritmo, endemáis decidiuse empregar o entorno docker para executar os experimentos. Cada unha destas tarefas engadíronse o *product backlog*.

Sprint 2

Na segunda semana planificouse rematar coa tarefa HU5, e engadir as tarefas HU6, HU7 e HU8. Estas tarefas novas correspóndense coa proposta feita na reunión da semana anterior e coa tarefa de paralelizar a execución do programa, para poder realizar experimentos de forma máis eficiente. O final da semana puidéronse rematar todas as tarefas. Na reunión desa semana propúxose empregar a técnica de *fine-tunning* para partir dun modelo preentrenado e ver se os resultados do algoritmo melloraban.

Sprint 3

Na terceira semana planificáronse realizar as tarefas HU9, durante o transcurso da semana démonos de conta que había un problema co uso da memoria RAM, o consumo excesivo da memoria volátil facía que o operativo cortara a execución do algoritmo. Houbo que crear a tarefa HU10 para solucionar este problema. Para o final da semana puideronse rematar as dúas tarefas.

Sprint 4

Na cuarta semana planificouse realizar as tarefas HU11 e HU12. Estas correspóndense coa modificación do *pipeline* para empregar un único chunk para positivos e outro para negati-

vos, e coa mellora da eficiencia do algoritmo. Esta mellora realizouse mediante a reducción da lectura escritura en disco do algoritmo. Esta semana sobrou algo de tempo e decidiuse experimentar con MSV nun *notebook* de jupyter. Nesta semana démonos de conta que o proxecto non era flexible a modificacións e decidiuse engadir a tarefa HU13 ó *product backlog*.

Sprint 5

Nesta semana planificouse a realización da tarefa HU13, esta tarefa requeriu da realización dun deseño que veremos no capítulo 6. Seguido da realización do deseño veu a implementación, alongandose esta ata o final da semana. O testeo do código quedou pendente para a seguinte semana. Endemáis decidiuse engadir as tarefas HU14 e HU15 ó *product backlog*.

Sprint 6

Nesta semana planificouse rematar coa tarefa pendente da semana anterior e comenzar coa tarefa HU14 e HU15. Ó longo da semana foise testando o código resultante da refactorización e correxindo os erros que se foron atopando. Ó longo desta semana tamén se puido implementar o novo clasificador SVM, o correspondente á tarefa HU13. Finalmente, avaliouse o funcionamento do algoritmo probándoo con diferentes versións do *pipeline*, travallo que se corresponde coa tarefa HU15. En vista dos resultados obtidos na avaliación deste clasificador, plantexouse probar outros clasificadores polo que se engadiu a implementación do clasificador LSTM ao *product backlog* com HU16 e HU17.

Sprint 7

Nesta semana planificouse a realización das tarefa HU16 e HU17. A implementación do clasificador foi un pouco máis tedioso que o do SVM xa que trátase dun algoritmo algo máis complexo que require da utilización da librería Keras. Endemáis esta librería emprega Tensorflow e algunhas das máquinas nas que se lanzaron os experimentos non tiñan tarxetas gráficas dedicadas con núcleos Cuda. Isto provocou que durante a utilización do algoritmo xurdiran errores, os cales eran descoñecidos o principio, xa que so recibíamos un *core dumped* baleiro. Ó final da semana faltaba realizar parte da avaliación, pero os resultados que tiñamos, esperábanse mellores, polo que se decidiu tratar de melloralos incluindo novos datos de entrenamento. Tamén decidiuse que habería que filtrar de forma máis estricta o vocabulario e mellorar a escolla de hiperparámetros mediante a validación cruzada. Estas taréfas engadíronse o *product backlog* seguindo a mesmo orde como HU18, HU19 e HU20 mais a avaliación do resultado da validación cruzada como HU21.

Sprint 8

Ó comenzo desta semana planificouse rematar co resto da avaliación e a realización das tarefas HU18, HU19 e HU20. Durante a semana puidose rematar co resto da avaliación e facer as tarefas HU18 e HU19. Estas puideronse facer rápido gracias as modificacións previas no deseño. Ó final da semana implementouse a validación cruzada e , coma sobrou algo de tempo, comezouse coa avaliación desta aproximación, esta arroxounos resultados moi optimistas, do 0.9 de precisión, polo que nos fixo sospeitar que había algo que funcionaba mal. Nesta semana decidiuse que intentaríamos facer a mesma validación cruzada co algoritmo LSTM, polo que se engadiron o *product backlog* as tarefas HU22 e HU23 correspondentes a implementación e a avaliación da validación cruzada.

Sprint 9

Ó comezo desta semana planificouse rematar coa avaliación e realizar as tarefas HU22 e HU23. Durante a semana os resultados da validación cruzada seguían sendo moi optimistas pero o avaliar o algoritmo cos hiperparámetros seleccionados os resultados foron moito piores. Tras investigar o problema detectouse que a forma coa que se decide se un usuario é positivo ou negativo conformaba outro clasificador porriba do SVM. Polo que habería que adaptar o método para validar este segundo clasificador. Decidiuse devolver as tarefas HU22 e HU23 ao *product backlog* e engadir a tarefa HU24 a cal consistía en realizar esta modificación na validación cruzada. O resto da semana adicouse a realización desta modificación.

Sprint 10

O comenzo desta semana planificouse rematar coa tarefa HU24 e retomar a tarefa HU21. Durante a semana rematouse a adapatación da validación cruzada o proxecto e realizouse a avaliación da validación cruzada do clasificador SVM.

5.4.1 Recursos e Custos

Recursos humanos

Como xa se falou na sección 5.2 houbo 3 persoas implicadas no proxecto as cales se repartiron os roles de *Product Owner Scrum Mastrer*, *Product Owner* e o equipo de desenvolvemento, que neste caso estivo formado por unha persoa.

Recursos materiáis

Os recursos materiáis confórmanos o ordenador personal do membro do equipo de desenvolvemento. A maiores contarase cun cluster formado por 7 servidores proporcionados polo

iRlab. Estos servidores empregaranse durante o periodo de experimentación.

Recursos software

As tecnoloxías que se empregaron durante a realización o traballo foron xa descritas na sección 3. Estas tecnoloxías teñen licencias gratuitas ou de estudante polo que non supuxeron ningun custo adicional.

Custos

Nesta sección farase o cálculo do custo estimado da realización deste traballo. Primeiro presentaremos a estimación dos custos por unidade/hora para or recursos humanos xunto co custo e período de amortización dos servidores.

Equipo	Director	Analista	Desemvolvedor
Manuel Couto Pintos	-	26€	20.8€
Javier Parapar	45.5€	-	-
Alfonso Landin	45.5€	-	-

Táboa 5.1: Custos por hora para os recursos humanos. [9]

Harware	Vida útil	Uso	Valor(€)	Total(€)
Servidor 1	4 anos	2 mes	12000€	500€
Servidor 2	4 anos	2 mes	4500€	187.5€
Servidor 3	4 anos	2 mes	4500€	187.5€
Servidor 4	4 anos	3 mes	2500€	156.25€
Servidor 5	4 anos	3 mes	2500€	156.25€
Servidor 6	4 anos	3 mes	2500€	156.25€

Táboa 5.2: Custos por hora para os recursos humanos

O custo dos salarios basaronse no informe *studio salarial - Sector TIC en Galicia2015-2016* [9]. Os custos da seguridade social veñen incluidos no cálculo do salario.

ROL	$tiempo(h/Sprint)$	NºSprints	Coste(€h)	Total(€)
Desenvolvedor	37.5	10	20.8€	7800€
Analista	37.5	10	26€	975€
Director	8(ambos)	10	45€	3600€
Servidores	-	-	-	1343.75
Total	-	-	-	9143.75€

Táboa 5.3: Previsión de custos

Capítulo 6

Deseño

A hora de realizar un proxecto o deseño é unha das etapas máis importantes. Unha vez se ten os requisitos haberá que plantexar como se estructurará o código. Para poder facer un deseño que sexa escalable e mantible compre seguir principios de deseño e boas prácticas da programación orientada a obxetos. Entre outros, están os principios **SOLID** e os principios **KISS** endemáis dos patróns de deseño.

6.1 Partes da Ferramenta

Durante o deseño, a ferramenta foi decomposta en partes, cada unha das cales ten a súa función e responsabilidade única e independente do resto. As funcionalidades impleméntanse nos Managers os cales son usados polos Command para efectuar as tarefas do *pipeline*. O container compartese por toda a aplicación para realizar varias tarefas e o scheduler permitirá a execución secuencial ou paralela dos traballos da ferramenta. A continuación presentase un UMLde alto nivel das relacións entre as clases principais 6.1.

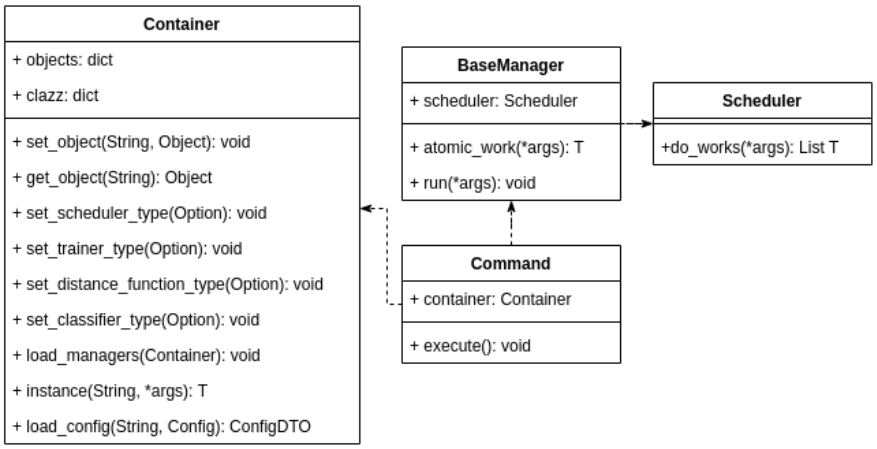


Figura 6.1: UMLde alto nivel

6.1.1 Container

Este patrón atópase con frecuencia en linguaxes coma Java. Por poñer un exemplo, este patrón emprégase en frameworks como Spring para implentar a inxección e a inversión de dependencias. Neste proxecto, emprégase reflexión para instanciar clases de forma dinámica, situación na que este patrón é tremendamente util. Pero tamén se empregan para facilitar a implementación doutros patróns, como é o caso do Singleton e Factory dos cales o último utilizámolo para instanciar o Scheduler, o DeltaMethod e o Classifier ou para o transporte e almacenamento de variables como acontece coas instancias das clases xeradas coas factorías ou os datos creados en cada unha das etapas do *pipeline* que deberán pasarse á seguinte etapa. Podemos ver as firmas dos métodos da clase Container na figura 6.1.

6.1.2 Command

A ferramenta poderá executar calquera parte do *pipeline* de forma independente, e poderemoslle decir que partes en concreto queremos que execute. Para poder lograr isto implementaremos un patrón comando. Isto lógrase mediante unha clase abstracta Command a cal podemos ver na figura 6.1 da que heredarán diferentes clases concretas asociadas mediante un diccionario ó nombre dos comandos. A aplicación ten un *parser* o cal leerá deste diccionario as claves aceptadas como comandos. No momento de executarse, o usuario introducirá os comandos e a ferramenta recollerá do diccionario as clases dos Command para encolalos por orde, este será a mesmo orde co que se executará o método exec de cada comando. A responsabilidade do comando é traer os datos necesarios para a súa tarefa, chamar ó mánager concreto que encapsula a lóxica da funcionalidade e gardar os novos datos xerados.

CommandRead

A clase concreta CommandRead é unha implementación da superclase Command. Esta clase fai uso do ReaderManager para ler os datos dos diferentes *datasets*. Ó mesmo tempo a clase CommandRead está composta polas clases CommandReadTrain e CommandReadTest cada unha delas encárgase de ler os datos de *training* e os de *testing* respectivamente, xuntas conforman o comportamento da clase CommandRead. Ao ter encapsuladas cada unha das accións de lectura, podemos leer de forma independente o de forma conxunta cada un dos *datasets*. A continuación vemos o fragmento do diagrama UML destas clases 6.2.

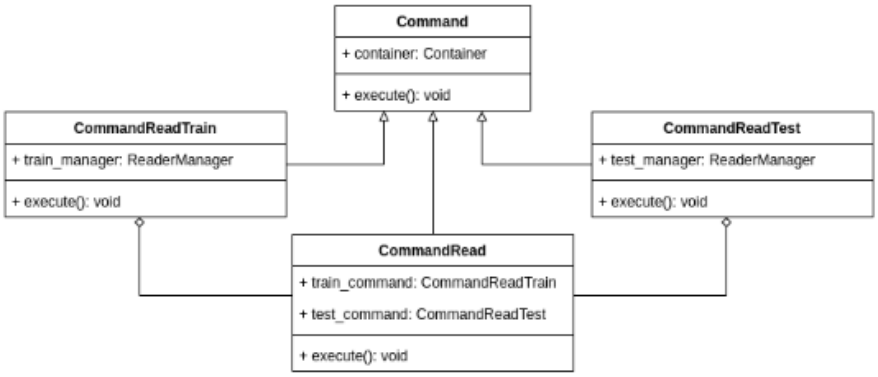


Figura 6.2: UML das classes Reader

CommandWrite

A clase concreta CommandWrite é unha implementación da superclase Command a cal fai uso do WriterManager, o esquema é similar ó do ReaderManger. Esta clase está composta por un CommandWriteTrain e CommandReadTest para escribir os .txt que se empregarán na seguinte etapa do *pipeline*. Compre diferenciar a funcionalidade do CommandWriter da do CommandWriterPerUser. A primeira agrupa por carpetas os exemplos de entrenamento en función de se son positivos ou negativos. A segunda agrupa por carpetas os exemplos de entrenamento por usuario. Este grupo de comandos permite escribir o conxunto de entrenamento, o conxunto de test ou ambos dunha vez. Pódese ver o diagrama de clases a continuación. 6.3

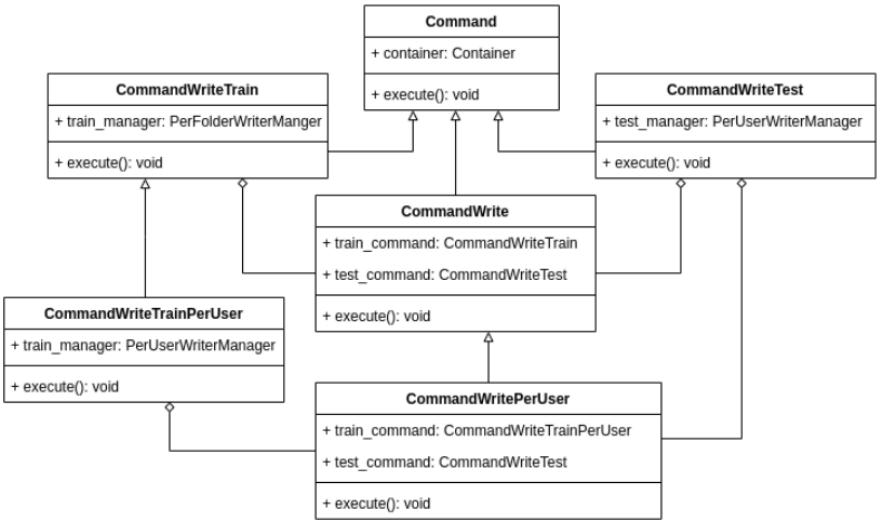


Figura 6.3: UML das clases Writer

CommandModel

O comando CommandModel segue a mesma estructura que o CommandWriter.Emprega o ModelManager para operar, e permite o entrenamento dos modelos de *embeddings* de *training* de *testing* ou ambos dependento de se chama a CommandTrainModel, CommandTestModels ou CommandModels. Tamén permite procesar o conxunto de exemplos de entrenamento agrupandoos pola súa clase ou agrupándoos por ususario dependendo de se chama a CommandTrainModel, CommandTrainPerUserModel ou ComandPerUserModel. A continuación podemos ver o UML. 6.4

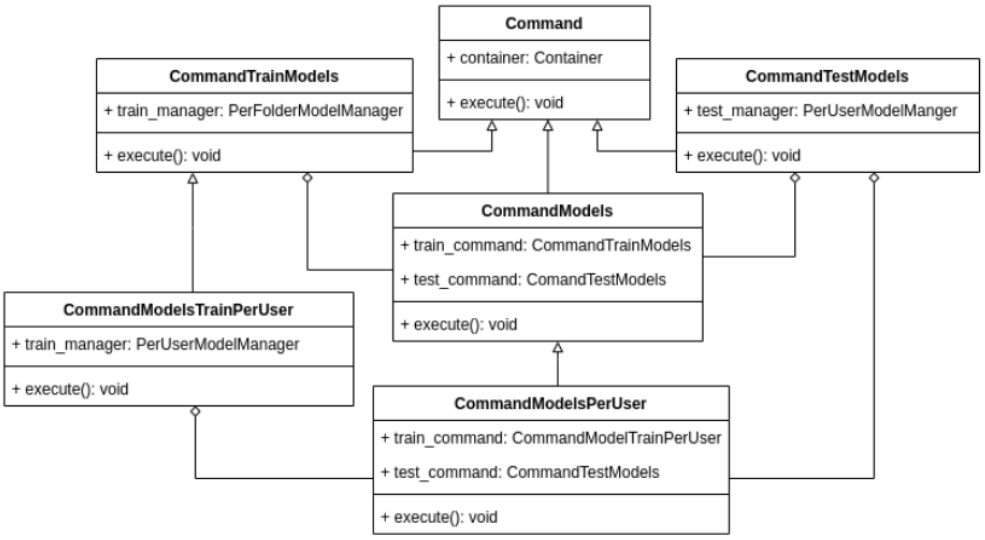


Figura 6.4: UML das clases Model

CommandDelta

Seguindo co *pipeline* da ferramenta e sen despegarnos da estructura coa que vimos traballando, temos o CommandoDelta, o cal fai uso do DeltaManager. Igual que ata agora permítenos traballar cos *datasets* de *training* e *testing* de forma separada ou conxunta. Tamén permítenos traballar co conxunt de entrenamento agrupando os exemplos pola súa clase ou por usuario se chama a CommandDeltasTrainPerUser ou CommandDeltasPerUser . A continuación pódese ver o UML. 6.5

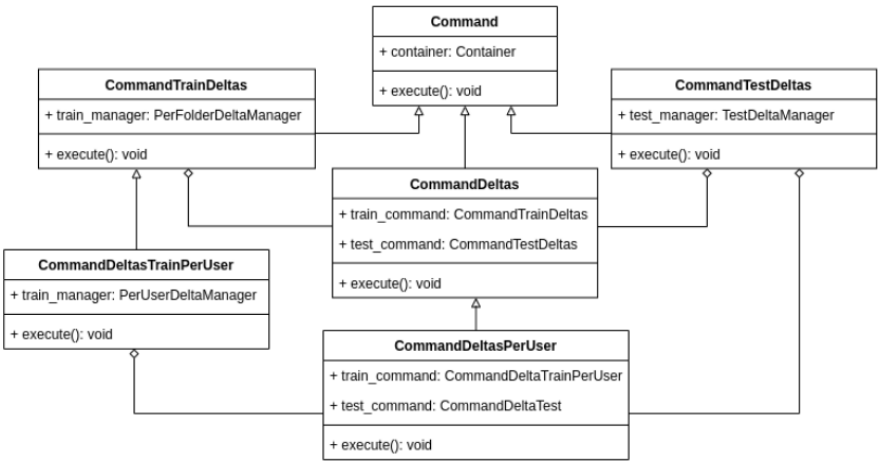


Figura 6.5: UML das clases Delta

CommandStent

Este comando fai o mesmo que o CommandModel e o CommandDelta xuntos pero reducindo o número de operacións de lectura-escritura. Incumple en certa medida o principio de responsabilidad única, pero o traballar con discos en rede fíxose necesario optimizar a aplicación sacrificando lixeiramente a calidade do deseño. Ademáis do CommandStent, tamén poderemos executar por separado a operación stent no training set e no test set mediante os comandos, CommandStentTrain e CommandStentTest e igual que nos anteriores comandos podemos executar as operacións agrupando pola clase dos exemplos ou por usuario mediante os comandos CommandStentPerUser e CommadnStentTrainPerUser. Podese ver o diagrama UML a continuación 6.6.

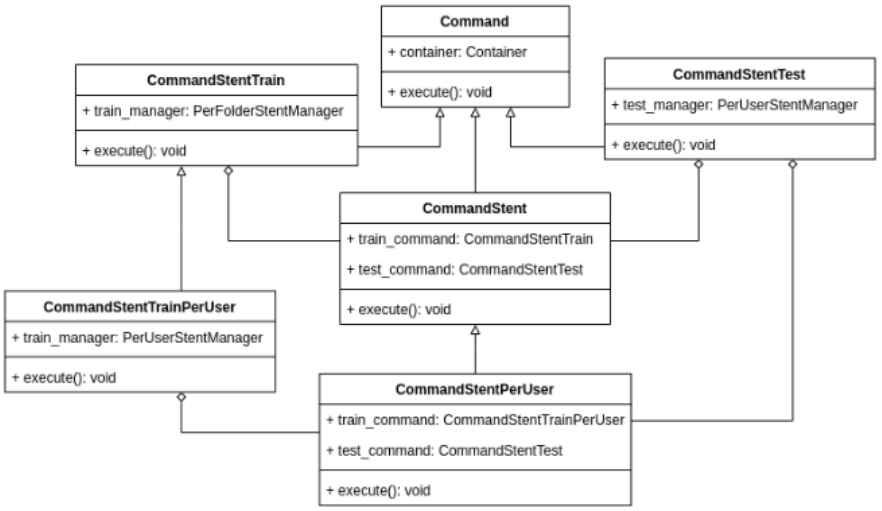


Figura 6.6: UML das clases Stent

CommandClassify e CommandEvaluate

O commando Classify correspóndese coa tarefa de clasificación, como sempre, o commando fai uso do seu mánager correspondente. Neste caso traballarase cos *datasets* en conxunto cun único comando xa que empregarase o conxunto de *training* para entrenar os modelos e co conxunto de testing para comprobar se xeneralizou ben. Finalmente o comando evaluate encárgase de calcular as métricas de rendemento através do seu mánager

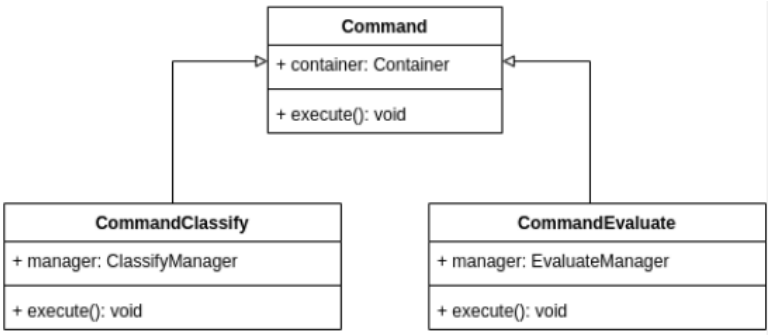


Figura 6.7: UML das clasess Classify e Evaluate

6.1.3 Manager

A lóxica específica de cada parte do *pipeline* atópase encapsulada nos mánagers. Estes implementan un patrón estratexia, de forma que hay unha clase abastracta BaseManager da cal herdan unha serie de clases concretas. Cada unha destas clases concretas implementan a lóxica de cada fase do *pipeline*. Internamente fan uso dun planificador que lles permite executar o traballo computacionalmente máis custoso de forma secuencial ou de forma paralela, dependendo da opción que seleccionara o usuario. Endemáis poderá facer uso doutras clases para a realización dos traballos. A continuación vemos cada unha delas.

ReaderManager

Esta clase é a encargada de ler os *datasets*. Os datasets pódense presentar de varias formas, por este motivo queremos procesar cada un deles de forma independiente. Para poder facer isto empregamos un arquivo para configurar as características de cada *dataset*. Neste arquivo indicamos o tipo de clase que empregaremos para procesar o XML, a súa ruta, e dependendo do tipo de *dataset*, empregaremos o valor de verdade para a ruta ou o *path* ao arquivo cos valores de verdade.

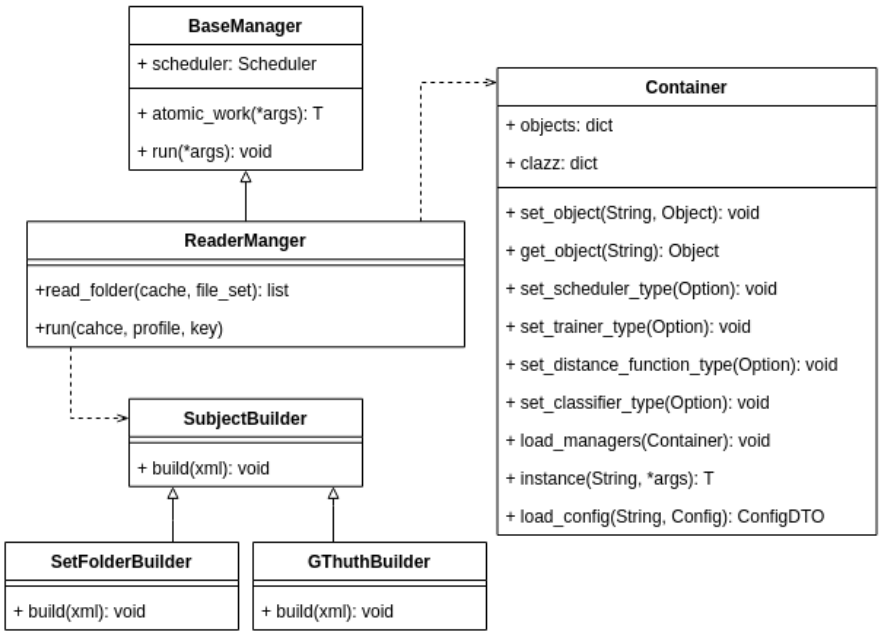


Figura 6.8: UML da classe ReaderManager

WriterManager

O WriterManager e o encargardo de reordear os datos recollidos dos *datasets* por data e volver a escribirlos para que a ferramenta TWEC os poida ler para entrenar os modelos. Podemos quer ordealos por usuario tendo un conxunto de *chunks* por cada un ou podemos querer agrupalos por positivos e negativos e ter un conxunto de *chunks* para cada grupo. Para lograr esta versatilidade temos unha clase abstracta da que herdan dúas clases concretas.

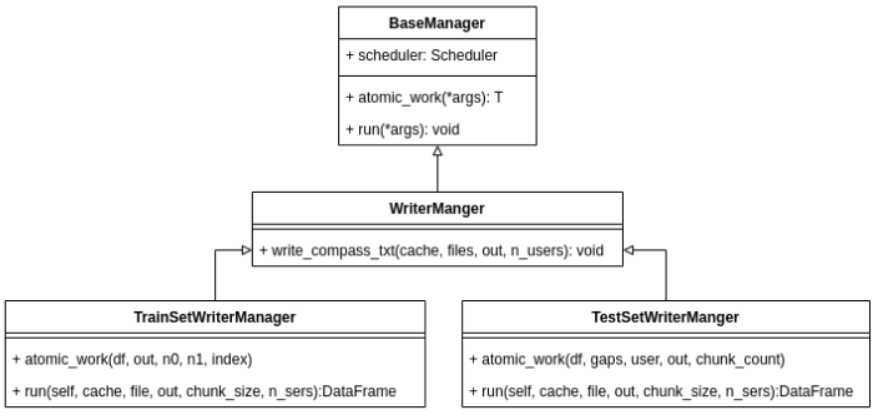


Figura 6.9: UML da classe WriterManager

ModelManager

O ModelManager é o encargado de ler os datos anteriormente procesados e utilizalos para o entrenamento do aliñador e os chunks temporáis. Estos chunks temporáis poden ser do conxunto de *training* ou o de *testing*, e non se procesan igual. Endemáis desto, o usuario ten a opción de facer *fine tuning*, para lograr esto temos a clase AlignerTrainer da cal herdan dúas clases que implementan cada unha destas estratexias.

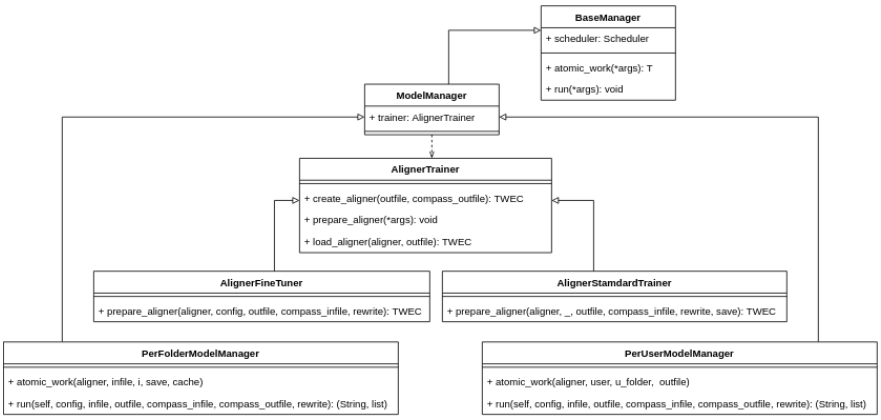


Figura 6.10: UMLda classe ModelManger

DeltaManager

O DelataManager é o encargado de calcular canto se moveron os *embeddings* das palabras con respecto ao aliñador. Este Mánager é unha clase abstracta da cal herdan dúas clases concretas, unha para *training* e outra para *testing*. Endemáis fai uso dunha clase DeltaFunction a cal encapsula a función que calcula canto se moveron os embeddings das palabras e o filtrado do vocabulario, se este é seleccionado polo usuario.

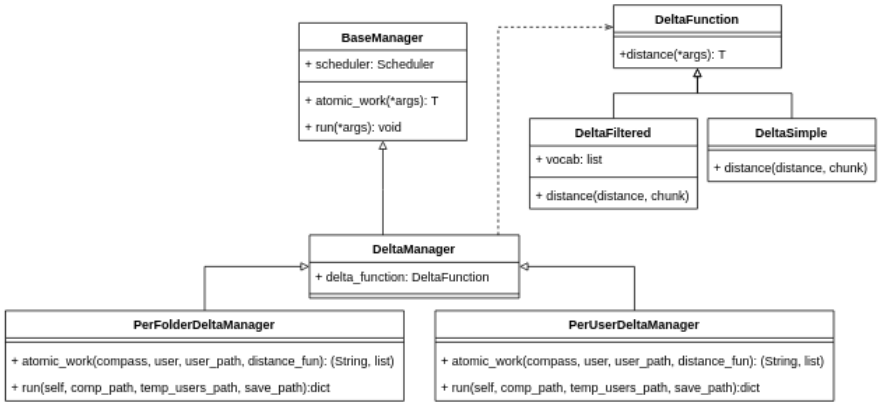


Figura 6.11: UMLda classe DeltaManager

StentManager

O StentManager encapsula a funcionalidade do ModelManager e DeltaManager xuntos nunha clase. Igual que estas conta con unha estratexia para o conxunto de entrenamento e outra para o conxunto de *testing*. Ademáis emprega as mesmas clases auxiliares DeltaFunction e TrainAligner.

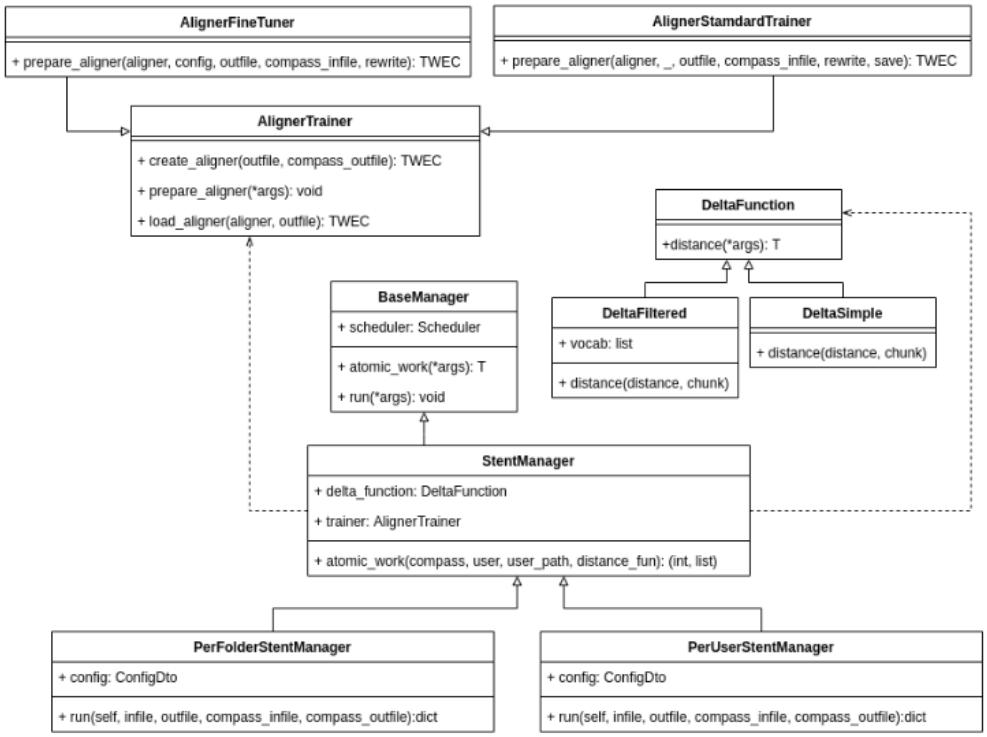


Figura 6.12: UMLda classe StentManager

ClassifyManager

O ClassifyManager é o encargado de realizar o entrenamento do clasificador e logo clasificar os usuarios do conxunto de *testing*. Fai uso da clase ClassifierMethod, a cal ten unha implementación concreta para cada un dos tipos de clasificadores que se empregaron ó longo do proxecto.

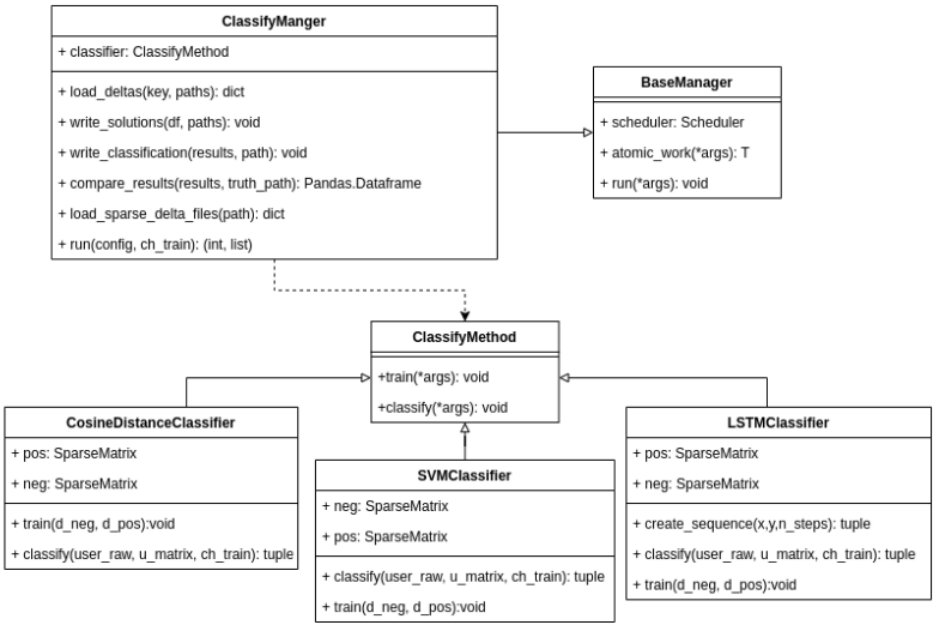


Figura 6.13: UMLda classe ClassifyManager

6.1.4 Scheduler

O Scheduler permite ó usuario escoller se quere executar as tarefas en paralelo ou de forma secuencial, para logralo implementa unha estratexia. Emprégase un *flag*, o cal se non está presente a ferramenta carga a estratexia secuencial, pola contra se está presente cargará a estratexia paralela. A estratexia escollida compártese co resto da aplicación através do container.

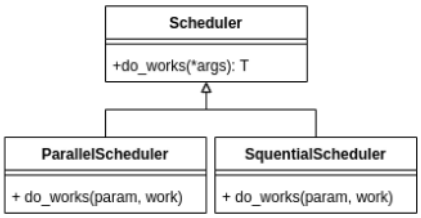
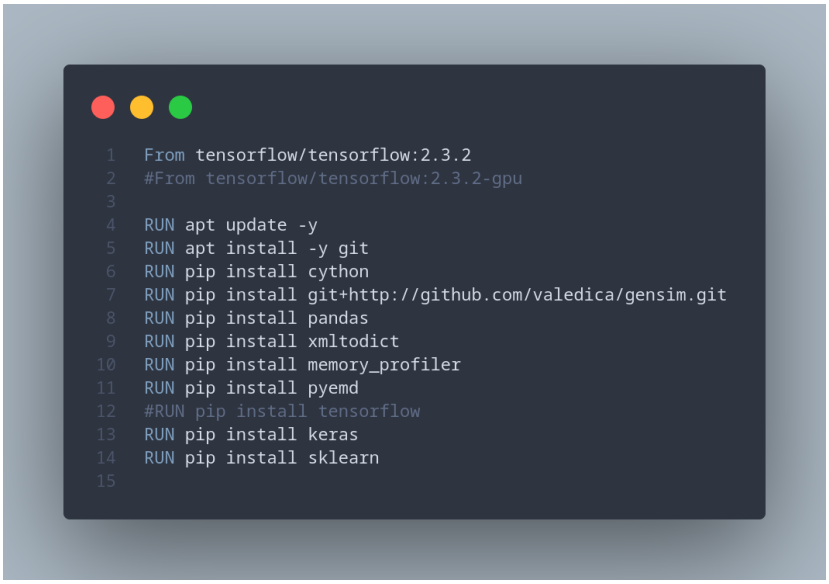


Figura 6.14: UMLda classe Scheduler

Capítulo 7

Experimentos e Resultados

Un dos puntos chave deste proxecto é a experimentación e a avaliación dos resultados de cada unha das versións do pipeline. Para realizar os experimentos dispúxose de varias máquinas. Algunhas con GPUs máis potentes e outras con CPUs máis potentes. Ademáis, houbo que compartir estas máquinas con varios investigadores. Gracias a Docker cada persoa puido illar as versións das librerías e a execución dos seus experimentos da dos demáis.



```
From tensorflow/tensorflow:2.3.2
#From tensorflow/tensorflow:2.3.2-gpu

RUN apt update -y
RUN apt install -y git
RUN pip install cython
RUN pip install git+http://github.com/valedica/gensim.git
RUN pip install pandas
RUN pip install xmltodict
RUN pip install memory_profiler
RUN pip install pyemd
#RUN pip install tensorflow
RUN pip install keras
RUN pip install sklearn

```

Figura 7.1: Dockerfile

Na figura 7.1 móstrase unha imaxe dun Dockerfile. Neste arquivo configúranse o que deberá ter instalado a imaxe empregada para arrancar os contenedores dos experimentos. O seguinte comando xera esta imaxe da que estamos a falar.

```
$ sudo docker build -t <nome imaxe> -< Dockerfile
```

Para arrancar un contenedor introducimos o seguinte comando, que demáis monta o volumen */folder1* da máquina *host* na carpeta */folder2* da máquina virtual. O parámetro *–gpus* permite seleccionar as tarxetas gráficas ás que terá acceso a máquina docker.

```
$ docker run -v /folder1:/folder2 --gpus all -it id /bin/bas
```

Cada experimento require dunha configuración determinada do *pipeline*. Durante a implementación fóronse engadindo parámetros de configuración a medida que as necesidades do pipeline requerían máis versatilidade. A continuación mostrarase un exemplo de arquivo final de configuración 7.2.

```
{
  "training_ctx": "/mnt/datasets/",
  "testing_ctx": "/mnt/datasets/",
  "resources_ctx": "/mnt/experiments/manux/early_twec/resources/",
  "training": ["train_2017_pos", "train_2017_neg"],
  "testing": ["test_2017"],
  "training_cached_pos": "training/cached/pos/",
  "training_cached_neg": "training/cached/neg/",
  "training_models_pos": "training/model/positives/",
  "training_models_neg": "training/model/negatives/",
  "training_deltas_pos": "training/deltas/positives/",
  "training_deltas_neg": "training/deltas/negatives/",
  "testing_cached": "testing/cached/",
  "testing_models": "testing/model/",
  "testing_deltas": "testing/deltas/",
  "all_compass": "",
  "results": "results/",
  "classification": "classification.json",
  "eval_file": "test_golden_truth_pd.csv",
  "fine_tuning": "GoogleNews-vectors-negative300.bin",
  "filter_vocab": "NRC-Emotion-Lexicon-Wordlevel-v0.92.txt"
}
```

Figura 7.2: Arquivo de configuración

Para executar un experimentos haberá que seleccionar unha serie de *goals*, un tamaño de *chunks* para o conxunto de *training* e para o conxunto de *testing* e un arquivo de configuración. A maiores poderase seleccionar se se executará en paralelo, se realizara filtrado, se se finetuneará e qué versión do classificador empregará. A continuación mostrase o comando de exemplo para executar o pipeline.

```
$ python main.py -g read write model_delta classify evaluate
-ch_tra 100 -ch_tes 25 -p -f -t -v 2 -c config.json
```

7.1 Versións e comparativa intraproxecto

Nesta sección farase unha comparativa entre as versións dentro do proxecto, vendo a evolución das diferentes métricas. explicaranse os diferentes experimentos realizados, e a evolución das métricas ao longo das diferentes versións do pipeline.

7.1.1 Versión v1.0

Coma xa se explicou anteriormente no sprint un implenentáronse 2 aliñadores para a fase de *training*, un para positivos e outro para negativos. Logo empregouse o aliñador de negativos para entrenar os modelos temporáis dos usuarios de test. Como xa se comentou na sección 1.2 non se puido xerar un clasificador desta aproximación. Chamaremos a esta versión como a versión **v1.0** e seguiremos esta nomenclatura para cada unha das seguintes versións.

7.1.2 Versión v1.1

Nesta versión decidiuse xuntár os textos nun único aliñador. Desta forma puidemos obter os primeiros resultados do pipeline. A hora de escoller o tamaño dos chunks optamos por empregar 25 textos e 100 textos para o conxunto de training e 25 textos para o tamaño de *testing*. Chamaremos o experimento con 100 textos o **v1.1a** e o experimento con 25 textos o **v1.1b**. Os resultados podense ver na seguinte tabla.

	$ERDE_5$	$ERDE_{50}$	F1	P	R
v1.1a	15.08 %	**11.00** %	**0.217**	**0.123**	**0.917**
v1.1b	**12.48** %	12.47%	0.041	0.027	0.083

Táboa 7.1: Resultados versión v1.1

Como pódese ver o resultado non é moi bo. No caso do experimento **v1.1a** obtense un *recall* moi alto debido a que se clasificaron todos os usuarios como positivos. No caso do clasificador **v1.1b** acontece que o clasificador erra case todas as predicións.

7.1.3 Versión v1.2

Nesta versión faise un afinamento sobre un modelo preentrenado para empregar como *compass*, espérase que se poida aproveitar mellor o vocabulario dos contexto, e que os modelos se comporten dunha forma máis estable. Outra vez faremos dous experimentos cada un deles cun tamaño de *chunk* diferente. O experimento **v1.2a** empregará 100 textos por *chunk* de training e o experimento **v1.2b** empregará 25 textos por *chunk* de *training*.

	$ERDE_5$	$ERDE_{50}$	F1	P	R
v1.2a	**15.37** %	**14.69** %	0.151	0.088	**0.521**
v1.2b	16.49%	16.09 %	**0.175**	**0.103**	0.500

Táboa 7.2: Resultados versión v1.2

Observando os datos vemos, que non houbo melloría dos datos na versión con tamaño de chunk 100, pero sí mellorou un pouco a versión do experimento con tamaño de chunk 25.

7.1.4 Versión v1.3

Nesta versión intentamos reducir a dimensionalidade do problema e escoller características máis relevantes para a clasificación. Esta vez faremos catro experimentos, filtrado e con *fine tuning*, con tamaños de *chunk* 100 **v1.3a** e tamaño de *chunk* 25 **v1.3b** e filtrado e sen *finetuning* con tamaños de *chunk* 100 **v1.3c** e tamaño de *chunk* 25 **v1.3d**. Vemos os resultados na seguinte tabla.

	$ERDE_5$	$ERDE_{50}$	F1	P	R
v1.3a	**17.62**%	**13.72**%	0.182	0.106	0.646
v1.3b	18.58%	16.57%	**0.205**	**0.120**	**0.688**
v1.3c	17.64%	13.75%	0.182	0.106	0.646
v1.3d	18.41%	16.67%	0.199	0.117	0.667

Táboa 7.3: Resultados versión v1.3

Neste caso, podemos apreciar que os resultados dos experimentos con filtrado melloran en todos os casos menos na comparativa da versión sen finetuning e tamaño de chunk 100.

7.1.5 Versión v1.4

Ata agora, agrupábanse os datos de exemplo en positivos e negativos. Nesta versión do experimento agrúpanse os textos dos exemplos por usuario, en vez de por deprimidos e non deprimidos, desta forma farase unha comparativa de máis gano fino.

Chegados a este punto o mellor resultado ata o momento foi o do experimento **v1.4.2b**, o cal se corresponde co experimento no que agrupamos por usuarios, non filtramos o vocabulario, fixemos *fine tuning* e agrupamos *chunks* con tamaño de texto 100. Entendemos que o clasificador de distancia de coseno non é capaz de captar a evolución da linguaxe dos usuarios deprimidos e non deprimidos.

	$ERDE_5$	$ERDE_{50}$	F1	P	R
v1.4.1a	15.08%	11.00%	0.21	0.123	0.917
v1.4.1b	**12.48**%	12.47%	0.041	0.027	0.083
v1.4.2a	14.03%	13.99%	0.173	0.100	0.646
v1.4.2b	12.70%	**10.89** %	**0.233**	**0.133**	**0.958**
v1.4.3a	15.45%	12.72%	0.219	0.125	0.917
v1.4.3b	12.87%	12.87%	0.0	0.0	0.0
v1.4.3c	15.15%	12.42%	0.220	0.125	0.917
v1.4.3d	18.41%	15.04%	0.220	0.127	0.833

Táboa 7.4: Resultados versión v1.4

7.1.6 Versión v2

Nesta versión decidimos cambiar o método de clasificación, sustituindo a distancia de coseno por unha máquina de soporte vectorial. A continuación unha tabla con cada un dos experimentos. Os experimentos son os mesmos que nas versións **v.1**, **v.2** e **v.3** per empregando o novo clasificador. Primeiro mostramos a taboa cos resultados dos experimentos agrupando por positivos e negativos.

	$ERDE_5$	$ERDE_{50}$	F1	P	R
v2.1a	12.18%	11.24%	0.306	0.30	0.312
v2.1b	9.99%	8.58%	0.302	0.179	**0.958**
v2.2a	8.23%	8.20%	0.345	0.294	0.417
v2.2b	5.80%	5.72%	0.367	0.228	0.938
v2.3a	7.24%	7.19%	0.385	0.305	0.521
v2.3b	**6.63**%	**6.56**%	0.331	0.204	0.875
v2.3c	7.03%	6.98%	**0.394**	**0.310**	0.542
v2.3d	6.76%	6.688%	0.326	0.2	0.875
v2.4.1a	7.50%	7.46%	0.436	**0.415**	0.458
v2.4.1b	5.74%	5.67%	0.372	0.235	0.896
v2.4.2a	8.53%	8.50%	0.317	0.264	0.396
v2.4.2b	8.47%	8.38%	0.283	0.165	1.00
v2.4.3a	5.88%	5.88%	**0.456**	0.352	0.646
v2.4.3c	**5.61**%	**5.55**%	0.424	0.299	0.729
v2.4.3d	8.87%	8.78%	0.271	0.157	**0.979**

Táboa 7.5: Resultados versión v2

Analizando os resultados ata o momento vemos que o mellor resultado correspóndese co do experimento **2.4.1a** ou o **2.4.3a** nos cales agrupáronse os exemplos por usuario, cada *chunk* de training tivo 100 textos e non foron finetuneados. O primeiro experimento tivo filtrado mentras que o segundo non.

Estes experimentos remataron coa clasificación mediante SVM. Analizando as tablas pódese ver que xeralmente o SVM clasifica mellor que a distancia de coseno. Tamén podemos ver que, no caso deste clasificador, agrupar por usuario funciona mellor que agrupar por positivos e negativos.

7.1.7 Versión v3

O clasificador SVM mellorou os resultados con respecto ao clasificador por distancia de coseno, pero segue se ter en conta a información temporal. Debido a esto decidimos empregar un modelo baseado en redes neuronáis recurrentes as cales si teñen en conta información temporal. Escollemos as LSTM por funcionar ben captando información a curto e longo prazo. Estas redes requiren ser alimentadas por datos nun formato determinado. Debido a esto so puidemos empregar a versión de datos agrupados por usuario para o entrenamento. A continuación móstranse as táboas cos resultados dos experimentos.

	$ERDE_5$	$ERDE_{50}$	F1	P	R
v3.4.1a	10.36%	9.01%	0.436	0.567	0.354
v3.4.1b	**6.51**%	6.46%	0.500	0.464	0.542
v3.4.2a	10.61%	8.98%	0.447	**0.607**	0.354
v3.4.2b	7.28%	**5.76**%	**0.557**	0.478	**0.667**
v3.4.3a	8.15%	6.36%	0.420	0.316	0.625
v3.4.3b	0.0%	0.0%	0.0	0.0	0.0
v3.4.3c	8.90%	7.26%	0.378	0.267	0.646
v3.4.3d	0.0%	0.0%	0.0	0.0	0.0

Táboa 7.6: Resultados versión v3

Chegados a este punto, observamos que o mellor resultado ata o momento é o obtido no experimento **v3.4.2a** ou no experimento **v3.4.2a** dependendo de se lle damos máis valor a o *precision* ou o *recall*. En xeral pódense observar mellores resultados co clasificador LSTM e parece que o filtrado non está mellorando os resultados.

7.1.8 Versión v4

Nesta versión o obxetivo e coller o mellor clasificador ata o momento e incrementar o tamaño do *training* data. A continuación mostraranse os resultados dos experimentos realizados sobre o LSTM entrenado cos datos de *training* proporcionados polo CLEF eRisk 2017 [2] e os datos de testing proporcionados polo CLEF eRisk 2018 [3].

	$ERDE_5$	$ERDE_{50}$	F1	P	R
v4.4.1a	8.69%	7.45%	0.500	0.523	0.479
v4.4.1b	8.28%	8.25%	0.439	**0.529**	0.375
v4.4.2a	**7.37** %	**5.90**%	**0.561**	0.508	0.625
v4.4.2b	8.01%	6.63%	0.518	0.453	0.604
v4.4.3a	8.54%	7.30%	0.460	0.400	0.542
v4.4.3b	9.09%	7.17%	0.360	0.241	**0.708**

Táboa 7.7: Resultados versión v4

7.1.9 Comparativa intraproxecto

Tras recoller os datos de cada unha das versións, vemos como foron evoluindo os diferentes valores das métricas F_1 $precision$, $recall$, $ERDE_5$ e $ERDE_{50}$. A continución presentamos unhas gráficas que mostran esta evolución.

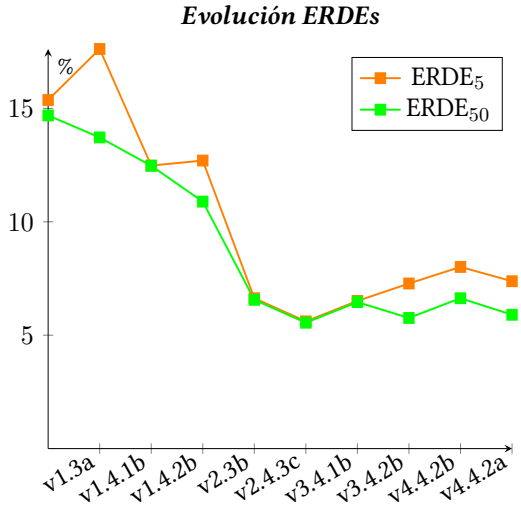


Figura 7.3: Evolución ERDEs

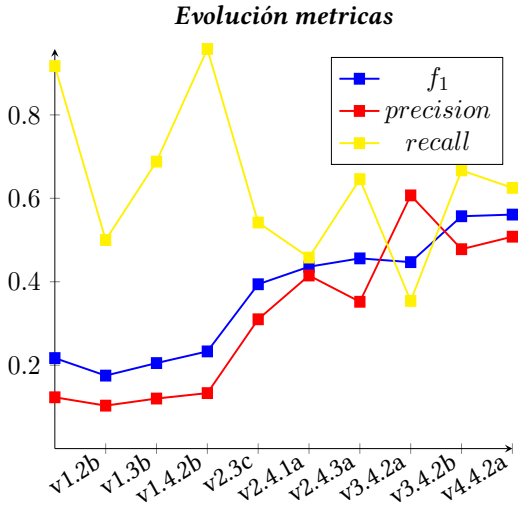


Figura 7.4: Evolución métricas

7.2 Comparativa cos algoritmos presentados a CLEF eRisk 2017

Unha vez temos os resultados da versión que obtivo mellores resultados, faremos unha comparación destes cos algoritmos presentados a o CLEF eRisk 2017 [2] e veremos como de ben ou mal sitúanse as mellores versións do algoritmo. A partir de agora referiremonos a solución como ETWEC.

	$ERDE_5$	$ERDE_{50}$	F1	P	R
GPLA	17.33%	15.83%	0.35	0.22	0.75
GPLB	19.14%	17.15%	0.30	0.18	0.83
GPLB	14.06%	12.14%	0.46	0.42	0.50
GPLB	14.52%	12.78%	0.47	0.39	0.60
FHDOA	12.82%	9.69%	**0.64**	0.61	0.67
FHDOB	12.70%	10.39%	0.55	**0.69**	0.46
FHDOC	13.24%	10.56%	0.56	0.57	0.56
FHDOD	13.04%	10.53%	0.57	0.63	0.52
FHDOE	14.16%	12.42%	0.60	0.51	0.73
UArizonaA	14.62%	12.68%	0.40	0.31	0.58
UArizonaB	13.07%	11.63%	0.30	0.33	0.27
UArizonaC	17.93%	12.74%	0.34	0.21	**0.92**
UArizonaD	14.73%	10.23%	0.45	0.32	0.79
UArizonaE	14.93%	12.01%	0.45	0.34	0.63
LyRA	15.65%	15.15%	0.14	0.11	0.19
LyRB	16.75%	15.76%	0.16	0.11	0.29
LyRC	16.14%	15.51%	0.16	0.12	0.25
LyRD	14.97%	14.47%	0.15	0.13	0.17
LyRE	13.74%	13.74%	0.08	0.11	0.06
UNSLA	13.66%	**9.68**%	0.59	0.48	0.79
UQAMA	14.03%	12.29%	0.53	0.48	0.60
UQAMB	13.78%	12.78%	0.48	0.49	0.46
UQAMC	13.58%	12.83%	0.42	0.50	0.37
UQAMD	13.23%	11.98%	0.38	0.64	0.27
UQAME	13.68%	12.68%	0.39	0.45	0.35
CHEPEA	14.75%	12.26%	0.48	0.38	0.65
CHEPEB	14.78%	12.29%	0.47	0.37	0.63
CHEPEC	14.81%	12.57%	0.46	0.37	0.63
CHEPED	14.81%	12.57%	0.45	0.36	0.62
NLPISA	15.59%	15.59%	0.15	0.12	0.21

Táboa 7.8: Resultados do eRisk 2017

Como se pode ver a solución ETWEC quedaría ben situada con respecto o CLEF eRisk 2017 [2]. A solución que nos supera con máis holgura é a de FHDO que ten moi bos resultados en F_1, $precision$, $recall$. Dase a casualidade, que esta solución tamén incorporou redes

neuronais recurrentes nas suas solucións, xunto con outras técnicas como LSA, BagofWords ou Paragraph Vectors. A solución UNSL tamén nos supera se empregamos o F_1 como medida comparativa, pero pola contra non lograron chegar o 0.5 de precision.

7.3 Comparativa cos algoritmos presentados a CLEF eRisk 2018

A continuación preséntanse os datos dos resultados do CLEF eRisk 2018 [3]. Tamén compararase ETWEC con estas solucións para tratar de decidir como de boa ou mala é a solución.

	$ERDE_5$	$ERDE_{50}$	F1	P	R
FHDO-BCSGA	9.21%	6.68%	0.61	0.56	0.67
FHDO-BCSGB	9.50%	**6.44%**	**0.64**	0.64	0.65
FHDO-BCSGC	9.58%	6.96%	0.51	0.42	0.66
FHDO-BCSGD	9.46%	7.08%	0.54	0.64	0.47
FHDO-BCSGE	9.52%	6.49%	0.53	0.42	0.72
LIIRA	9.46%	7.56%	0.50	0.61	0.42
LIIRB	10.03%	7.09%	0.48	0.38	0.67
LIIRC	10.51%	7.71%	0.42	0.31	0.66
LIIRD	10.52%	7.84%	0.42	0.31	0.66
LIIRE	9.78%	7.91%	0.55	0.66	0.47
LIRMMA	10.66%	9.16%	0.49	0.38	0.68
LIRMMB	11.81%	9.20%	0.36	0.24	0.73
LIRMMC	11.78%	9.02%	0.35	0.23	0.71
LIRMMD	11.32%	8.08%	0.32	0.22	0.52
LIRMME	10.71%	8.38%	0.37	0.29	0.51
PEIMEXA	10.30%	7.22%	0.38	0.28	0.62
PEIMEXB	10.30%	7.61%	0.45	0.37	0.57
PEIMEXC	10.07%	7.35%	0.37	0.29	0.51
PEIMEXD	10.11%	7.70%	0.39	0.35	0.44
PEIMEXE	10.77%	7.32%	0.35	0.25	0.57
RKMVERIA	10.14%	8.68%	0.52	0.49	0.54
RKMVERIB	10.66%	9.07%	0.47	0.37	0.65
RKMVERIC	9.81%	9.08%	0.48	**0.67**	0.38
RKMVERID	9.97%	8.63%	0.58	0.60	0.56
RKMVERIE	9.89%	9.28%	0.21	0.35	0.15
UDCA	10.93%	8.27%	0.26	0.17	0.53
UDCB	15.79%	11.95%	0.18	0.10	**0.95**
UDCC	9.47%	8.65%	0.18	0.13	0.29
UDCD	12.38%	8.54%	0.18	0.11	0.61
UDCE	9.51%	8.70%	0.18	0.13	0.29

Táboa 7.9: Primeira parte dos resultados da tafa un do eRisk 2018

	$ERDE_5$	$ERDE_{50}$	F1	P	R
UNSLA	**8.78**%	7.39%	0.38	0.48	0.32
UNSLB	8.94%	7.24%	0.40	0.35	0.46
UNSLC	8.82%	6.95%	0.43	0.38	0.49
UNSLD	10.68%	7.84%	0.45	0.31	0.85
UNSLE	9.86%	7.60%	0.60	0.53	0.70
UPFA	10.01%	8.28%	0.55	0.56	0.54
UPFB	10.71%	8.60%	0.48	0.37	0.70
UPFC	10.26%	9.16%	0.53	0.48	0.61
UPFD	10.16%	9.79%	0.42	0.42	0.42
UQAMA	10.04%	7.85%	0.42	0.32	0.62
TBSA	10.81%	9.22%	0.37	0.29	0.52
TUA1A	10.19%	9.70%	0.29	0.31	0.27
TUA1B	10.40%	9.54%	0.27	0.25	0.28
TUA1C	10.86%	9.51%	0.47	0.35	0.71
TUA1D	-	-	0.00	0.00	0.00

Táboa 7.10: Segunda parte dos resultados da tafa un do eRisk 2018

Nesta comparativa vemos que ETWEC queda un pouco peor se a comparamos empregando a métrica F_1 como referencia. Esta vez supéranos ata 3 participantes FHDO-BCSG, RKMVERID e UNSLE. A pesar do cal segue a ser unha das mellores solucións. Os que obtiveron mellores resultados collendo como referencia o F_1 foron FHDO-BCSG os cales presentaron varios modelos os que xuntos combinaron varias predicción base para decidir. Estes modelos empregan datos lingüisticos a nivel de usuario, *Bag of Words*, Embeddings de palabras mediante redes neuronais e redes neuronáis comvolucionáis.

7.4 Análise de resultados

Analizando os resultados vemos que a solución presentada foi capaz de superar a maioría dos algoritmos presentados o CLEF eRisk 2017 [2] e CLEF eRisk 2018 [3] quedando moi perto dos gañadores en ambos casos. A tarefa non é doada, isto evidénciase ó ver os resultados dos diferentes eRisk, nos que ninguén foi quen de superar o 0.64 de F_1. Un factor de dificultade moi notable é o feito de que os datos preséntanse con etiquetas a nivel de usuario e non de frase. Por este motivo introdúcese moito ruido no sistema. Vendo os resultados en perspectiva podemos decir que o filtrado do vocabulario tivo unha influencia discreta na mellora dos resultados, pero isto puido ser debido a unha mala selección das palabras a filtrar. Por outra banda a técnica de *fine tuning* resultou beneficiosa, sobre todo cando se aplicou xunto cos clasificadores LSTM e SVM. O tamaño de *chunk* 100 para o conxunto de training funcionou mellor que tamaño 25 cando se aplicou sen filtrado e sen *fine tuning* xunto cos clasificadores de coseno e o SVM. A agrupación por usuario tamén supuxo unha mellora nos resultados

acadados na maioría dos experimentos, tanto no clasificador por distancia de coseno como o SVM. Igualmente o incremento dos datos de entrenamento parece mellorar os resultados nalgúns casos. O algoritmo ETWEC logra posicionarse como unha das mellores solucións, TWEC demostra ser capaz de captar a evolución da linguaxe dos distintos usuarios e xunto co resto dos algoritmos que conforman o *pipeline* permiten, en certa medida, captar os patróns que caracterizan a evolución da linguaxe en persoas deprimidas e non deprimidas.

Capítulo 8

Conclusións e Traballo Futuro

Finalizado o proxecto, podemos decir que os obxetivos expostos na sección 1.2 quedaron cubertos. O proxecto puido finalizar con éxito e dentro dos plazos de entrega grazas a realización dunha correcta planificación.

- Durante a realización do proxecto probouse a utilidade do algoritmo TWEC para a detección temperá da depresión.
- Explorouse a combinación deste algoritmo con outros algoritmos intelixentes e cuantificouse o rendemento de cada unha das versións xeradas despóis de cada sprint.
- Ademáis disto puidose optimizar todo o proceso de execución da ferramenta para reducir ó mínimo o uso de memoria e tempo de CPU/GPU.

Ademáis da investigación plasmada neste documento, o proxecto ten como saida o *pipeline* desenrolado. A cal é unha ferramenta versatil capaz de adaptarse a novos formatos de datos, etapas de procesado e algoritmos intelixentes que se queiran incluir o *pipeline* da ferramenta. Grazas a esta versatilidade, o sistema permite continuar co traballo de investigación. Aproveitando o coñecemento adquirido exploraríanse as seguinte liñas de investigación:

- Implementar técnicas de *feature selection* para mellorar o filtrado do vocabulario
- Explorar outros algoritmos intelixentes e a combinación de varios deles para o clasificador.
- Explorar outros modelos de representación da linguaxe. Como *doc2vec* ou *bert* sustituindo a *TWEC* no *pipeline*.

Apéndices

Bibliografía

[1] OMS, "Depresión." [Online]. Available: https://www.who.int/es/news-room/fact-sheets/detail/depression/

[2] D. E. Losada, F. Crestani, and J. Parapar, "erisk 2017: Clef lab on early risk prediction on the internet: Experimental foundations," in *Experimental IR Meets Multilinguality, Multimodality, and Interaction*, G. J. Jones, S. Lawless, J. Gonzalo, L. Kelly, L. Goeuriot, T. Mandl, L. Cappellato, and N. Ferro, Eds. Cham: Springer International Publishing, 2017, pp. 346–360.

[3] ——, "Overview of erisk: Early risk prediction on the internet," in *Experimental IR Meets Multilinguality, Multimodality, and Interaction*, P. Bellot, C. Trabelsi, J. Mothe, F. Murtagh, J. Y. Nie, L. Soulier, E. SanJuan, L. Cappellato, and N. Ferro, Eds. Cham: Springer International Publishing, 2018, pp. 343–361.

[4] K. Tølbøll, "Linguistic features in depression: A meta-analysis," 12 2019.

[5] F. B. Valerio Di Carlo and M. Palmonari, "Training temporal word embeddings with a compass," *AAAI Technical Track*, 2019. [Online]. Available: https://arxiv.org/pdf/1906.02376.pdf/

[6] Wecare-u, *DEPRESIÓN Y SUICIDIO 2020 Documento estratégico para la promoción de la Salud Mental.* Wecare-u. Healthcare Communication Group, 2020. [En liña]. Dispoñible en: https://sepb.es/webnew/wp-content/uploads/2020/09/LibroBlancoDepresionySuicidio2020.pdf/

[7] D. M. e. T. S. W. Bruce Croft, *Search Engines: Information Retrieval in Practice*, Pearson, Ed. Pearson, 2010.

[8] C. H. e. H. H. Hobson Lane, *Natural Language Processing in Action*, Manning, Ed. Manning, 2019.

[9] V. Boullosa, "Guia salarial sector ti galicia 2015-2016." [En liña]. Dispoñible en: https://es.scribd.com/document/288511179/Guia-Salarial-Sector-TI-Galicia-2015-2016

[10] N. S. H. To'Meisha Edwards, "A meta-analysis of correlations between depression and first person singular pronoun use," 2017.

[11] G. C. Tomas Mikolov, Kai Chen and J. Dean, "Efficient estimation of word representations in vector space," *Cornell University, Computation and Language*, 2013. [En liña]. Dispoñible en: https://arxiv.org/pdf/1301.3781.pdf

[12] F. Rosenblatt, "The perceptron: a probabilistic model for information storage and organization in the brain." *Psychological review*, vol. 65, no. 6, p. 386, 1958.

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

[14] S. M. Mohammad and P. D. Turney, "Crowdsourcing a word–emotion association lexicon," *Computational intelligence*, vol. 29, no. 3, pp. 436–465, 2013.

[15] H. Takeuchi and I. Nonaka, "The new new product development game," *Harvard business review*, vol. 64, no. 1, pp. 137–146, 1986.