UNIVERSIDAD DE CHILE
FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS
DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN

# DISEÑO Y DESARROLLO DE MODELO PREDICTIVO DE RE-HOSPITALIZACIONES NO PLANIFICADAS USANDO DEEP LEARNING

MEMORIA PARA OPTAR AL TÍTULO DE
INGENIERA CIVIL EN COMPUTACIÓN

CONSTANZA CATALINA FIERRO MELLA

PROFESOR GUÍA:
JORGE PÉREZ ROJAS

MIEMBROS DE LA COMISIÓN:
CLAUDIO GUTIÉRREZ GALLARDO
JOCELYN DUNSTAN ESCUDERO

SANTIAGO DE CHILE
2019

RESUMEN DE LA MEMORIA PARA OPTAR
AL TÍTULO DE INGENIERA CIVIL EN COMPUTACIÓN
POR: CONSTANZA CATALINA FIERRO MELLA
FECHA: 2019
PROF. GUÍA: JORGE PÉREZ ROJAS

# DISEÑO Y DESARROLLO DE MODELO PREDICTIVO DE RE-HOSPITALIZACIONES NO PLANIFICADAS USANDO DEEP LEARNING

La tasa de re-hospitalizaciones no planificadas que tiene un centro médico, es utilizada como una métrica de calidad para comparar el nivel de atención de los hospitales a nivel internacional. Además en muchas ocasiones, las readmisiones de los pacientes son a causa de una complicación de la afección anterior, que con un tratamiento distinto podría haber sido evitada. Preveer esta complicación podría haber evitado también el gasto asociado a la re-hospitalización. Las readmisiones podrían prevenirse estableciendo consultas de seguimiento, llamadas por teléfono y educando mejor al paciente. Realizar esto con cada persona conllevaría un gasto muy grande, y por ello sería conveniente contar con alguna predicción de probabilidad que permitiese concentrar los recursos en los pacientes más riesgosos. En el último tiempo han sido propuestos distintos modelos que predicen probabilidad de re-hospitalización al momento del alta del paciente, pero todos han sido pensados y desarrollados para centros en Estados Unidos, Australia o países de Europa. Dentro de los modelos propuestos los que obtienen mejores resultados son los que utilizan redes neuronales profundas. Ninguno de estos modelos es directamente aplicable a la realidad chilena, pues utilizan códigos médicos que en Chile no son almacenados, se basan fuertemente en el texto libre en inglés, o asumen transformaciones a formatos que no son posibles de obtener directamente con los datos disponibles.

Este trabajo de título presenta el diseño y desarrollo de un modelo predictivo de probabilidad de re-hospitalización, basado en redes neuronales profundas, utilizando datos de un centro hospitalario chileno. En cooperación con la Clínica Las Condes, obtuvimos acceso a historiales médicos anonimizados de casi un millón y medio de pacientes, a lo largo de más de 8 años. Inspirándonos en algunos modelos ya existentes, propusimos una arquitectura de tipo deep learning que permite utilizar todo tipo de datos del historial médico: notas clínicas, códigos de diagnóstico, formularios, órdenes de laboratorios y de medicamentos, etc. Además consideramos las particularidades de cómo se almacena información de un paciente en Chile en el historial médico.

Probamos el modelo utilizando los datos históricos de los pacientes de la Clínica Las Condes, y obtuvimos un AUROC de 0.76, que es comparable a los mejores modelos que han sido presentados hasta la fecha (AUROC 0.77 y AUROC 0.76 para hospitales en Estados Unidos). A pesar de que los resultados son comparables a los obtenidos en otras publicaciones, hay que entender que esta es la primera prueba que se realiza con datos chilenos, y hasta donde sabemos, con datos en español. Si bien nuestros resultados son auspiciosos, aún faltan pruebas por realizar y modificaciones potenciales al modelo, para lograr obtener resultados que efectivamente se puedan utilizar en producción.

A mi familia y amigos.

*Porque no van a leer más que esta dedicatoria*

# Tabla de Contenido

# Índice de Tablas

# Índice de Ilustraciones

# Capítulo 1

# Introducción, motivación y objetivos

## 1.1. Introducción

La última década, tuvo una explosión en la cantidad de información clínica guardada digitalmente en forma de historiales de salud electrónicos (EHR por sus siglas en inglés). Aunque el objetivo inicial era almacenar los datos de un paciente para un mejor manejo administrativo del centro de salud, los datos han comenzado hace varios años a ser utilizados por investigadores para el desarrollo de modelos analíticos. Los EHR se han empleado para abordar diversas tareas del área de la salud, por ejemplo, el diagnóstico de enfermedades, la predicción de readmisión hospitalaria, la estimación de la duración de una hospitalización y el pronóstico de mortalidad intra-hospitalaria [14, 44].

Dentro de las problemáticas que han sido abordadas, la re-hospitalización de un paciente en una ventana de tiempo específica es un indicador ampliamente utilizado para comparar la calidad de diferentes centros de salud. Se entiende como "reingreso hospitalario" (re-hospitalización), la admisión no programada de un paciente luego de haber sido dado de alta en alguna fecha anterior (normalmente 30 días), pudiendo ser este reingreso a causa de la misma, u otra enfermedad [3].

Evitar la readmisión de un paciente es importante, pues la detección temprana de algún problema refleja una atención de calidad, permite reducir costos optimizando los recursos ambulatorios y, en el caso de hospitales públicos en Chile, permite aumentar la escasa disponibilidad de camas. Los costos asociados a una re-hospitalización son altos, pues es común que el paciente regrese producto de una complicación de la afección anterior, lo que implica estadías mas extensas, y uso de recursos en mayor cantidad y complejidad [23]. De hecho, un estudio conducido por el "Medicare Payment Advisory Committee (MedPAC)" [12] en 2007, reportó que el 17.6 % de las admisiones hospitalarias tuvieron una readmisión en menos de 30 días luego del alta médica. Entre ellas, un 76 % era evitable. En total, estas re-hospitalizaciones tuvieron un costo de 15 mil millones de dólares [18]. En el año 2012 en Estados Unidos, se estableció un programa que multa a los centros de salud que tienen excesivas re-hospitalizaciones en una ventana de 30 días [8], incitando así a los centros clínicos a subir los estándares de calidad y evitar estas readmisiones.

Un centro de salud puede reducir la cantidad de re-hospitalizaciones haciendo visitas de seguimiento, llamadas por teléfono y educando mejor al paciente. Sin embargo, realizar este seguimiento individualizado para todos los pacientes puede tener costos muy elevados, por lo que sería de gran utilidad poder focalizar estos recursos en grupos específicos de mayor riesgo. Por esto, creemos que es de alto impacto tener modelos que predigan la probabilidad de que un paciente sea reingresado, pudiendo así concentrar el seguimiento, y evitar los costos de una hospitalización o disminuir su extensión.

Desarrollar modelos predictivos a partir de los historiales medicos de los pacientes es una tarea compleja, pues los datos obtenidos de los EHR son particularmente complicados. Esto es debido en primer lugar a su naturaleza heterogenea pues contienen datos (1) numéricos (peso, edad), (2) datos de fechas (nacimiento, ingreso hospitalario), (3) datos categóricos (diagnósticos, procedimientos), (4) texto en lenguaje natural (notas, resumen de alta hostpitalaria) y (5) series de tiempos (señales vitales) [44]. En segundo lugar, por su dependencia temporal, pues existe una relación temporal entre las visitas pero no entre el conjunto de información que compone un encuentro (una consulta o visita al centro clínico). Por último, se tiene también la irregularidad temporal en que ocurren las visitas, y la densidad de ellas entre distintos pacientes, o incluso distintos períodos del mismo paciente [33].

Hasta aproximadamente el año 2015, solo se habían desarrollado modelos de predicción de readmisión utilizando regresiones logísticas en variables seleccionadas manualmente [18]. La estrategia más utilizada ha sido el uso de modelos lineales de estadística clásica, siendo los más comunes los modelos lineales generalizados (regresión logística, regresión de Cox, etc.) y la utilización de métodos bayesianos y regresiones regularizadas (LASSO y regresión Ridge) [19]. Estos métodos tradicionales de minería de datos y aprendizaje estadístico, típicamente requieren primero realizar un trabajo de extracción y creación de características relevantes (*feature engineering*), que luego serán utilizadas como entrada para el modelo. Este trabajo es intensivo en costo y tiempo, pues se requieren expertos en el área para poder definir atributos más efectivos y representativos que los datos en crudo. En consecuencia, esta estrategia presenta grandes dificultades en el desarrollo de modelos cuando se carece de conocimiento en el área y cuando se tienen datos complejos e incompletos. Además, se pierde la posibilidad de descubrir patrones desconocidos, y no es transferible a otras bases de datos o centros médicos [33, 49].

Por otro lado, métodos de aprendizaje profundo (en adelante *deep learning*), son algoritmos que aprenden múltiples niveles de representación a través de la composición de funciones no lineales [27]. Deep learning ha surgido como la estrategia de aprendizaje de máquinas preferida [27], y la que ha mejorado radicalmente el estado del arte en problemas que van desde visión computacional a reconocimiento de voz y procesamiento de lenguaje natural [19, 27]. Estos sistemas son conocidos por su habilidad de manejar grandes volúmenes de datos relativamente desorganizados y con un gran número de variables [42]. Por esto, deep learning se presenta como una atractiva oportunidad para modelar los complejos datos obtenidos de los EHR [33]. De hecho, desde 2015 comenzaron a aparecer publicaciones que utilizan deep learning para predecir la readmisión no planificada en una ventana fija de tiempo (profundizadas en 2.2).

En particular en Chile, relacionado a re-hospitalizaciones, solo pudimos encontrar el estu-

dio de Valjalo y Zamora [23], que analiza y encuentra variables explicativas en la readmisión hospitalaria, utilizando un modelo de regresión logística. En la revisión que realizan Valjalo y Zamora [23], nombran otros trabajos donde se analizan datos de readmisión hospitalaria en Chile, pero ninguno presenta un modelo predictivo, si no que describen el tipo de pacientes que consultó el servicio de urgencias [29], o identifican factores relacionados a la readmisión en pacientes psiquiátricos [48].

En consecuencia, decidimos diseñar e implementar un modelo predictivo de readmisiones no planificadas en una ventana de 30 días, utilizando técnicas de deep learning y los historiales clínicos de los pacientes de la Clínica Las Condes (incluyendo datos demográficos, generales, registros clínicos en texto libre, diagnósticos, informes, etc.). Pues como se expuso anteriormente, es de interés tener modelos que predigan los pacientes que son más riesgosos a volver a hospitalizarse. Y además, tiene aún mayor motivación en Chile pues no se ha desarrollado ninguno hasta la fecha. Por último, elegimos utilizar deep learning, pues las publicaciones recientes muestran que logra mejores resultados [37, 41]. Nuestros resultados preliminares son muy promisorios, obtuvimos valores AUROC que igualan los del estado del arte en la tarea de predicción de re-hospitalizaciones no planificadas, pues nuestro modelo obtuvo un 0.76 AUROC y el estado del arte a la fecha es de 0.77 y 0.76 AUROC en dos hospitales de Estados Unidos [42].

Este trabajo de título se enmarca dentro del proyecto "Creación de un modelo predictivo de hospitalizaciones de pacientes crónicos mediante Deep Learning", liderado por el profesor asociado de la Universidad de Chile Jorge Pérez, en conjunto con el Dr. Javier Mora, Cirujano Cardiovascular de la Clínica Las Condes y del Hospital Clínico San Borja Arriarán. El proyecto fue acogido como uno de los Proyectos de Investigación de la Dirección Académica de la Clínica Las Condes, lo que permitió que trabajáramos en conjunto con el Departamento de Integración de Sistemas de la Gerencia de Tecnologías de la Información, que facilitó el acceso a los datos y veló por la anonimidad de los historiales médicos.

## 1.2. Motivación

La principal motivación de este trabajo de título es generar nuevo conocimiento de punta en la vanguardia de la investigación en el área de fundamentos de los datos aplicados a predicciones clínicas, y junto con ello, el diseño de posibles nuevas aplicaciones y soluciones adaptadas a la realidad del sistema clínico chileno. En particular, buscamos aplicar en Chile técnicas y metodologías de estado del arte, utilizadas en otros países para la predicción de readmisiones, pues es una métrica relevante de calidad de atención hospitalaria, utilizada en la comparación de centros de salud a nivel mundial.

## 1.3. Objetivos

### 1.3.1. Objetivo General

Desarrollar un sistema de predicción de probabilidad de re-hospitalización en los 30 días posteriores al alta médica, basado en arquitecturas de tipo deep learning, utilizando datos del historial médico de pacientes de la Clínica Las Condes.

### 1.3.2. Objetivos Específicos

1. Obtener los datos y definir cuáles serán los que se van a utilizar.
2. Definir cómo se entregarán los datos a la red neuronal.
3. Diseñar e implementar un modelo de deep learning de predicción de re-hospitalización en una ventana fija de tiempo en el futuro.
4. Generar experimentos de entrenamiento y evaluación de la red neuronal para encontrar los hiperparámetros adecuados.
5. Definir e implementar otro modelo simple para comparar el rendimiento en la predicción.

## 1.4. Alcances

El alcance del presente trabajo, consiste en experimentar con modelos de aprendizaje profundo, la tarea de predicción de re-hospitalizaciones no planificadas. En específico, este experimento corresponda a:

- Definir los datos presentes en los registros de la Clínica Las Condes que pueden ser útiles para la tarea predictiva.
- Limpiar y transformar los datos para ser utilizados por algoritmos de clasificación.
- Definir "Re-hospitalización" no planificada para la realidad chilena.
- Diseñar e implementar una red neuronal profunda que haga uso del texto y de los datos presentes, sin mayor selección de parámetros.
- Desarrollar un modelo simple que sirva de comparación.
- Concluir acerca de las ventajas y desafíos de utilizar modelos de aprendizaje profundo en datos clínicos en Chile.

# Capítulo 2

# Antecedentes

A continuación, en la Sección 2.1 presentamos los conceptos y métodos necesarios para comprender el trabajo de título, y en la Sección 2.2 las publicaciones relevantes encontradas en las cuales se basan las decisiones de diseño tomadas.

Asumimos un conocimiento general de aprendizaje de máquinas y de redes neuronales, pues no se ahondará en todos los detalles. La Sección 2.1 solo pretende entregar una explicación general de los conceptos que utilizamos en este trabajo y en las publicaciones relacionadas. Gran parte de la Sección 2.1 está basada en Goodfellow et al. [20].

## 2.1. Marco teórico

### 2.1.1. Algoritmos de aprendizaje de máquinas

Mitchell [34] planteó la definición siguiente: "Se dice que un programa computacional aprende de una experiencia $E$ con respecto a algún tipo de tareas $T$ y medida de rendimiento $P$, si su rendimiento en las tareas $T$ medido por $P$ mejora con la experiencia $E$".

La tarea $T$ puede corresponder a una clasificación, regresión, traducción, eliminación de ruido, etc. Por otro lado, la medida de rendimiento $P$ sirve para evaluar el modelo y poder comparar si mejora o no. Puede ser simplemente la fracción de ejemplos para los que se produce la salida correcta o incorrecta, o pueden ser funciones más complejas según lo que se quiera medir. Por último la experiencia $E$, en los casos de aprendizaje supervisado, corresponde al conjunto de datos de entrenamiento que contiene los pares de la entrada y la salida esperada de la función que se quiere aprender.

#### Clasificador

Entrenar un clasificador para alguna tarea, es esencialmente aprender una función $f$ que aproxima $f^*$, donde $f^*$ convierte una entrada de datos $x$ en una categoría $y$. Es decir $f^*(x) = y$ y $f(x;\theta) \approx y$, donde $\theta$ son los parámetros que definen a $f$. Aproximar $f$ a $f^*$ corresponde entonces a estimar la probabilidad condicional $P(y|x;\theta)$ para predecir $y$ a partir de $x$.

Por ejemplo, si decidimos modelar $f^*$ con una función lineal, entonces $f(x; w, b) = wx + b$. Con lo cual el entrenamiento consistiría en aprender los parámetros $\theta = \{w, b\}$, tales que $f \approx f^*$.

**Función objetivo**

Para encontrar los mejores parámetros $\theta$ elegimos un estimador $J(\theta)$, o también llamado función objetivo, de costo o de pérdida, que buscaremos maximizar o minimizar dependiendo del caso. El estimador es una función de $\theta$ y de los datos del conjunto de entrenamiento, pues representa qué tanto error tiene el modelo al utilizar unos parámetros en específico. Si minimizamos el estimador, entonces encontraremos los parámetros $\theta^*$ con los que nuestro modelo es lo más cercano posible a la distribución de los ejemplos.

Dentro de las funciones de costo más clásicas tenemos "mínimos cuadrados" y "máxima verosimilitud" que se definen con las ecuaciones 2.1 y 2.2. Donde $\{x_{\mathrm{i}}, y_{\mathrm{i}}\}_{\mathrm{i}=1}^{N}$ es el conjunto de ejemplos de entrenamiento, $\widehat{y} = f(x; \theta)$ es la salida del modelo y $\mathbb{P}$ es la probabilidad que nos entrega el modelo.

$$J(\theta) = \frac{1}{2N} \sum_{\mathrm{i}=1}^{N} (\widehat{y_{\mathrm{i}}} - y_{\mathrm{i}})^2 \tag{2.1}$$

$$J(\theta) = -\sum_{\mathrm{i}=1}^{N} \log(\mathbb{P}(\widehat{y_{\mathrm{i}}} = y_{\mathrm{i}} | x_{\mathrm{i}}; \theta)) \tag{2.2}$$

En la mayoría de los casos, nuestro modelo define una distribución de probabilidad, por lo que utilizamos "máxima verosimilitud". En particular, verosimilitud negativa logarítmica (2.2) por su mayor facilidad de calcular. Se le conoce también como entropía cruzada pues la ecuación es equivalente a calcular la entropía entre la distribución de probabilidad de los ejemplos de entrenamiento y la distribución que predice el modelo ($H(p, q) = -\mathbb{E}_{X \sim p} \log[q(x)]$).

**Gradiente descendente**

El algoritmo de gradiente descendiente nos permite minimizar la función de costo $J(\theta)$ y encontrar los mejores parámetros. Para esto, lo que hace el algoritmo es actualizar iterativamente $\theta$ en la dirección en que más disminuye $J$, para así, luego de los suficientes pasos, alcanzar un mínimo local. La dirección $u$ en que más disminuye $J$ será contraria al gradiente, pues:

$$\min_{u, u^T u = 1} \quad \frac{\partial}{\partial \alpha} f(x + \alpha u)\big|_{\alpha = 0} \tag{2.3}$$

$$= \min_{u, u^T u = 1} \quad u^T \nabla_x f(x) \tag{2.4}$$

$$= \min_{u, u^T u = 1} \quad ||u||_2 ||\nabla_x f(x)||_2 \cos\theta \tag{2.5}$$

donde $\theta$ es el ángulo entre $u$ y el gradiente, por lo que $\theta = \pi$ minimiza la ecuación anterior. Entonces la dirección de $u$ debe ser contraria a la del gradiente para que $\theta = \pi$. Por lo tanto la actualización iterativa que hace el algoritmo es:

$$\theta_t = \theta_{t-1} - \varepsilon \nabla_x f(x) \tag{2.6}$$

Donde $\varepsilon$ es la "tasa de aprendizaje", qué define qué tan rápido queremos movernos sobre la curva.

### Gradiente descendente estocástico

En la práctica es imposible calcular el gradiente de la función de costo $J(\theta)$ asociada a todos los ejemplos, pues en 2.1 y 2.2 vemos que el costo total será lineal en la cantidad de ejemplos del conjunto de entrenamiento. La intuición de hacer gradiente descendente estocástico, es que el gradiente calculado es una expectativa, pues está basado en un estimador. Por lo tanto esta expectativa podría ser aproximada utilizando un conjunto más pequeño de los datos. En específico, tomamos un subconjunto aleatorio de ejemplos (*minibatch*) de un tamaño fijo relativamente pequeño, y calculamos el gradiente de $J(\theta)$ solo usando ese subconjunto, y actualizamos los parámetros en esa dirección. Luego de realizar este paso las suficientes veces, vamos a haber considerado todos los ejemplos, es decir vamos a haber avanzado en la dirección en que minimizamos su costo.

### Métricas de evaluación

Para comparar el rendimiento entre diferentes modelos existen distintas métricas que podemos utilizar. Cuál elijamos dependerá de qué deseamos medir, por ejemplo si queremos penalizar los falsos positivos o si nos interesa comparar qué modelo detecta mejor un evento poco común. Algunas métricas comunes que se basan en la matriz de confusión (Tabla 2.1) son las siguientes:

- **Exactitud** *(accuracy)*: Fracción de predicciones correctas del total de predicciones.
  Exactitud = VP+VN/(VP+FP+VN+FN).
- **Precisión**: Fracción de los ejemplos que fueron predichos como positivos que es correcta.
  Precisión = VP/(VP+FP)
- **Sensibilidad** *(recall)*: Proporción de ejemplos positivos que son efectivamente predichos como tal.
  Sensibilidad = VP/(VP+FN)
- **Especificidad**: Fracción de ejemplos negativos que son efectivamente predichos como tal.
  Especificidad = VN/(VN+FP)

| | Predicción + | Predicción - |
|---|---|---|
| Real + | Verdaderos positivos (VP) | Falsos Negativo (FN) |
| Real - | Falsos Positivos (FP) | Verdadero Negativos (VN) |

Tabla 2.1: Matriz de confusión

**Curva ROC.** Describe cómo varía la sensibilidad frente a la especificidad para un sistema clasificador binario, según se varía el umbral de discriminación. Tal como se ve en la Figura 2.1, en el eje horizontal tenemos la probabilidad o puntaje de pertenecer a la clase positiva, y al mover el umbral de clasificación (el valor $p^*$), los valores de sensibilidad y especificidad cambian, con lo cual se construye la curva ROC que se visualiza a la derecha de la Figura 2.2.

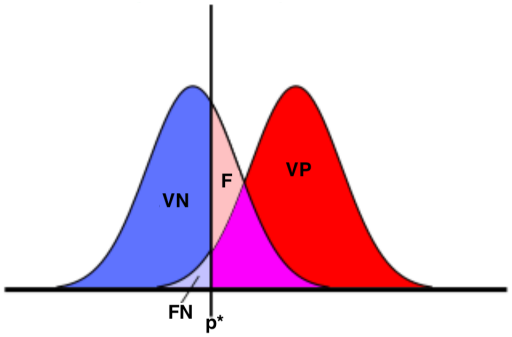



Figura 2.1: Distribución de probabilidad de la señal observada (Wikipedia [2])

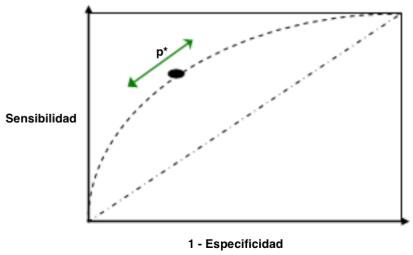



Figura 2.2: Curva ROC (Wikipedia [2])

**AUROC**. Es el área bajo la curva ROC, y sirve como indicador de la capacidad separadora del clasificador, es decir, qué tan bien logra diferenciar entre la clase positiva y la negativa. AUROC = 1 implica que la sensibilidad y especificidad son iguales a 1, y por lo tanto todos los ejemplos positivos y negativos son perfectamente clasificados. Así, buscamos valores AUROC lo más cercanos a 1, y notamos que AUROC=0.5 significa que existe un 50 % de probabilidad de que el modelo clasifique bien, es decir el clasificador no es capaz de distinguir entre las dos clases, y la curva ROC correspondería a la recta punteada de la figura 2.2.

#### Generalización

Los algoritmos de aprendizajes de máquinas son creados con el objetivo de poder predecir de forma correcta en datos no vistos anteriormente. Por lo tanto no buscamos simplemente que el modelo tenga un error bajo en los ejemplos con los que entrenamos, si no que queremos además un error bajo en los ejemplos de prueba también. Esta pérdida la conocemos también como "error de generalización". En consecuencia, buscamos modelos que sean capaces de minimizar el error de entrenamiento y el de generalización, estos dos objetivos pueden llevar a "sobreajuste" o "subajuste". Decimos que un modelo está subajustado si no es capaz de minimizar el error de entrenamiento, y decimos que está sobreajustado si la diferencia entre el error de entrenamiento y el de testing es demasiado grande, es decir el modelo no es capaz de generalizar para nuevos casos.

### 2.1.2. Redes neuronales profundas

Los métodos estadísticos clásicos modelan $f$ como una función lineal y aprenden sus parámetros $\theta$ respectivos. Para extender estos modelos para representar funciones no lineales de $x$, podemos aplicar el modelo lineal sobre una entrada transformada $\phi(x)$, donde $\phi$ es una función no lineal. La estrategia de deep learning es aprender $\phi$ tal que $f(x;\theta,w) = \phi(x;\theta)^T w$, donde $\theta$ son los parámetros utilizados para aprender $\phi$ entre una amplia clase de funciones, y $w$ son los pesos que convierten $\phi(x)$ en la salida deseada.

La definición exacta de $\phi(\cdot)$ depende del modelamiento, pero la unidad básica utilizada para construir redes neuronales es una transformación de la forma $\phi(x) = g(Wx + b)$, donde $g(\cdot)$ es una función no lineal conocida como función de activación, $W \in [h, \mathrm{e}]$ es la matriz de pesos, y $b \in [h, 1]$ es el *bias* que permite trasladar la recta. Esto correspondería a "una capa" de tamaño $h$ de una red neuronal. Si definimos $\phi(x) = g'(W'(g(Wx + b)) + b')$ con

$W' \in [h', h]$ y $b' \in [h', 1]$. Entonces tendríamos una red de dos capas, una de tamaño $h$ y la otra de tamaño $h'$, y así podríamos seguir agregando capas y agregando distintos niveles de abstracción. Estas capas son también conocidas como "capas ocultas" para diferenciarlas de la última capa que es la de salida.

Otra forma clásica en que se definen las redes neuronales es a través de la definición de una "neurona", que siguiendo con la notación anterior sería la operación de una fila de la matriz de pesos $W$, es decir la neurona i-ésima realizaría el cálculo: $g(W_{[\mathrm{i},:]}x + b_{\mathrm{i}})$. Por lo tanto al decir que tenemos una capa de tamaño $h$, nos referimos a que tenemos $h$ neuronas.

### Entrenamiento

A grandes rasgos, entrenar una red es encontrar los parámetros $\theta$ con que se minimiza la función de costo, tal como se describió anteriormente. La principal diferencia es que en el caso de las redes neuronales, calculamos el error y luego lo propagamos desde la capa más externa hasta la primera, es decir, calculamos cuánto aporta cada uno de los pesos o valores de las matrices a ese error. Este cálculo es realizado utilizando la regla de la cadena según las dependencias de la red definida, y es conocido como *backpropagation*. Finalmente, utilizando el algoritmo de gradiente descendiente, se actualizan los valores de las matrices.

### Hiperparámetros

Entendemos como hiperparámetros, todas las elecciones de diseño de la red que pueden variar su rendimiento. Por ejemplo, la cantidad de neuronas en una capa, la función no lineal a usar, la cantidad de capas, los valores de las regularizaciones (2.1.3), etc. Para elegir estos valores simplemente probamos con qué combinación obtenemos mejores resultados. Para elegir qué conjuntos de valores probar, existen distintas técnicas:

- **Búsqueda por cuadrícula**: Entrenaremos y evaluaremos la red con cada una de las combinaciones posibles. Éstas quedan definidas por el producto cartesiano entre todos los conjuntos de valores discretizados de cada hiperparámetro, que son definidos arbitrariamente. Este método tiene la ventaja de que es totalmente paralelizable, pero posee la gran desventaja de que la dimensionalidad de las configuraciones a probar aumenta rápidamente (por la cantidad de variables y sus valores a probar).
- **Búsqueda aleatoria**: Seleccionamos aleatoriamente las combinaciones de valores para cada hiperparámetro y probamos el modelo con cada una de ellas. Esta técnica es totalmente paralelizable, y ha sido demostrado que es más eficiente que la búsqueda por cuadrícula [6]. De todas formas igual que en la búsqueda por cuadrícula, la dimensionalidad de configuraciones a probar crece rápidamente.
- **Optimización bayesiana**: En este algoritmo empleamos un proceso gaussiano para aprender la función de rendimiento del modelo, y utilizamos los resultados de rondas pasadas de experimentación para iniciarlo. Luego, el proceso gaussiano nos indica qué nueva configuración de hiperparámetros deberíamos probar para obtener la máxima nueva información de la función de rendimiento. Así, la búsqueda de hiperparámetros es guiada y podemos minimizar la cantidad de combinaciones que probamos. Pero la desventaja del algoritmo es que no es completamente paralelizable, y el cálculo de la siguiente combinación de hiperparámetros a probar aumenta linealmente con la cantidad de puntos ya probados.

**Neuronas ocultas y de salida**

Los tipos de neuronas quedan definidos por la función de activación que aplican a la combinación lineal. En principio, cualquier tipo de neurona puede ser utilizada tanto en la capa de salida como en la capa oculta, pero según sus propiedades se utilizan más en una que en la otra. A continuación presentamos una descripción general de las funciones softmax, sigmoide, ReLu y tangente hiperbólica.

**Sigmoide** Es común utilizar esta función cuando se quiere tener valores entre 0 y 1, y concentrar los más grandes y los más pequeños. Podemos visualizar la función sigmoide en la Figura 2.3 y está definida por:

$$\sigma(z) = \frac{1}{1 - \exp(-z)} \qquad \sigma(z) \in [0, 1] \tag{2.7}$$

**Tangente hiperbólica** Esta función de activación tiene propiedades similares a la sigmoide, pero su mayor diferencia es que la salida está entre -1 y 1. Podemos observarla en la Figura 2.4 y está definida por:

$$\tanh(z) = \frac{\exp(z) - \exp(-z)}{\exp(z) + \exp(-z)} \qquad tanh(z) \in [-1, 1] \tag{2.8}$$

**ReLu** La función de activación ReLu o unidad rectificada lineal (por sus siglas en inglés), es la elección por defecto recomendada en una neurona. Aplicar esta transformación a la salida de una función lineal, resulta en una transformación no lineal. No obstante, la función se mantiene bastante cercana a ser lineal, en el sentido de que es una función lineal por partes (Figura 2.5). En consecuencia, ReLu conserva varias de las propiedades que hacen a los modelos lineales fáciles de optimizar. Por ejemplo, el hecho de que no se satura para grandes valores de $z$. Está definida por:

$$\text{rect}(z) = \text{máx}(z, 0) \tag{2.9}$$

**Softmax** La función softmax es comúnmente utilizada en la capa de salida de un clasificador para representar una distribución de probabilidad sobre $n$ clases diferentes. En raras ocasiones, puede ser utilizada en una capa oculta si deseamos modelar una elección entre $n$ diferentes opciones de alguna variable interna. Está definida por:

$$\text{softmax}(z)_{\text{i}} = \frac{\exp(z_{\text{i}})}{\sum_j \exp(z_j)} \tag{2.10}$$

### 2.1.3. Regularizaciones

Son modificaciones que hacemos al algoritmo de aprendizaje con el objetivo de reducir el error de generalización, no el error en el conjunto de entrenamiento.

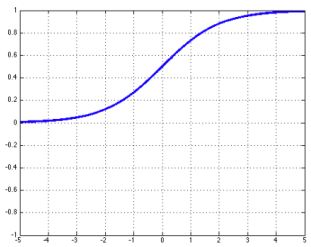


Figura 2.3: Función de activación sigmoide

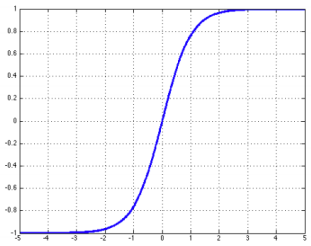


Figura 2.4: Función de activación tangente hiperbólica

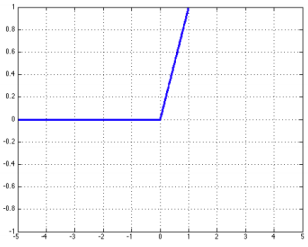


Figura 2.5: Función de activación ReLu

**L2**

La regularización L2 agrega un término en la función de costo para penalizar el uso de pesos muy grandes. También se conoce como *weight decay*, que es el término que se agrega en la actualización de los pesos. Ambas son equivalentes, dado que si se agrega el valor en la función de costo (2.11), luego al derivar este valor para obtener las actualizaciones de los pesos (2.12), se obtiene el *weight decay* $\lambda' = 2\lambda$.

$$L_{\text{total}} = L + \lambda ||w||_2^2 \tag{2.11}$$

$$w_{\text{i}+1} = w_{\text{i}} + \alpha(\lambda' w_{\text{i}} - \langle \frac{\partial L}{\partial w}|_{w_{\text{i}}} \rangle) \tag{2.12}$$

**Dropout**

Dropout [45] es una técnica de regularización bastante simple pero muy efectiva. Esencialmente lo que propone es eliminar neuronas con una probabilidad $(1-p)$ en cada iteración del entrenamiento, o dicho de otro modo mantener las neuronas con una probabilidad $p$. Luego al testear la red, utilizamos todas las neuronas para computar la predicción. El resultado de realizar esto es que el modelo aprende información más valiosa de los datos, es menos propenso a sobreajustarse, y por ende alcanza mejor rendimiento. La intuición detrás de dropout, es que al eliminar aleatoriamente algunas neuronas, estamos entrenando potencialmente un conjunto exponencial de sub redes al mismo tiempo. Luego al predecir con todas las neuronas, estamos promediando las predicciones de cada sub red.

### 2.1.4. Word Embeddings

Los embeddings son vectores numéricos de una dimensión $m$, que buscan representar variables discretas y categóricas de forma continua. De esta forma contienen relaciones de distancia en algún espacio vectorial.

El concepto de embedding viene de Bengio et al. [5], donde definieron una arquitectura de red neuronal para aprender automáticamente embeddings de palabras a partir de un corpus grande de texto. Posteriormente en 2013, Mikolov et al. [31] modificaron la arquitectura para mejorar el tiempo de entrenamiento y pusieron a disposición de la comunidad embeddings pre entrenados.

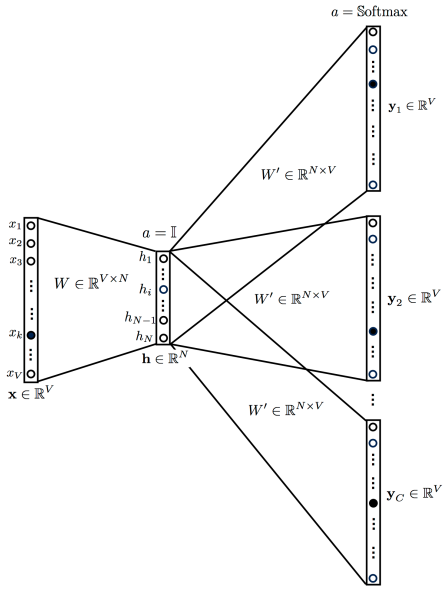



Figura 2.6: Modelo skip-gram para aprender embeddings (Universidad de Stanford [1])

La lógica principal de los embeddings de palabras es aprender una palabra $w$ según su contexto, entendido como las $k$ palabras que están antes y después de $w$. En Mikolov et al. [31] propusieron dos métodos: *skip-gram* (Figura 2.6), donde a partir de una palabra se aprende a predecir su contexto, y *CBOW* en que a partir del contexto (las $2k$ palabras) hacemos que la red aprenda a predecir la palabra central. La arquitectura definida para entrenar estos vectores es bastante simple, de hecho plantearon que no era necesario aplicar una función no lineal en la capa oculta, lo que los hace mucho más rápidos de entrenar. Como observamos en la Figura 2.6, la entrada es un vector "one-hot", que tiene encendida solo la palabra a partir de la cual queremos predecir su contexto. Esto provoca que al ser multiplicado con la matriz de la capa oculta obtengamos solamente una fila, que será justamente el vector que representará esta palabra. Luego a partir de este vector, aplicamos una softmax para obtener las probabilidades de que cada palabra en el vocabulario sea contexto de la inicial. En Mikolov et al. [31] entrenaron vectores de esta forma, los llamaron *word2vec* y los pusieron a disposición de la comunidad.

*CBOW* y *skip-gram* fueron los primeros embeddings masivamente utilizados, y que permitieron mejorar resultados en diversas tareas de procesamiento de lenguaje natural con tan solo incluirlos en la red. Posteriormente, se crearon embeddings de palabras más sofisticados como GloVe [39], y extensiones de word2vec como fasttext [7]. Recientemente, se comenzaron a utilizar embeddings contextuales, inspirados en dos limitaciones de *word2vec*: en primer lugar que forma solamente una representación para los muchos significados que puede tener una palabra, y que no produce representaciones para palabras fuera del vocabulario. En 2018 fue presentado ELMO [40], que modela el texto a nivel de caracteres, lo que le permite formar vectores para palabras fuera del vocabulario. ELMO utiliza una RNN bidireccional 2.1.5, con lo que es capaz de tener en consideración el contexto de la palabra para generar un embedding particular para ese uso. Más tarde, Devlin et al. [15] presentaron BERT, una arquitectura basada en atención [47] que es capaz de considerar todo el contexto (las palabras antes y después de la central) al mismo tiempo, a diferencia de ELMO que utiliza una RNN que considera lo que viene después, y otra RNN para lo que viene antes.

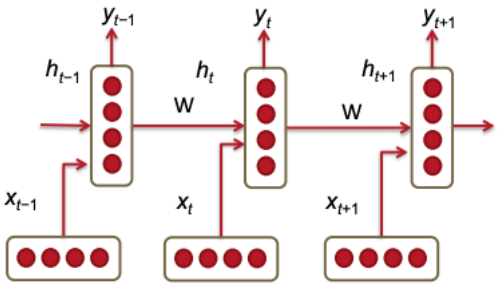



Figura 2.7: Arquitectura red neuronal recurrente (Universidad de Stanford [1])

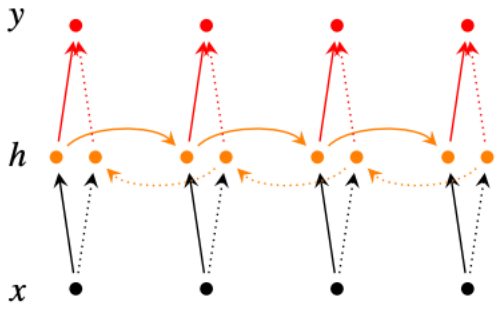



Figura 2.8: RNN bidireccional (Universidad de Stanford [1])

### 2.1.5. Redes neuronales recurrentes

Las redes neuronales recurrentes son un tipo de redes neuronales, pero que su principal particularidad es que admiten entradas de diferente largo. Fueron introducidas para procesar texto, pues las entradas comúnmente son una oraciones que tienes distinto largo de palabras.

La arquitectura de una red neuronal recurrente (RNN) se resume en la Figura 2.7, donde $x_{\text{i}}$ es cada vector de la entrada, por ejemplo los embeddings correspondientes a cada palabra de una oración. Cada rectángulo vertical es una capa oculta del paso $t$, y cada una de ellas realiza una operación matricial con sus dos entradas: $x_t$ y $h_{t-1}$, y luego aplica una función no lineal a la suma de ellas produciendo $h_t$. Finalmente, las neuronas de la capa oculta $t$ pueden ser pasadas por una capa de salida con una función softmax para obtener las probabilidades sobre el conjunto deseado, por ejemplo sobre el vocabulario de posibles palabras siguientes en la oración. Esto se resume en las ecuaciones siguientes:

$$h_t = \sigma(W_{hh}h_{t-1} + W_{hx}x_t + b_1) \tag{2.13}$$
$$y_t = \text{softmax}(W_s h_t + b_2) \tag{2.14}$$

Además, podemos apilar una RNN sobre otra, utilizando la salida $y_t$ como la entrada $x_t$ de otra RNN, creando así RNN de múltiples capas. También podemos definir RNN bidireccional, utilizando una RNN que consume la entrada de izquierda a derecha y otra que lo hace de derecha a izquierda, y la salida de la RNN bidireccional es la concatenación de ambas salidas (Figura 2.8).

Las principales ventajas de las RNN es que pueden procesar secuencias de largo variable, y en teoría pueden utilizar información de cualquier paso anterior. Además, el tamaño del modelo es independiente del tamaño de la entrada. Pero tienen las desventajas de que computarlas es lento pues son secuenciales y por ende los cálculos no pueden ser paralelizados. También, en la práctica no son capaces de acceder a la información de pasos demasiado antiguos, por problemas como explosión o desaparición de gradiente, explicado a continuación.

### Recorte de gradiente

La salida de una RNN es resultado de la reiterada multiplicación de $W_{hh}$ con cada una de las capas ocultas. Al calcular la variación de la función de costo para la salida de la capa i, en relación a alguna capa oculta anterior $j$, obtenemos que esta salida es proporcional a $W_{hh}$, es decir:

$$\frac{\partial J_{\mathrm{i}}(\theta)}{\partial h_j} \propto W_{hh}^{(\mathrm{i}-j)} \tag{2.15}$$

De esta forma, si los pesos de la matriz $W_{hh}$ son mucho menores o mucho mayores que 1 y $(\mathrm{i} - j)$ es lo suficientemente grande, entonces $W_{hh}^{(\mathrm{i}-j)}$ tiende a 0 o $W_{hh}^{(\mathrm{i}-j)}$ tiende a $\infty$ respectivamente. Así, la contribución de $x_j$ al predecir $y_{\mathrm{i}}$ disminuye si el gradiente desaparece muy rápidamente, y no podemos saber si en realidad no existe dependencia entre esas dos entradas o simplemente el modelo no es capaz de aprenderla. Por otro lado, la explosión del gradiente es más fácil de detectar, pues los valores se indeterminarán en el computador (convirtiéndose en NaN por ejemplo).

Mikolov et al. [38] introdujeron una heurística para controlar el problema de la explosión del gradiente conocida como recorte de gradiente (*gradient clipping*). Lo que hace, es simplemente actualizar el gradiente a algún pequeño valor fijo cuando sobrepasa un umbral, controlando así que en ningún momento los valores sean demasiado grandes. Para solucionar el problema de la desaparición del gradiente, podemos utilizar ReLu en vez de sigmoide, pues la derivada de ReLu es 1 o 0, que evita que la multiplicación tienda a 0.

### LSTM

En la práctica es bastante difícil entrenar una RNN para que capture dependencias lejanas en la entrada, por lo que posteriormente aparecieron arquitecturas como GRU [10] y LSTM [21] (Por las siglas en inglés de "Memoria de largo y corto plazo"), que son redes neuronales recurrentes modificadas para que logren capturar esas dependencias más fácilmente.

La unidad básica de una red recurrente de tipo LSTM sigue las siguientes ecuaciones:

$$\mathrm{i}_t = \sigma(W^{(\mathrm{i})}x_t + U^{(\mathrm{i})}h_{t-1}) \qquad \text{Puerta de la entrada} \tag{2.16}$$

$$f_t = \sigma(W^{(f)}x_t + U^{(f)}h_{t-1}) \qquad \text{Puerta de olvido} \tag{2.17}$$

$$o_t = \sigma(W^{(o)}x_t + U^{(o)}h_{t-1}) \qquad \text{Puerta de salida} \tag{2.18}$$

$$\tilde{c}_t = \tanh(W^{(c)}x_t + U^{(c)}h_{t-1}) \qquad \text{Nueva memoria de la celda} \tag{2.19}$$

$$c_t = f_t \times c_{t-1} + \mathrm{i}_t \times \tilde{c}_t \qquad \text{Memoria final de la celda} \tag{2.20}$$

$$h_t = o_t \times \tanh(c_t) \tag{2.21}$$

La unidad de una LSTM en un paso $t$ está compuesta por 3 partes: la memoria de una celda $c_t$, el estado oculto $h_t$ y la entrada $x_t$. Además, posee 3 puertas que controlan cómo utiliza la entrada $x_t$, el estado oculto anterior $h_{t-1}$ y la memoria anterior $c_{t-1}$, para generar su estado oculto $h_t$ y la nueva memoria de celda $c_t$. Estas puertas son:

- Puerta de entrada (2.16): controla qué parte de la nueva entrada $x_t$ es útil preservar.
- Puerta de olvido (2.17): controla cuánto de la memoria de la celda anterior $c_{t-1}$ se debe considerar.

- Puerta de salida (2.18): controla qué partes de la memoria de la celda deberían ser presentadas en el estado oculto $h_t$.

Así, la memoria final de la celda (2.20) toma el consejo de la puerta de olvido $f_t$ y filtra parte de la memoria pasada $c_{t-1}$. De igual forma, toma el consejo de la puerta de entrada y considera sólo una parte de la nueva memoria $\tilde{c}_t$. Finalmente se produce la memoria de la celda sumando ambas partes.

### 2.1.6. Redes neuronales convolucionales

Las redes neuronales convolucionales son un tipo de redes neuronales que su operación básica es la convolución. Recordemos que la convolución en una dimensión, entre $f$ y $g$ evaluada en $n$ es igual a $\sum_{m=-M}^{M} f(n-m)g(m)$, que es equivalente a aplicar el filtro $g$ sobre una ventana de tamaño $2M$. Así, para el caso de dos dimensiones el filtro $g$ que aplicamos es una matriz también 2 dimensional.

Este tipo de redes neuronales puede aplicarse en una oración utilizando el caso 2 dimensional. Pues podemos ver la concatenación de los embeddings de las palabras como la matriz de dos dimensiones de entrada.

#### Pooling

Es un tipo específico de filtro que se suele utilizar en este tipo de redes neuronales, pues permite reducir la dimensionalidad de alguna capa e introduce invarianza en la red. Simplemente lo que hace es tomar el mínimo (min-pooling) o el máximo (max-pooling) de un conjunto de entrada, formalmente: $\hat{c} = \text{máx}(\mathbf{c})$ con $\mathbf{c} = [c_1, ..., c_n]$ y $c \in R^l$, y por lo tanto $\hat{c} \in \mathbf{c}$.

## 2.2. Estado del arte

A continuación realizamos una revisión general de las publicaciones más relevantes que utilizan deep learning para crear representaciones de los EHR, y/o que utilizan técnicas de deep learning para predecir readmisión hospitalaria.

### 2.2.1. Primeras pruebas de RNN en readmisión hospitalaria

Habiendo leído 3 revisiones de aplicaciones de deep learning en EHR [33, 49, 44], la publicación más antigua que encontramos que utiliza historiales médicos para predecir readmisión hospitalaria, fue de Futoma et al. [18] en 2015. Este estudio nace del interés de tener mejores modelos de predicción de re-hospitalización, luego de que en Estados Unidos comenzara la nueva regulación que multa a los centros de salud según su desempeño en métricas de calidad. Futoma et al. realizan una comparación de modelos estadísticos lineales y no lineales, y además una comparación entre el mejor de ellos con uno basado en deep learning. Los modelos estadísticos utilizan una base de datos que contiene más de 3 millones de admisiones, agrupadas en 280 grupos de diagnósticos relacionados. En cambio la comparación con el modelo de deep learning es realizada en 5 grupos de enfermedades de interés, que según indican en la publicación, es un conjunto de menor tamaño dada la dificultad de entrenar este tipo

de modelos. Los clasificadores estadísticos comparados fueron: regresión logística, regresión logística penalizada, otra variación de regresión logística, conjunto de árboles de predicción (*random forests*) y máquinas de vectores de soporte (SVM). Evaluaron los modelos en cada grupo de diagnóstico y en la base de datos completa y usaron el área bajo la curva ROC (AUC) para compararlos, con lo que obtuvieron que la regresión logística penalizada y el conjunto de árboles de predicción eran los mejores predictores de re-hospitalización en 30 días. Luego comparan la regresión logística penalizada a una red neuronal profunda (DNN por sus siglas en inglés) de 3 capas ocultas con función de activación sigmoid y softmax en la final, y teniendo $0{,}75 * N$ nodos en cada capa oculta (donde $N$ es el número de nodos de entrada). Evaluaron estos dos clasificadores en los 5 grupos de enfermedades de interés y en cada uno de ellos la red neuronal profunda tuvo mejor AUC (3/5 veces mejor), concluyendo así que deep learning obtuvo un mejor desempeño global.

### 2.2.2. Med2Vec: Representación vectorial de visitas médicas

En Agosto de 2016 Choi et al. [9] buscando obtener mejores representaciones de los EHR e inspirándose en word2vec [32], definieron una red neuronal de 2 capas que aprende una representación para los códigos (de diagnóstico, tratamientos, etc.) que componen una visita y una representación para las visitas de cada paciente llamada *Med2Vec*. Como se observa en la Figura 2.9, la entrada de la red corresponde a la representación de una visita, formada por el vector con 1's y 0's según si la visita contenía o no el código de esa posición. Luego, eso es pasado por una capa oculta de la red y el resultado es concatenado a otro vector que representa los datos demográficos (edad, sexo y étnia). Por último ese vector es pasado por la segunda capa oculta y finalmente al resultado se le aplica una softmax para predecir los vectores que representan las visitas adyacentes temporalmente (análogamente a lo que ocurre en word2vec donde se predicen las palabras adyacentes en una oración). Así, los vectores que forman la martiz $W_c$ corresponden a las representaciones de cada uno de los códigos, y el resultado de las dos capas ocultas $v_t$ es la representación de la visita. Estos vectores fueron evaluados de diferentes formas. Una de esas fue utilizarlos como entrada para predecir la visita futura (los códigos que componen la siguiente visita) y para predecir si el paciente se encuentra dentro de un grupo de riesgo o no. Para predecir la siguiente visita utilizaron softmax dadas dos visitas consecutivas y para inferir el grupo de riesgo regresión logística dada una visita. Compararon el desempeño de esta representación con otros métodos anteriormente utilizados en una base de datos de 800 mil pacientes, y concluyeron que Med2Vec supera a los anteriores en todas las tareas.

### 2.2.3. DeepCare: Sistema end-to-end basada en una LSTM para predecir resultados médicos

A finales de 2016 fue presentado *DeepCare* [41], que es un arquitectura de aprendizaje basada en deep learning que lee registros médicos y que modela la progresión de enfermedades y predice su futuro. Pham et al. [41] deciden considerar en este modelo los 4 problemas principales que detectan en los EHR: (i) Dependencias de largo plazo, (ii) una admisión, es un conjunto de diagnósticos e intervenciones (procedimientos y medicaciones), (iii) registro episódico en tiempos irregulares, (iv) interacciones confusas entre la progresión de una enfermedad y las intervenciones. La arquitectura que utilizaron se puede entender a grandes rasgos en los siguientes puntos:

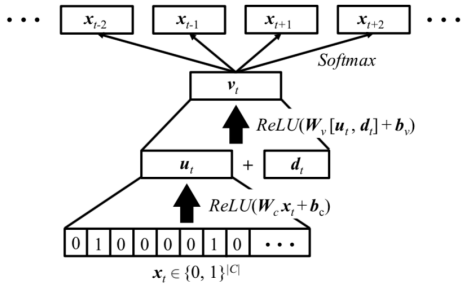



Figura 2.9: Estructura de Med2Vec (Imagen tomada de Choi et al. [9])

1. Para cada uno de los códigos de diagnóstico e intervención, se crean vectores numéricos de tamaño $M$ inicializados de forma aleatoria y aprendidos a través del entrenamiento de la red. Luego, para cada visita se forma la matriz $A \in \mathbb{R}^{M \times |D|}$ que contiene los vectores de cada diagnóstico de la visita concatenados. Análogamente se forma $B \in \mathbb{R}^{M \times |D|}$ para las intervenciones. Luego, se realiza un *pooling* para obtener $x_t \in \mathbb{R}^M$ y $p_t \in \mathbb{R}^M$ que son los vectores representativos de diagnósticos e intervenciones respectivamente. Este *pooling* puede ser de tipo máximo, promedio o suma normalizada.
2. En segundo lugar las secuencias $u_0, .., u_n$ son ingresadas a una variación de una red LSTM para obtener los estados de enfermedad $h_0, .., h_n$. Cada secuencia $u_t$ está formada por $[x_t, p_t, m_t, \Delta t]$ donde $x_t, p_t$ representan los códigos de diagnósticos e intervenciones, $m_t$ indica si la admisión fue planificada ($m_t = 1$) o no ($m_t > 1$), y $\Delta t$ es el tiempo transcurrido entre la admisión actual y la anterior. A continuación enumeramos las variaciones realizadas a la LSTM:

$$\mathrm{i}_t = \frac{1}{m_t}\sigma(W_\mathrm{i}x_t + U_\mathrm{i}h_{t_1} + b_\mathrm{i})$$
$$o_t = \sigma(W_o x_t + U_o h_{t-1} + Pop_t + b_o)$$
$$f_t = \mathrm{d}(\Delta_{t-1:t})\sigma(W_f x_t + U_f h_{t-1} + Q_f q_{\Delta_{t-1:t}} + P_f p_{t-1} + b_f)$$

   La actualización del portal de entrada (i) es ponderado por la inversa de $m_t$ para ponderar en mayor medida las admisiones no planificadas, lo que provoca que se considera en mayor medida la nueva información. Por otro lado, el portal de salida ($o$) que controla los estados de las enfermedades es moderado por la intervención actual ($p_t$). En el portal de olvido ($f$) se agregan las intervenciones anteriores ($p_{t-1}$) para capturar dependencias del pasado. Además, en este mismo portal se agrega el vector $q_{\Delta_{t-1:t}}$ que captura el $\Delta_{t-1}$ de forma paramétrica. Por último el portal de olvido es ponderado por una función monótona decreciente para representar que la influencia de una enfermedad decae en el tiempo.
3. Se realiza un *pooling* de promedio ponderado sobre una ventana determinada, para que de esta forma los estados de enfermedades recientes sean mayormente considerados. Se realiza *pooling* en ventanas de 12 horas, 24 horas y sobre toda la historia. Luego estos tres vectores son concatenados en un solo vector $\bar{h} = [\bar{h}_{12}, \bar{h}_{24}, \bar{h}_{all}]$.
4. Finalmente este vector $\bar{h}$ es pasado como entrada a una red neuronal de una capa oculta con dropout, para realizar la predicción.

*DeepCare* fue evaluado en dos conjuntos de datos, uno de personas con enfermedades mentales que contenía 7191 pacientes, y otro con enfermos de diabetes que contenía 6109

pacientes. En cada uno se predijo, entre otras cosas, readmisiones hospitalarias no planeadas. Utilizaron ventanas de 12 meses para el conjunto de diabetes y 3 meses para el de enfermedades mentales. Algo particularmente interesante de la evaluación que realizaron fue que compararon *DeepCare* con modelos que iban agregando las distintas componentes de su arquitectura y demostraron que con la inclusión de cada una el F-score iba mejorando. En particular, compararon con un conjunto de bosques aleatorios (*random forests*), luego con una RNN con regresión logística, más tarde reemplazaron las unidades de la red por unidades LSTM, posteriormente cambiaron la regresión logística por una red neuronal, y por último agregaron las variaciones a la celda LSTM. Cada uno de estos pasos mejoró la predicción en ambos conjuntos de datos. Concluyendo que *DeepCare* supera a los diversos modelos comparados.

### 2.2.4. Deepr: Sistema end-to-end basado en una red CNN para predecir riesgo futuro

Posteriormente, en enero de 2017 fue presentado *Deepr* [37], que es un sistema que utilizando técnicas de deep learning, modela los historiales clínicos de cada paciente y clasifica sobre eso, tal como lo hacía *DeepCare* [41], pero se diferencian en que esta nueva arquitectura funciona bien en historiales médicos más pequeños (con menos registros). *Deepr* es validado prediciendo readmisiones hospitalarias no planificadas en ventanas de tiempo de 3 y 6 meses. Si bien es evaluado en una ventana de tiempo diferente a la de interés del presente trabajo, la arquitectura de aprendizaje propuesta es relevante. Inspirados en técnicas de procesamiento de lenguaje natural, plantean modelar cada una de las visitas como un conjunto no ordenado de diagnósticos y tratamientos que formarían una oración. Cada oración (i.e. visita) es concatenada a la siguiente con un *token* particular que representa el espacio temporal entre las dos visitas, estos *tokens* fueron definidos como intervalos de tiempo en meses: (0-1], (1-3], (3-6], (6-12] y 12+, que son tratados como palabras únicas. Así, el historial de un paciente no es más que un conjunto de oraciones (códigos de diagnósticos y tratamientos) concatenadas con símbolos especiales (los *tokens* temporales), formando finalmente un documento. Luego, lo que hacen es formar word embeddings a través del método word2vec [32]. Por último, estos vectores son pasados por una capa convolucional que opera sobre una ventana deslizante y produce otros vectores, los que luego son unidos a través de la operación *max pooling* que toma el máximo punto a punto, obteniendo finalmente un único vector que representa el historial médico del paciente. Luego, este vector es utilizado en algún tipo de clasificador para la tarea específica. En este caso Nguyen et al., utilizaron una regresión logística para predecir si un paciente iba a ser re-hospitalizado o no, en la ventana específica definida. Compararon este modelo con una representación del texto utilizando *bag of words* [30] y clasificando con una regresión logística regularizada, concluyendo que la arquitectura de deep learning tenía un mejor desempeño (evaluado utilizando AUC). Además, desarrollan algunas posibilidades de visualización de la red para comprender de mejor forma cómo está clasificando, mostrando que es posible estudiar qué historiales médicos son parecidos entre sí, cuáles diagnósticos están relacionados, y por último desde la salida de la convolución se puede visualizar cuáles son las señales más fuertes que se mantienen.

*Deepr* [37] se diferencia de *Med2Vec* [9] principalmente en que *Deepr* aprende representaciones vectoriales de los códigos de visitas y procedimientos (*words embeddings*) y luego, utilizando una red convolucional que forma una representación de "motivos" (que represen-

tan las señales más fuertes de una ventana fija de *words embeddings*), las que finalmente son unidas con *max pooling*. En cambio, *Med2Vec* aprende vectores numéricos de las visitas utilizando una red neuronal profunda, con la hipótesis que dada una visita debería ser posible inferir las anteriores y las siguientes. Por lo tanto, a pesar de que ambas desarrollan representaciones vectoriales numéricas sobre los historiales médicos, utilizan diferentes arquitecturas para formarlas, y en consecuencia representan diferentes conceptos. Además, en *Deepr* interpretan los motivos más importantes encontrados por la capa convolucional y las representaciones de cada paciente, pero en cambio en *Med2Vec* interpretan las representaciones de las visitas y de los códigos. Por último, *Deepr* toma en cuenta el tiempo entre cada evento mientras que *Med2Vec* no.

### 2.2.5. Google: modelo de deep learning que utiliza la información en los EHR directamente

Por último, la publicación más reciente de Rajkomar et al. [42], presenta una representación de EHR basada en el formato FHIR (por las siglas *Fast Healthcare Interoperability Resources*) que es evaluada en diversas tareas predictivas, tales como: mortalidad interhospitalaria, readmisión no planificada en ventana de 30 días, duración de hospitalización y diagnóstico final de un paciente al momento del alta. El modelo propuesto supera en cada una de las tareas a los preexistentes. Proponen un ensamblaje de tres modelos de deep learning distintos, en específico, utilizaron el promedio de las predicciones de cada uno de los modelos. Usaron datos históricos de dos hospitales distintos, donde cada uno contenía: 9136 (10.7 %) y 15,939 (14.6 %) casos de readmisiones en una ventana de 30 días. A continuación presentamos los detalles de la representación de cada paciente y una perspectiva general de cada uno de los modelos.

En Rajkomar et al. [42] la representación de los datos puede resumirse como sigue:

1. Los datos obtenidos en cada uno de los centro de salud son convertidos según los estándares de FHIR. Este esquema define la representación de alto nivel de datos médicos en "recursos". Los recursos son representaciones de conceptos del mundo sanitario: paciente, médico, problema de salud, observación, etc. Cada recurso puede contener múltiples propiedades, por ejemplo un recurso del tipo medicación podría contener el nombre de mercado, el nombre genérico, ingredientes, etc. Los datos son importados a este esquema sin unificar la terminología particular de cada centro. Si bien este proceso fue manual, es directo pues la mayoría de los campos de las tablas de EHR están relacionados directamente a un tipo de recurso y propiedades en FHIR, y en el caso de que no lo estén se pueden definir extensiones a los recursos según describe el protocolo FHIR.
2. Luego los recursos de FHIR son mapeados a embeddings. Como dijimos anteriormente, cada recurso está compuesto de propiedades, cada una de estas es primero mapeada a un identificador *feature*, y luego cada *token* único dentro de ellas es mapeado a un *token* ID. Por ejemplo, dentro del recurso "medicación" la propiedad "ingrediente" recibe un identificador de *feature*, y cada una de sus instancias (cada ingrediente dentro de ese recurso) recibe un *token* ID único. Luego, cada uno de los *token* ID son mapeados a un embedding de tamaño d definido como hiperparámetro para cada *feature*. Por último, estos embeddings son concatenados en orden temporal. Los *tokens* dentro de una propiedad, pueden ser uno sólo en el caso de elementos singulares, o pueden ser una

secuencia para el caso de texto. Un detalle, es que a las propiedades que corresponden a códigos de sistemas, se les asigna un token único que representa la concatenación del nombre y el código numérico.

3. Por último, cada modelo considera de forma distinta el momento temporal de cada recurso. Pero todos utilizan de alguna forma la diferencia de tiempo en segundos hasta el momento de la predicción, definido como *delta-time*. El momento de la predicción para las readmisiones hospitalarias (que es la tarea que nos interesa), fue al ingreso del paciente, 24 horas después y al momento del alta. Pero la principal predicción, y la mostrada en los resultados, fue al darle el alta, pues es la de mayor interés para los hospitales.

Los *tokens* descritos en el punto 1 son utilizados en cada uno de los 3 modelos, los que luego son ensamblados promediando las probabilidades predichas por cada uno. Los modelos a grandes rasgos fueron los siguientes:

1. **Red neuronal recurrente ponderada**: En esta red, se definen primero secuencias formadas por *tokens* en intervalos de tiempo de 12 horas. Además, para cada token ID se define un "peso". Luego todos los token ID diferentes dentro de un mismo Tipo de Propiedad en el intervalo de 12 horas, son promediados ponderados por ese "peso". El peso junto con los embeddings es aprendido durante el entrenamiento. Además, a cada secuencia se le concatena al final un entero que representa el momento temporal, y que es definido como el logaritmo del promedio redondeado al siguiente entero, dividido por un factor controlado como hiperparámetro. Luego cada una de estas secuencias de embeddings son alimentadas a una red neuronal recurrente con $n$ capas, en específico, a una red LSTM. La capa final es una regresión logística o softmax dependiendo de la tarea y el modelo es optimizado para minimizar la pérdida logarítmica (*log-loss*). Utilizaron diferentes técnicas de regularización, entre ellas dropout [45] a nivel de embeddings, capas, y otros; zoneout [26], L2 [36]. Utilizaron un *batch size* de 128 y optimizaron utilizando Adagrad. Por último, los hiperparámetros fueron encontrados a través de una búsqueda basada en procesos Gaussianos.
2. **Feedforward Model with Time-Aware Attention**: En este modelo se utiliza la concatenación de embeddings definida en el punto 2 de la explicación de la representación de los datos realizada anteriormente. A esta secuencia, le agregaron además un embedding inicial con su asociado *time-delta*($\Delta$) igual a 0. Considerando entonces los embeddings $E_{\mathrm{i}}$ $\mathrm{i} = 0, ..., n$, alimentaron una red neuronal feed-forward con el vector d dimensional $E = \sum_j \beta_j E_j$ y los escalares $log(n+1)$ y $\log(\sum_j \beta_j^2)$, donde

$$\beta_j = \frac{\mathrm{e}^{\alpha_{\mathrm{i}}}}{\sum_j \mathrm{e}^{\alpha_j}}$$
$$\alpha_{\mathrm{i}} = \sum_{j=1}^{k} p_{j,\mathrm{i}} \;\; A_j(\Delta_{\mathrm{i}})$$
$$\vec{p_{\mathrm{i}}} = P \times E_{\mathrm{i}}$$

y por lo tanto $p_{j,\mathrm{i}}$ son los escalares de cada fila $j$ del vector $\vec{p_{\mathrm{i}}}$. Por último las funciones $A_1(\Delta), ..., A_k(\Delta)$ son las que representan el tiempo en la red, y cada una es de alguna

de las siguientes formas (típicamente no se usan todas las formas en un mismo modelo):

$A(\Delta) = 1$
$A(\Delta) = \Delta$
$A(\Delta) = \log(\Delta + 1\text{días})$
$A(\Delta) =$ función por partes con segmentos exponenciales y pendientes aprendidas

Los hiperparámetros utilizados fueron la dimensión de los embeddings $d$ entre 16 y 512, el número de capas ocultas de la red $n$ entre 0-3, y la cantidad de neuronas entre 10-512.

3. **Boosted, embedded time-series model**: Para este modelo utilizaron bi-gramas, tri-gramas y 4-gramas formados a partir de los *tokens* descritos anteriormente en el punto 2 de la representación de los datos. Crearon además 10 reglas binarias que particionan la lista de ejemplos (propiedad, valor, *time-delta*) en dos, utilizando restricciones a la propiedad seleccionada o imponiendo un umbral al valor o al tiempo a considerar. Así, las reales instancias de cada regla, se obtienen seleccionando las entradas (propiedad, umbral para el valor, umbral para el intervalo de tiempo) de forma aleatoria. Luego, cada regla binaria (predicado) se le asocia un peso, para luego pasar la suma ponderada por una capa softmax y obtener la predicción. Con esto, utilizaron el algoritmo de *boosting*, en el que a partir de reglas débiles se puede obtener una regla robusta, para aprender los pesos de cada predicado. Luego de reiteradas rondas, los predicados binarios finales fueron integrados en un vector de dimensión 1024, con el que se alimentó una red neuronal feed-forward con 2 capas ocultas, cada una con 1024 neuronas utilizando la función no lineal ELU [11] (unidad exponencial lineal). En cuanto a regularización, sólo en el entrenamiento, agregaron ruido Gaussiano y aplicaron dropout a la red.

Para comparar el desempeño de este ensamblaje de los 3 modelos anteriormente explicados, Rajkomar et al. [42] implementaron además modelos basados en regresión logística sobre características obtenidas por selección manual, publicadas en otras investigaciones. Los resultados para readmisiones no planificadas en una venta de 30 días, evaluando en el momento del alta, fueron para el caso del ensamblaje de un 0.77 y 0.76 de *accuracy* (exactitud) en cada uno de los hospitales, superando a los otros modelos estadísticos comparados. No obstante, es relevante notar que el mejor de los modelos de estadística clásico obtuvo 0.75 de accuracy en ambos hospitales.

Luego de esta revisión general de algunas de las publicaciones que consideramos más relevantes para este trabajo, ya sea porque predecían la misma tarea o porque presentaban una interesante representación del historial del paciente, concluimos lo siguiente:

- Encontrar manualmente buenas representaciones de los historiales médicos de los pacientes es una tarea compleja que requiere expertos del área.
- Los modelos que representan los EHR utilizando arquitecturas de deep learning, han superado en variadas tareas predictivas a los modelos de estadística clásica que utilizan variables creadas manualmente.

# Capítulo 3

# Metodología para el desarrollo del modelo predictivo

La metodología general del trabajo sigue el proceso KDD (*Descubrimiento de conocimiento en bases de datos* por sus siglas en inglés) propuesto en Fayyad et al. [17], que se refiere al amplio proceso de obtener información de grandes bases desde datos. Podemos visualizar un diagrama de este en la Figura 3.1. En general, corresponde a la acción reiterada de cada uno de los siguientes pasos:

1. **Selección de los datos**: definir el subconjunto de datos desde los cuales se hará la extracción. Esto puede corresponder a seleccionar algunas variables (columnas) y/o algunos ejemplos (filas).
2. **Pre-procesamiento**: eliminación de datos ruidosos o irrelevantes y manejo de datos faltantes.
3. **Transformación**: representar los datos de alguna forma útil para la tarea que se desea realizar.
4. **Minería de datos**: decidir el propósito del modelo: clasificación, regresión, clustering, etc. Elegir el modelo y los parámetros apropiados para encontrar patrones en los datos.
5. **Interpretación y evaluación**: comprender el resultado de la minería de datos y definir alguna métrica o método para determinar qué tan bueno o malo es.
6. **Conocimiento**: generar reportes, tablas o gráficos en los que representar los resultados de la minería de datos.

En las Secciones 3.1 y 3.2 especificamos los detalles correspondientes a los pasos 1 y 2 del proceso KDD, luego en la Sección 3.3 presentamos la transformación realizada en los datos para poder utilizarlos en los clasificadores. Por último, en la Sección 3.4 explicamos los detalles de los modelos predictivos que utilizamos. Los puntos 5 y 6 del proceso KDD los presentamos en el capítulo de Resultados (4). Además, en la Sección 3.5 presentamos cómo formamos los conjuntos de entrenamiento, validación y prueba.

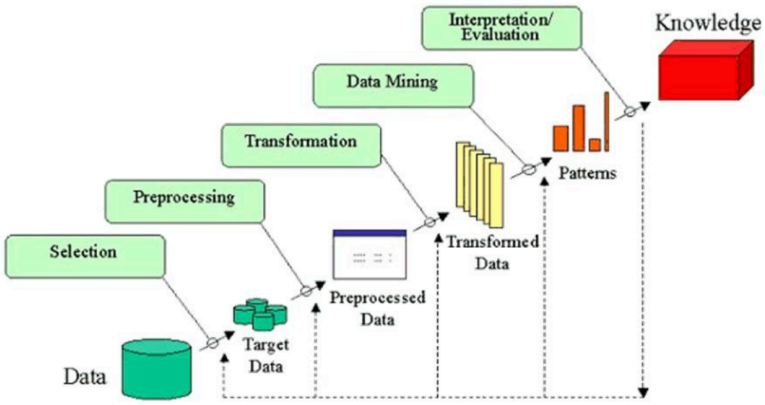



Figura 3.1: Diagrama de proceso KDD (Universidad de Regina, CA [46])

## 3.1. Datos

### 3.1.1. Obtención de los datos

El proceso de obtención de los datos en la Clínica Las Condes comenzó por la presentación del proyecto al comité de ética, que vela porque se cumplan las leyes nacionales y reglas internas de privacidad y ética en general. En esa instancia presentamos el proyecto y sus objetivos, y enfatizamos en la no necesidad de identificar a los pacientes, o de tener datos personales que no estén relacionados a la salud o el cuidado del paciente. El comité planteó la inquietud con respecto a la ley 20.548 "que regula los derechos y deberes que tienen las personas en relación con acciones vinculadas a su atención en salud, reconoce expresamente la reserva de la información contenida en la ficha clínica junto a los otros derechos" [35]. Pero luego se planteó en la reunión que si los datos son anonimizados no se necesita el consentimiento informado de cada persona, pues no es posible identificar a quién pertenecen qué datos.

En consecuencia, el proyecto fue aprobado condicionado a que los datos que nos entregarían estarían anonimizados. Y por ende, el equipo de informática nos dio acceso a vistas de los datos y no directamente a las tablas donde almacenan la información, evitando que pudiéramos identificar a los pacientes o que tuviéramos acceso a sus datos personales como RUT, nombre, dirección, tipo de previsión etc [14].

### 3.1.2. Selección de variables y pre-procesamiento

Utilizamos los datos de historiales clínicos de los pacientes de la Clínica Las Condes entre noviembre de 2009 y diciembre de 2018, que incluyen texto libre de motivo de consulta, diagnósticos, evolución, exámenes y estudios realizados, historial de hospitalizaciones, y variables sociodemográficas de los pacientes.

La Clínica Las Condes ingresa y almacena los historiales médicos de sus pacientes a través de los servicios de Cerner, que le permiten mantener, visualizar y editar las fichas electrónicas. Las bases de datos de Cerner son complejas y extensas, y contienen mucha más de información de la de interés para este proyecto. Por ejemplo, poseen los datos administrativos de la clínica,

como quién administra un medicamento, qué medio de pago utiliza el paciente, quién recibió el dinero, etc. Poseen también datos necesarios para mantener las relaciones entre las distintas tablas de la base de datos. Además, la información está repartida de forma compleja en un gran número de tablas, por ejemplo para obtener los datos demográficos de un paciente es necesario combinar varias de ellas.

Por lo tanto, en una primera etapa trabajamos en entender qué tipo de información existía en la ficha clínica, cómo estaba organizada y cómo era ingresada. Definimos los datos por los que está compuesto el historial médico de un paciente a través de conversaciones con los profesionales de la clínica que mantienen la base de datos, y con las enfermeras y médicos que trabajan en la ficha electrónica. Posteriormente, los técnicos de informática de la clínica pusieron estos datos a nuestra disposición, entregando acceso a vistas en una base de datos relacional que son formadas a partir de consultas a distintas tablas, para así simplificar la información para el uso específico que nosotros le daremos. Los datos recibidos, y que componen el historial médico de un paciente, se enumeran a continuación:

- Datos personales
* Problemas
* Encuentros

A su vez, cada encuentro puede tener asociados:

* Diagnósticos
* Órdenes
* Formularios
* Notas Clínicas

Los * hacen referencia a que pueden haber 0 a $n$ instancias, por ejemplo un encuentro puede tener 0 o varias notas clínicas. En el Apéndice A, describimos de forma detallada cada uno de estos tipos datos que conforman el historial médico de un paciente. Para cada uno además, precisamos los campos que lo forman y cuáles de estos fueron descartados. También especificamos algunos pre-procesamientos realizados para manejar datos incompletos o erróneos.

Avanzada la etapa de pre-procesamiento, nos comentaron que también disponían de los resultados numéricos de los exámenes de laboratorio que se realizan los pacientes en la clínica. Pero, dados los plazos fijos del trabajo de título, no fue posible incluirlos. Esto queda como trabajo propuesto para una segunda iteración del proyecto.

Los datos puestos a nuestra disposición contienen:

- 1.475.224 pacientes.
- 7.955.974 encuentros.
- 6.188.676 de diagnósticos.
- 19.942.118 de órdenes.
- 21.508.633 de formularios, que contienen 21.236 campos distintos, y corresponden a 17GB en texto plano.

- 18.448.316 de notas clínicas, que corresponden a 16,4GB en texto plano.

### 3.1.3. Exclusión de *outliers*

Aparte del pre-procesamiento que realizamos en las variables para completar o corregir datos vacíos o erróneos, solamente eliminamos *outliers*, como explicamos a continuación.

Al visualizar la cantidad de hospitalizaciones por paciente (Figura 3.2), notamos que algunos concentraban una cantidad de admisiones que se escapaba de lo normal. Al consultar con el personal de la clínica, nos comentaron que existen pacientes que exigen que el tratamiento de diálisis que se realizan constantemente, sea ingresado como hospitalización cada vez. Estos casos otorgarían muchos ejemplos positivos, provocando probablemente que el modelo aprendiese a detectar el historial específico de esas personas, y por ende obtendría mejor desempeño. Pero esta mejora sería irreal, pues solo estaría aprendiendo el historial de un par de pacientes, y no generalizaría bien. En consecuencia, decidimos no considerar las personas que contenían más de 100 admisiones.

## 3.2. Selección de ejemplos

La información descrita en la sección 3.1 está en las bases de datos en formas de vistas y tablas. Pero para entrenar algún modelo, buscamos representar la información en pares $\{x_{\mathrm{i}}, y_{\mathrm{i}}\}_{\mathrm{i}=1\ldots n}$, donde $x$ es la entrada al modelo, e $y$ es la etiqueta a predecir. En particular, la predicción de re-hospitalización no planificada es una clasificación binaria: se readmite o no el paciente en los 30 días posteriores al alta. En consecuencia, $x$ corresponde al historial médico completo hasta el momento del alta, e $y$ es igual a 1 si se vuelve a admitir en la ventana de tiempo o 0 si no.

Es importante notar que el historial médico de un paciente puede formar más de un par $\{x, y\}$, pues si es hospitalizado varias veces, entonces cada evento de alta corresponderá a un $x$ y cada uno tendrá su propio $y$ según sea o no readmitido de forma no planificada en los posteriores 30 días. Por ejemplo si una persona se hospitaliza de forma no planificada en $\mathrm{d}_1$ y luego en una fecha posterior $\mathrm{d}_2$, entonces tendríamos un ejemplo que correspondería al historial médico hasta $\mathrm{d}_1$ con su etiqueta 1 o 0, y otro con el historial médico hasta $\mathrm{d}_2$ con su propia etiqueta también.

A continuación, describimos en profundidad el conjunto de ejemplos $\{x, y\}$ que vamos a utilizar para nuestros modelos. En la Sección 3.2.1 detallamos cómo determinamos los valores de $y$ para cada ejemplo. Luego, en la Sección 3.2.2 estipulamos qué hospitalizaciones consideraremos válidas, es decir, todos los $x$ que son válidos.

### 3.2.1. Etiqueta de re-hospitalización no planificada

Como decíamos en la introducción de la sección, $y$, la etiqueta a predecir, será 1 en el caso de que luego del alta el paciente es hospitalizado de forma no planificada dentro de los 30 días posteriores. Nos interesa concentrarnos en admisiones "no planificadas", pues no es de interés predecir admisiones que se espera sucedan, por ejemplo en el caso de que luego

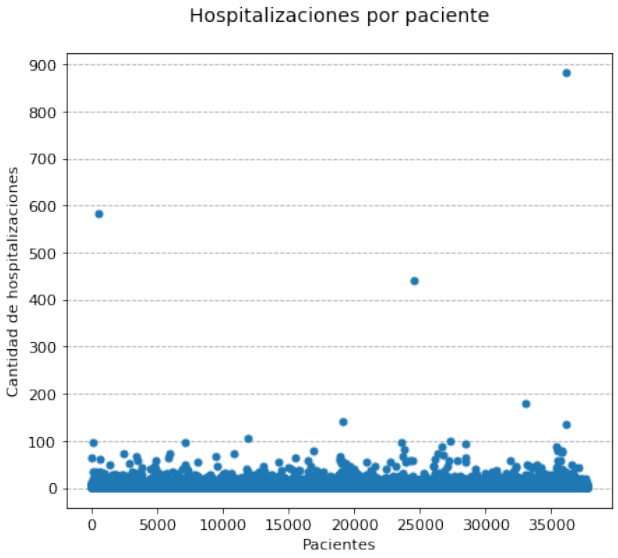



Figura 3.2: Visualización de cantidad de admisiones por paciente

de una cirugía un paciente deba volver a hospitalizarse como parte de su rehabilitación. En estos casos como el de rehabilitación no se busca evitar la readmisión.

A continuación, describimos en primer lugar la definición formal de re-hospitalización no planificada, y planteamos las dificultades de aplicarla en Chile, y en particular en la Clínica Las Condes. En segundo lugar, presentamos la aproximación de la definición de no planificada determinada en conjunto con la Clínica.

### Definición de "no planificada" según CMS

Centros de Servicios de Medicare y Medicaid (CMS por sus siglas en inglés), es una agencia federal estadounidense que administra los programas de cobertura de salud, los estándares para asegurar la portabilidad de la información médica, y los estándares de calidad. En 2011 publicaron la métrica de "Re-hospitalización no planificada para cualquier causa, y cualquier hospital"[3]. Ahí presentan la metodología con la que calculan la métrica para cada hospital, y con ello la definición de re-hospitalización no planificada y el algoritmo para definir el conjunto válido de datos.

CMS en su reporte, define "readmisión" como una hospitalización sucesiva en un centro médico de cuidado intensivo dentro de 30 días luego de la fecha del alta de una admisión elegible (no planificada). Y define también "no planificados", como eventos clínicos agudos que requieren una admisión de urgencia. El algoritmo que plantean para determinar las admisiones planificadas, se basa en 3 principios:

1. Algunos tipos de tratamientos son siempre considerados planificados (cirugías de transplante, quimioterapia, inmunoterapia, rehabilitación);
2. Si no, una re-hospitalización planificada es definida como un ingreso no crítico para un procedimiento programado; y
3. Admisiones por enfermedades críticas o por complicaciones nunca son planificadas.

Luego, el algoritmo determina que una readmisión es planificada si cualquiera de los predicados siguientes se cumple:

1. Se realiza un procedimiento que está en la lista de procedimientos que son siempre planificados independiente del diagnóstico
2. El diagnóstico principal, está en la lista de diagnósticos que son siempre planificados.
3. Se realiza un procedimiento que está en la lista de procedimientos potencialmente planificados, y el diagnóstico principal no está en la lista de diagnósticos de egreso críticos.

Las listas de procedimientos y diagnósticos requeridas para llevar a cabo el algoritmo son entregadas en el reporte en códigos CCS, que son una agregación obtenida a través de AHRQ CCS [13], a partir de los códigos médicos ICD-CM-9.

Las dificultades principales de aplicar directamente la definición de CMS en el caso de la Clínica Las Condes, son:

1. No se tienen registro de los códigos de los procedimientos. Éstos se encuentran descritos en distintos formularios, pero no se ingresa ningún tipo de codificación.
2. El 82,3 % de los diagnósticos está con codificación en SNOMED CT y un 17,7 % en ICD-10-CM, por lo que habría que transformar ambos códigos a ICD-9-CM, pues AHRQ CCS para ICD-10-CM está aún en fase beta. Además, la conversión de SNOMED CT a ICD-9-CM no es completa, hay algunos códigos que no tienen equivalente y sólo encontramos la transformación disponible en una aplicación web.

### Definición de "no planificada" según jefes de proyecto del área de informática y jefa de enfermería de la Clínica Las Condes

En la Clínica Las Condes se programan hospitalizaciones a través de "Pre registros". Se realiza un pre registro cada vez que se quiere programar alguna cirugía o procedimiento con antelación. En general, según explica la jefa de enfermería, existen dos tipos de pre registros:

1. Cuando un paciente necesita un procedimiento de urgencia, pero debe esperar por un pabellón, o si necesita alguna preparación específica que toma tiempo (por ejemplo estar en ayuna)
2. Cuando un paciente programa algún procedimiento, tratamiento o cirugía con antelación.

Los pre registros tienen fecha de término cuando inicia el encuentro, y fecha de inicio cuando se programa el procedimiento. Los jefes de proyecto del área informática en conjunto con la jefa de enfermería, creen y aproximan, que los pre registros que tienen una duración de tiempo menor a 24 horas, corresponden a eventos del tipo 1 recién descrito, y por ende son no planificados.

Para una segunda iteración del proyecto, se podría definir algún conjunto de tipos de hospitalizaciones que considerara las oncológicas y psiquiátricas, que son descartadas por la CMS, para no considerarlas. Pero dado que este conjunto no está determinado, no podíamos descartar estas admisiones directamente, por lo cual no lo hicimos en este trabajo.

### 3.2.2. Criterio de inclusión

Es necesario definir qué hospitalizaciones son válidas para ser utilizadas como $x$ en el modelo, pues en casos donde no depende de la clínica si el paciente se vuelve a re-hospitalizar, no tiene sentido considerarlos como ejemplos negativos o positivos. Basándonos en los criterios de inclusión de la CMS (Centros de Servicios de Mediacare y Medicaid [13]), y siguiendo los utilizados en otras publicaciones que predicen readmisiones no planificadas (como en Huang et al. [22]), decidimos descartar las hospitalizaciones en que se cumple alguna de las siguientes situaciones:

- El paciente fallece durante el encuentro
- El paciente es trasladado a otro centro médico.
- Se le otorga el alta en contra del consentimiento médico.

En Huang et al. [22] eliminan también las admisiones que son de tipo "recién nacido", pero en la Clínica Las Condes ese tipo de encuentros no son etiquetados como hospitalizaciones, si no que directamente con la categoría de recién nacido. Por lo que no fue necesario descartarlos.

## 3.3. Transformación

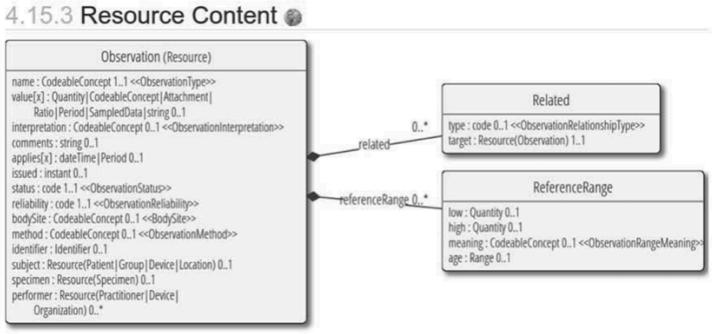
Como hemos dicho anteriormente la entrada del clasificador ($x$), será el historial del paciente hasta el fin de una hospitalización válida. Estos datos corresponden a la información personal, los problemas, y encuentros, que contienen diagnósticos, órdenes, notas clínicas y formularios (ver Apéndice A). En esta sección presentamos en detalle cómo modelamos y transformamos esta información para representarla por un vector numérico que pueda ser utilizado por un clasificador.

Revisando cómo modelan el historial médico de un paciente en otros estudios (Sección 2.2), vemos que la mayoría de las publicaciones lo representan a partir del conjunto de códigos de diagnósticos y procedimientos [9, 41, 37]. En Chile no se exige legalmente registrar los códigos de tratamientos, por lo que en la Clínica Las Condes no están disponibles. Pero tenemos la información de estos procedimientos almacenada en las órdenes y formularios. En consecuencia, buscamos un modelamiento más flexible, que pudiese hacer uso de esta información no estructurada. Así, decidimos seguir la representación presentada en Rajkomar et al. [42], que permite utilizar cualquier tipo de datos a partir de la representación FHIR.

### 3.3.1. FHIR

FHIR [28] es una estructura de datos que fue desarrollada para representar datos clínicos de forma consistente, jerárquica y extensible, para así simplificar el interacmbio de datos entre distintos centros. En rasgos generales, FHIR organiza la información en recursos, que son representaciones de conceptos del mundo sanitario: paciente, médico, problema de salud, observación, etc. Los recursos se basan en estructuras JSON o XML, como se ve en el ejemplo de la Figura 3.3.

Los datos de los centros clínicos utilizados en Rajkomar et al. [42] no estaban almacenados en el estándar FHIR, pero fueron manualmente transformados. Ellos plantean que la con-




*Source*: Resource Content diagram from FHIR specification (http://hl7.org/fhir/observation.html)

```
{
  "resourceType": "Observation",
  "name": {
    "coding": [
      {
        "system": "http://loinc.org",
        "code": "8480-6",
        "display": "Systolic blood pressure"
      }
    ]
  },
  "valueQuantity": {
    "value": 109,
    "units": "mm Hg",
    "system": "http://unitsofmeasure.org",
    "code": "mm[Hg]"
  },
  "appliesDateTime": "1999-07-02",
  "status": "final",
  "subject": {
    "reference": "Patient/example"
  }
}
```

Figura 3.3: Ejemplo de recurso de "Observación" tomado de Mandel et al. [28]

versión es directa de realizar, no obstante, comprender cada uno de los recursos de FHIR y mapearlos a los datos y tablas disponibles en la Clínica Las Condes implicaba invertir tiempo en materias que no eran el objetivo de este trabajo de título. Por lo que decidimos generar una aproximación al protocolo FHIR. Básicamente, consideramos como recursos cada uno de los tipos de datos disponibles: Datos personales, Problemas, Encuentros, Ordenes, Diagnósticos, Formularios y Notas Clínicas. Y definimos sus campos según los datos que componen a cada uno (detalles en Apéndice A).

### 3.3.2. *Features* y *tokens*

Dejando de lado los datos personales, cada uno de los recursos tiene una fecha en que es ingresado al sistema. Dado lo anterior es directo visualizar el historial de un paciente como un conjunto de recursos en una línea de tiempo. Así, podemos definir $x$ como la secuencia de recursos en la línea de tiempo (ver ejemplo en la parte superior de la Figura 3.8), más los datos personales.

La transformación de la lista de recursos a un vector de enteros presentada en Rajkomar et al. [42] se basa en la definición de que un recurso está compuesto por *features*, que corresponden a cada uno de los campos, y cada valor posible del campo es un *token*. Por ejemplo en la Figura 3.8 (en la parte izquierda "FHIR Resource"), tenemos el recurso "medication_order" que contiene el *feature* "medication_order.ingredient.text", del cual observamos los *tokens*

"Piperacillin" y "Tazobactam".

Además, para los campos que son de tipo texto se considera un *token* por cada palabra. Tal como vemos en la Figura 3.8 para el *feature* con ID 5, "procedure.code.text".

### 3.3.3. Conversión de las tablas de datos a recursos FHIR

Los historiales clínicos de los pacientes de la Clínica Las Condes, son almacenados en una base de datos relacional que contiene variadas tablas donde se guarda la información de cada encuentro para cada paciente. Obtener el historial completo de un paciente toma en promedio 10 segundos, pues debemos acceder a varias tablas, algunas veces descomprimir texto, y en algunos casos enviar una cantidad no menor de información. Demorarse al menos 10 segundos en cargar un ejemplo es inviable para entrenar una clasificador. Por lo que decidimos pre procesar los datos de los pacientes y almacenarlos en archivos locales. De todas formas obtener todos los datos de los 117.057 pacientes totales, tomó alrededor de 14 días.

Decidimos guardar el historial de cada paciente de forma ordenada y conveniente para el uso siguiente que le daríamos. Primero, definimos la clase `JsonTimeline` que se encarga de obtener la información de la base de datos y generar los recursos tipo FHIR. Así, dada una persona y una lista de identificadores de encuentros (correspondientes a las hospitalizaciones válidas):

1. Crea un objeto `Persona` que almacenará toda la información que nos interesa.
2. Obtiene la información asociada directamente al paciente y no a un encuentro (notas clínicas, datos demográficos, problemas), y la agrega al objeto `Persona`.
3. Obtiene todos los encuentros asociados a la persona, y los recorre en orden cronológico.
4. Para cada uno de ellos, crea un objeto `Encuentro` y obtiene todos los datos asociados a él (diagnósticos, notas clínicas, órdenes, formularios). Y asocia el encuentro al objeto `Persona`.
5. Además, calcula los índices de la lista total de encuentro que corresponden a los identificadores de encuentros dados.

Finalmente obtenemos un objeto `Persona` que está compuesto por la información asociada al paciente y la lista de `Encuentros`. Luego, cada recurso implementa una función para retornar un diccionario estilo FHIR con todos sus datos. Así, a partir del objeto `Persona` obtenemos recursivamente el historial completo de recursos FHIR. Observamos un ejemplo de este resultado en la Figura 3.4.

Luego, a través de la clase `FeaturesAndTokens` procesamos esta lista de recursos para obtener una lista de pares: (`timestamp`, lista de pares (*feature* ID, *token* ID)) como se observa en el ejemplo de la Figura 3.5. Decidimos almacenar los datos de esta forma pues nos permite tener los recursos ordenados cronológicamente, y además, al guardar directamente el timestamp podemos eventualmente utilizar distintas ventanas o agrupaciones en el tiempo de los datos. Además, permite procesar los distintos *features* de forma diferente si uno así lo quisiera.

Para lograr esta conversión, primero seguimos los tipos estipulados en el Apéndice A

```
[
 ('recurso encuentro',
     {'clase': {'categorical': {'id': 389, 'text': 'Urgencia'}},
      'tipo': {'categorical': {'id': 8913614, 'text': 'Urgencia'}},
      'razon': {'text': 'Golpe PIE DER/1*'},
      'date': {'performed_time': 1344889320.0}}),
 ('recurso orden',
  {'date': {'performed_time': 1344889373.0},
   'unidad medica': {'categorical': {'id': 11737699,
     'text': 'Urg. Escolar'}},
   'tipo': {'categorical': {'id': 10969620,
     'text': 'Ingreso/Traslado/Alta'}},
   'indicacion': {'categorical': {'id': 12664502,
     'text': 'Consulta general de urgencia'}},
   'sinonimo': {'categorical': {'id': 12664505,
    'text': 'Consulta general de urgencia'}},
   'resumen': {'text': 'Consulta general de urgencia '},
   'unidad': {'categorical': {'id': 67234, 'text': 'Discern Expert'}}}),
 ...
 ('recurso formulario',
  {'nombre': {'categorical': {'id': 17927571,
     'text': 'Indicaciones Informe Atención Urgencia'}},
   'date': {'performed_time': 1344892848.0},
   'contenido': [
    {'campo': {'text': 'CASE'},
     'valor': {'text': 'Interconsulta Traumatología'}},
    {'campo': {'text': 'Informe Atención Otras Indicaciones'},
     'valor': {'text': 'Bota de marcha\r\nHielo local\r\nControl en traumatologia en 1 semana'}},
    {'campo': {'text': 'Informe Atención Reposo'},
     'valor': {'text': 'Deportivo'}},
    {'campo': {'text': 'IU Destino al Egreso'},
     'valor': {'text': 'Domicilio con Indicaciones'}},
    {'campo': {'text': 'N° de Días Reposo'},
     'valor': {'text': '15'}}
   ]})
]
```

Figura 3.4: Ejemplos recursos de un paciente en estilo FHIR

para decidir qué campos consideraríamos como categorías y cuáles como texto que debía ser tokenizado. El tokenizador que utilizamos fue desarrollado por Jorge Pérez[1], y le realizamos algunos mínimos cambios:

- Eliminamos tags de la forma "<%tag%>", que fueron generados a partir de la transformación de las notas clínicas en formato rtf a texto plano. Pues el resto de los archivos que fueron transformados a partir del formato HTML, no contenían estos tags.
- Eliminamos espacios extras, manteniendo solamente saltos de línea y espacios simples.
- Los números fueron transformados al token "<num>". El argumento principal de realizar esto, fue que mantener los números tal cual, provocaría que obtendríamos una gran cantidad de tokens únicos pues le otorgaríamos un token ID a cada número distinto. Por otro lado, pre procesar todos los números de los distintos lugares donde tenemos texto (notas clínicas, formularios, resumen órdenes, etc.), para generar buckets o algún tipo de representación que evitara este problema, implicaba trabajo extra que atrasaría los plazos acotados del trabajo de título.

Es importante notar que `FeaturesAndTokens` mantiene un diccionario que transforma los nombres de los *features* a identificadores, y también un diccionario para cada uno de

[1] `https://github.com/jorgeperezrojas/NLPGES/blob/master/models/src/pre_processing_for_train.py`

```
[
 (1344889320.0, [(5, 4), (2, 3), (3, 6), (3, 18), (3, 20), (3, 1)]),
 (1344889373.0, [(6, 10), (8, 19), (7, 20), (9, 5), (9, 13), (9, 2),
                 (9, 18), (12, 6)]),
 ...
 (1344892848.0, [(16, 80), (17, 401), (18, 3), (18, 4), (17, 33),
                 (17, 34), (17, 35), (17, 49), (18, 40), (18, 42),
                 (18, 43), (18, 2), (18, 50), (18, 53), (18, 2),
                 (18, 55), (18, 30), (18, 56), (18, 30), (18, 3),
                 (18, 20), (17, 33), (17, 34), (17, 40), (18, 60),
                 ... ])
]
```

Figura 3.5: Ejemplo de archivo del historial de un paciente a partir de los recursos de la figura 3.4

los `features` que convierte sus tokens únicos a identificadores. La instancia de esta clase fue guardada en un archivo, para poder volver de los identificadores al texto, y poder ser utilizada posteriormente para crear historiales para nuevos pacientes.

Finalmente, producimos un archivo para cada paciente que contiene un diccionario con dos llaves: "pers" y "enc". Donde cada valor de las llaves, contiene la lista de pares que visualizamos en la Figura 3.5, correspondiente a los *features* y *tokens* de cada recurso en orden cronológico. Los recursos que están en la llave "pers", son los que no están asociados a un encuentro, si no que son propios de la persona (más detalles en el apéndice A).

#### Visualizaciones de los *features* y *tokens*

Luego de procesar los historiales de todos los pacientes con la clase `FeaturesAndTokens`, observamos en las Figuras 3.6 y 3.7 los *features* existentes y la cantidad de *tokens* de cada uno. Podemos ver que los *features* que contienen texto tienen muchos más *tokens* únicos.

## 3.4. Modelos predictivos

Diseñamos y desarrollamos dos modelos predictivos de re hospitalización en 30 días. Ambos son capaces de incluir los datos a nivel personal, pero en este trabajo de título alcanzamos a probar considerando solamente los recursos de los encuentros. A continuación describimos cómo cada uno de ellos representa $x$ a partir de los *features* y *tokens*, y sus arquitecturas y detalles de implementación. En la Sección 3.4.1 presentamos el modelo estadístico simple que utilizamos de referencia, y en la Sección 3.4.2 explicamos los detalles del modelo de tipo deep learning.

El modelo simple (3.4.1) fue entrenado utilizando un procesador Intel i5-7267U 3.10GHz, y el modelo de tipo deep learning (3.4.2) fue entrenado utilizando una tarjeta gráfica RTX 2080 Ti 11GB.

### 3.4.1. Bolsa de palabras basada en TF-IDF

Definimos un modelo simple basado en estadística y características relevantes para comparar la red neuronal. Nos basamos en el modelo de referencia utilizado en Huang et al. [22] para definir nuestra bolsa de palabras.

| feature ID | feature | cantidad de tokens |
|---|---|---|
| 1 | recurso encuentro\|clase\|categorical\|id | 5 |
| 2 | recurso encuentro\|tipo\|categorical\|id | 22 |
| 3 | recurso encuentro\|razon\|text | 7453 |
| 4 | recurso diagnostico\|diagnostico\|coding system\|... | 3 |
| 5 | recurso diagnostico\|diagnostico\|code value\|cat... | 25075 |
| 6 | recurso diagnostico\|tipo\|categorical\|id | 7 |
| 7 | recurso nota clinica\|titulo\|categorical\|id | 1286 |
| 8 | recurso nota clinica\|contenido\|text | 1199562 |
| 9 | recurso nota clinica\|estado\|id | 3 |
| 10 | recurso orden\|unidad medica\|categorical\|id | 154 |
| 11 | recurso orden\|tipo\|categorical\|id | 28 |
| 12 | recurso orden\|indicacion\|categorical\|id | 4598 |
| 13 | recurso orden\|sinonimo\|categorical\|id | 13032 |
| 14 | recurso orden\|resumen\|text | 71480 |
| 15 | recurso orden\|unidad\|categorical\|id | 6 |
| 16 | recurso formulario\|nombre\|categorical\|id | 15038443 |
| 17 | recurso formulario\|contenido\|campo\|text | 5672 |
| 18 | recurso formulario\|contenido\|valor\|text | 929794 |
| 19 | recurso orden\|catalogo\|categorical\|id | 16 |
| 20 | recurso diagnostico\|descripcion manual\|text | 15877 |
| 21 | recurso diagnostico\|area que ingresa\|categoric... | 4 |
| 22 | recurso diagnostico\|servicio clinico\|categoric... | 193 |
| 23 | recurso diagnostico\|ranking\|categorical\|id | 2 |

Figura 3.6: *Features* y *tokens* presentes en los recursos de los encuentros

| feature ID | feature | cantidad de tokens |
|---|---|---|
| 1 | recurso persona\|sexo\|categorical\|id | 3 |
| 2 | recurso persona\|nacionalidad\|categorical\|id | 105 |
| 3 | recurso persona\|techo de anos a 2019 | 110 |
| 4 | recurso problema\|diagnostico\|coding system\|cat... | 4 |
| 5 | recurso problema\|diagnostico\|code value\|catego... | 20677 |
| 6 | recurso problema\|descripcion manual\|text | 19401 |
| 7 | recurso nota clinica\|titulo\|categorical\|id | 124 |
| 8 | recurso nota clinica\|contenido\|text | 33135 |
| 9 | recurso nota clinica\|estado\|id | 3 |

Figura 3.7: *Features* y *tokens* presentes en los recursos a nivel personal

**BOW**. Una bolsa de palabras o BOW por las siglas en inglés, es un método simple que utiliza la cantidad de apariciones de una palabra para representar algún texto. Es decir, modela un corpus a través de un vector, donde cada posición contiene el número de apariciones de cada palabra. Se le conoce como *bolsa* pues ignora el orden y las relaciones de las palabras.

Utilizar todos los *tokens* de los distintos *features* es probablemente una mala idea para representar la entrada. Además, buscamos simular una elección manual de señales importantes, por lo que decidimos utilizar TF-IDF como métrica para seleccionar los *tokens* relevantes.

**TF-IDF**. Es una medida numérica estadística, que recibe su nombre de las siglas en inglés de: Frecuencia de un Término - Frecuencia Inversa de Documento [43]. Busca reflejar qué tan importante es una palabra dentro de un documento, dado un conjunto de documentos. Se define como:

$$\mathrm{tf}(t, \mathrm{d}) = \frac{\mathrm{f}(t, \mathrm{d})}{|\mathrm{d}|} \tag{3.1}$$

$$\mathrm{idf}(t, D) = \log\left(\frac{|D|}{|\{\mathrm{d} \in D : t \in \mathrm{d}\}|}\right) \tag{3.2}$$

$$\text{tf-idf}(t, \mathrm{d}, D) = \mathrm{tf}(t, \mathrm{d}) * \mathrm{idf}(t, D) \tag{3.3}$$

Donde $\mathrm{f}(t, \mathrm{d})$ es la frecuencia de $t$ en el documento d y $D$ es el conjunto de documentos. Así,

tf pondera lo frecuente que es la palabra en el documento actual, e idf representa qué tan común o extraña es esta palabra en el resto de los documentos. En consecuencia, las palabras que son muy frecuentes en todos los documentos tendrán un idf cercano a 0 y por ende un tf-idf bajo. En cambio para las palabras que aparecen en unos pocos documentos, su idf será parecido por el logaritmo, y el valor tf-idf dependerá principalmente de la frecuencia en el documento actual.

Así, el vector $x$ de entrada al modelo es una bolsa de palabras de los 5000 tokens más relevantes según su valor tf-idf. Consideramos el valor final tf-idf de un token como el promedio de los que obtiene en cada documento, donde un documento es simplemente todos los *tokens* de todos los *features* de un paciente.

Luego, entrenamos una regresión logística sobre este vector de cuentas de tamaño 5000, minimizando el logaritmo negativo de la verosimilitud (NLL).

### 3.4.2. LSTM basada en embeddings ponderados

Luego de haber decidido utilizar el modelamiento a través de *features* y *tokens* con el formato FHIR (Sección 3.3), encontramos que tenía sentido probar una de las redes neuronales utilizadas en la misma publicación [42]. Elegimos implementar el primer modelo, que está compuesto principalmente por una LSTM que recibe secuencias que representan los datos dentro de una ventana fija de tiempo.

Rajkomar et al. [42] plantean representar el historial de un paciente a través de una secuencia de vectores $x_t$, donde $x_t$ representa el historial del paciente en una ventana de tiempo fija, que ellos utilizan como 12 horas. Definen cada $x_t$ como la concatenación de los embeddings de cada uno de los *features* existentes, más un embedding que representa el tiempo, donde el embedding de cada *feature*, está definido como el promedio ponderado de los distintos *tokens* presentes en la ventana. Por ejemplo, observamos en la parte inferior de la Figura 3.8 que el $x_t$ final, es la concatenación de los embeddings para cada *feature*; En particular, notamos que la parte correspondiente al *feature* con ID 3 ("medication_order.ingredient.text"), está formada por el promedio ponderado de los embeddings correspondientes al *token* 85 ("Piperacillin") y al *token* 19 ("Tazobactam"), utilizando los pesos que se encuentran a la derecha.

El tiempo lo representamos a través de un embedding que es concatenado al final de cada secuencia $x_t$. El vector que contiene el embedding correspondiente, lo obtenemos de la misma forma que en la publicación de Rajkomar et al. [42]. Utilizamos la diferencia de tiempo en segundos al momento de la predicción, y tomamos el entero siguiente del logaritmo del promedio de las diferencias de tiempo, entre todos los recursos de la ventana, dividido por un factor manejado como hiperparámetro. Considerando que las diferencias de tiempo pueden ir desde 1 segundo hasta los casi 9 años de datos que tenemos, esta discretización tiene el efecto de considerar de igual forma los eventos dentro de un período, donde para períodos más cercanos considera ventanas pequeñas en que los considera de igual forma, pero para sucesos lejanos son todos considerados igual. Visualizamos esto en la Figura 3.9.

Esta arquitectura de embeddings es entrenada en conjunto con los pesos de la LSTM. Es decir, los valores de los embeddings de los tokens y de los pesos para ponderar, son aprendidos en el entrenamiento del clasificador.

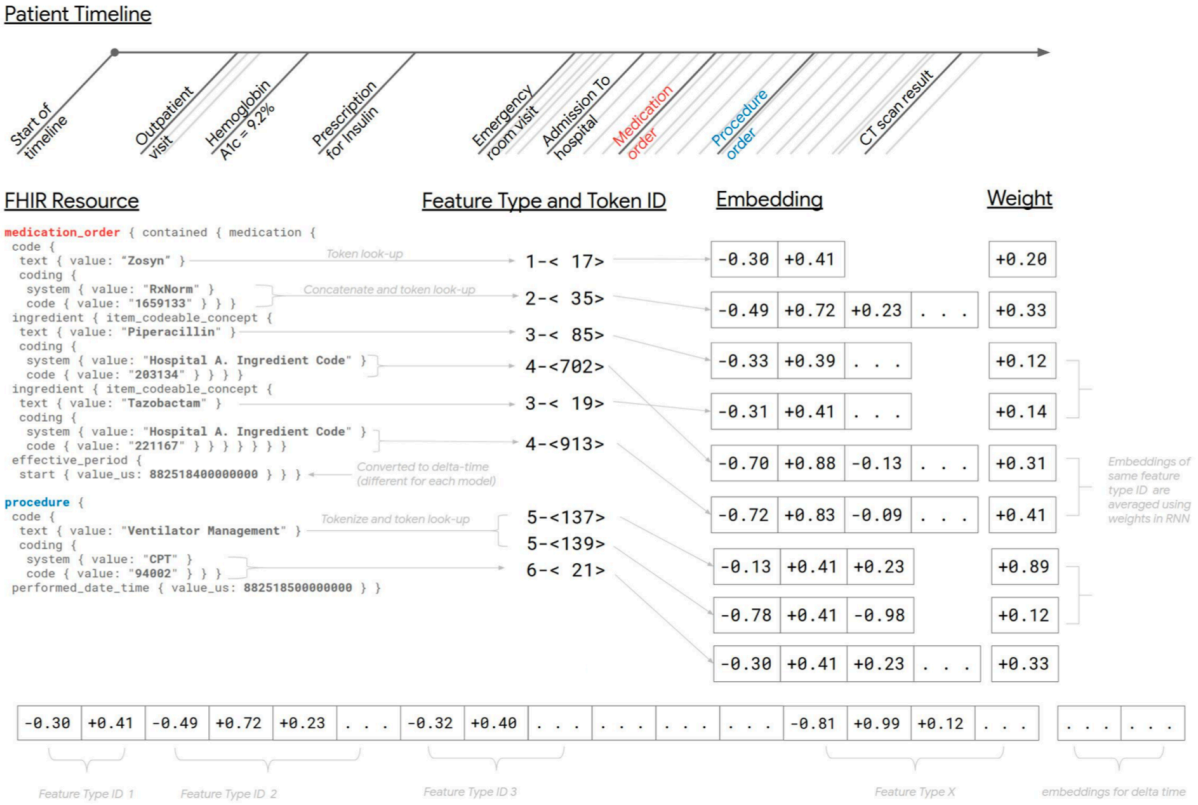



Figura 3.8: Conversión de recursos FHIR a embeddings (tomada de Rajkomar et al. [42])

En Rajkomar et al. [42] definieron un mínimo de frecuencia que debía tener cada *token* para ser considerado. Si la frecuencia era menor a eso, entonces se utilizaba algún embedding para *tokens* fuera del vocabulario. Ellos decidieron que un *token* debía estar al menos 2 veces en el conjunto de entrenamiento, o al menos 100 si es que era un código de observación o un valor numérico concatenado a su unidad de medida. En nuestro caso, decidimos considerar un mínimo de frecuencia de 3 para cada *token*, pues revisando los recursos FHIR que utilizaron en su trabajo, notamos que los nuestros poseían considerablemente más texto libre. Además, definimos un embedding extra para cada *feature* que será el mapeo que se considerará para *tokens* poco frecuentes o fuera del vocabulario en los conjuntos de prueba. La cantidad de tokens por *feature* luego de eliminar los menos frecuentes puede visualizarse en 3.10. Además, revisamos los *tokens* que fueron eliminados, y en su mayoría corresponden a errores de escritura, dentro de los cuales en notas clínicas por ejemplo encontramos: 'ganglionar', 'nauseasdolor', 'gargantat', 'desmao', 'tecg', 'dlolor', 'diareatos', 'maeos', 'voitos'.

Por otro lado, notamos que los embeddings de los *tokens* son particulares para cada *feature*, es decir, aprendemos una matriz para cada *feature* donde los vectores son los embeddings de los *tokens*. Por lo tanto pueden tener distinto tamaño, tal como se ve en la Figura 3.8. De hecho en Rajkomar et al. [42] determinan el tamaño de cada uno de los embeddings utilizando la heurística:

$$6\alpha\sqrt[4]{V}, \tag{3.4}$$

donde $V$ es el tamaño del vocabulario del *feature* y $\alpha$ es un hiperparámetro. Con esto, plantearon que el tamaño final de $x_t$ fue 100.

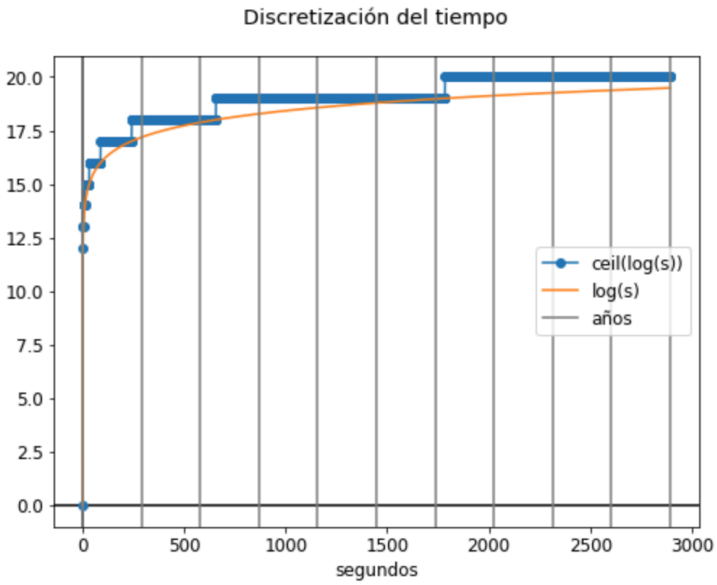



Figura 3.9: Visualización de la discretización del tiempo

En nuestro caso, utilizando la misma heurística (3.4) y una mínima frecuencia de 3 por *token*, obtuvimos que la secuencia $x_t$ tenía tamaño 509. Decidimos ajustar un poco la heurística para que el tamaño de la secuencia fuese más parecido, llegando a un tamaño de secuencia 176 con la heurística:

$$3\alpha\sqrt[5]{V}. \tag{3.5}$$

En el modelo presentado por Rajkomar et al. [42] utilizan varias regularizaciones, concretamente: dropout en los embeddings, zoneout [26], dropout en el estado oculto de la LSTM, RNN dropout variado [25] y L2. Agregamos todas estas regularizaciones, excepto las que aún no están implementadas en Pytorch: zoneout[2] y dropout variado[3]. Decidimos no incluirlas pues implementarlas implicaba utilizar tiempo en detalles que no iban totalmente de la mano con los objetivos este trabajo.

Al igual que en la publicación [42] el modelo fue optimizado para minimizar el logaritmo negativo de la regresión logística final con la que realizamos la clasificación. Pero a diferencia de ellos, utilizamos Adam [24] para la optimización en lugar de Adagrad [16], pues es más independiente de la tasa de aprendizaje definida en un inicio. También, utilizamos los mismos hiperparámetros encontrados por Rajkomar et al. [42] a través de optimización bayesiana.

A continuación explicamos algunas modificaciones que decidimos realizarle a la arquitectura planteada por Rajkomar et al. [42]. Estos cambios pueden ser comprendidos de mejor forma observando la figura 3.11.

### Pesos no negativos

En la publicación, determinan que los pesos para ponderar los embeddings deben ser valores no negativos pues queremos realizar un promedio ponderado entre los vectores, entonces

[2]Pull request abierto hasta la fecha: `https://github.com/pytorch/pytorch/pull/4838`
[3]`https://discuss.pytorch.org/t/variational-dropout/23030/6`

| feature ID | feature | Porcentaje poco frecuente | cantidad tokens final |
|---|---|---|---|
| 1 | recurso encuentro\|clase\|categorical\|id | 20.00% | 4 |
| 2 | recurso encuentro\|tipo\|categorical\|id | 4.55% | 21 |
| 3 | recurso encuentro\|razon\|text | 64.98% | 2610 |
| 7 | recurso nota clinica\|titulo\|categorical\|id | 14.70% | 1097 |
| 8 | recurso nota clinica\|contenido\|text | 65.25% | 416854 |
| 9 | recurso nota clinica\|estado\|id | 0.00% | 3 |
| 17 | recurso formulario\|contenido\|campo\|text | 7.19% | 5264 |
| 18 | recurso formulario\|contenido\|valor\|text | 61.91% | 354130 |
| 10 | recurso orden\|unidad medica\|categorical\|id | 4.55% | 147 |
| 11 | recurso orden\|tipo\|categorical\|id | 0.00% | 28 |
| 12 | recurso orden\|indicacion\|categorical\|id | 10.40% | 4120 |
| 13 | recurso orden\|sinonimo\|categorical\|id | 18.72% | 10593 |
| 14 | recurso orden\|resumen\|text | 60.71% | 28084 |
| 15 | recurso orden\|unidad\|categorical\|id | 0.00% | 6 |
| 19 | recurso orden\|catalogo\|categorical\|id | 0.00% | 16 |
| 4 | recurso diagnostico\|diagnostico\|coding system\|... | 0.00% | 3 |
| 5 | recurso diagnostico\|diagnostico\|code value\|cat... | 52.84% | 11825 |
| 20 | recurso diagnostico\|descripcion manual\|text | 47.42% | 8348 |
| 6 | recurso diagnostico\|tipo\|categorical\|id | 0.00% | 7 |
| 21 | recurso diagnostico\|area que ingresa\|categoric... | 0.00% | 4 |
| 22 | recurso diagnostico\|servicio clinico\|categoric... | 26.94% | 141 |
| 23 | recurso diagnostico\|ranking\|categorical\|id | 0.00% | 2 |

Figura 3.10: Cantidad de *tokens* por *feature* luego de eliminar los que tenían frecuencia menor a 3

no tendría sentido tomar el promedio con valores negativos. En Pytorch no es posible condicionar un embedding para que sus valores sean siempre positivos, por lo que para asegurar esta condición evaluamos en primer lugar, utilizar la función ReLu sobre el vector de pesos (el embedding de tamaño 1). Pero el principal problema con esta estrategia era que si en algún punto el valor del peso se hacía 0, entonces la ponderación a este token nunca iba a poder aumentar su valor, puesto que estaría "apagada" para la red y tendría gradiente 0 al momento de realizar la propagación del error final. En consecuencia, decidimos que era una mejor idea aprender estos pesos a través de una red *feed-forward* de una capa que en la salida tuviese alguna función que asegurara la condición de no negatividad. Así, los pesos en nuestra red son obtenidos de una capa lineal con una función sigmoide en la salida. Analizamos también la opción de utilizar la función softmax, pero no buscábamos que la atención se concentrara en algunos valores en particular, por lo cual preferimos la sigmoide para mantener los pesos positivos.

### Variación para formularios

En los datos de la Clínica Las Condes tenemos información almacenada en formularios, que luego de la transformación, quedaron definidos como el "recurso formulario". Pero en realidad no existe ningún recurso similar en el formato FHIR, a diferencia del resto de los recursos que definimos según los datos de la clínica. Recordemos que el recurso formulario (A.6) está compuesto por un título y una lista de campos y valores. Si lo consideráramos de la misma forma que lo hacemos con el resto de los recursos, tendríamos los *features*: "recurso formulario.titulo", "recurso formulario.contenido.campo", "recurso formulario.contenido.valor";

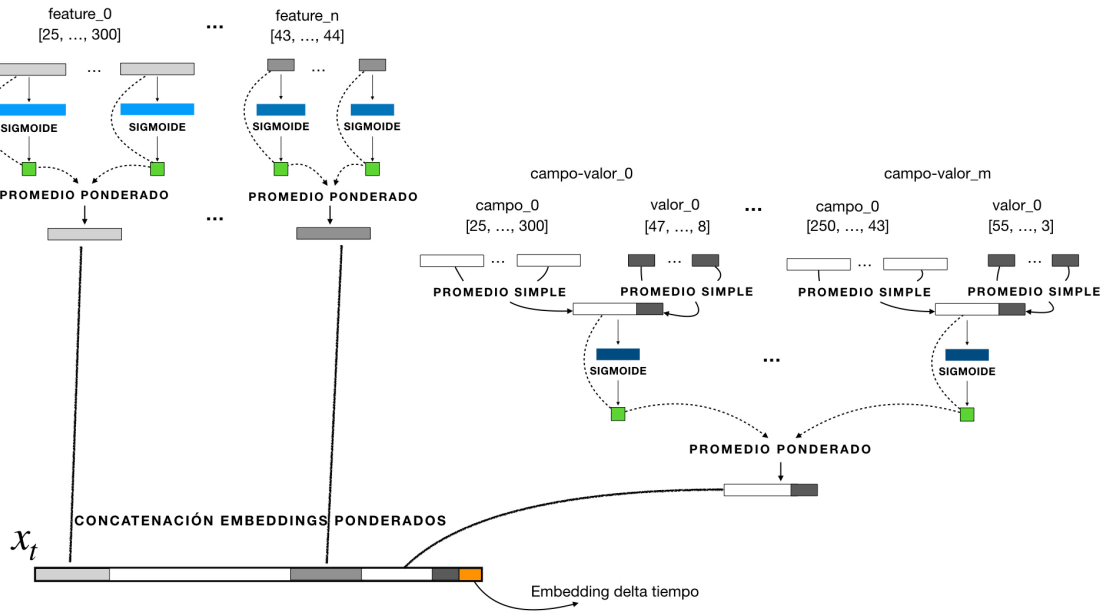



Figura 3.11: Arquitectura para calcular los vectores de la secuencia de entrada de la LSTM

Entonces el promedio de los embeddings de los *tokens* de los campos representaría el *feature* de los campos, y el promedio de los embeddings de los *tokens* de los valores representaría el *feature* de los valores. Pero el principal problema con realizar esto, es que estaríamos perdiendo la relación campo-valor, es decir, la relación de la respuesta con su pregunta. En consecuencia, decidimos buscar alguna representación que tuviera sentido con el diseño del resto de la red, y que además tuviera en consideración la relación de pares entre estos dos *features*.

Lo que hicimos fue considerar los datos del recurso formulario aparte, y determinamos que la entrada de estos datos estaría dada por los pares campo-valor, donde cada parte está compuesta por un conjunto de *tokens*, pues cada una es simplemente texto. Luego, obtenemos la representación de un campo o valor realizando un promedio simple entre los embeddings de los *tokens* que lo componen. Posteriormente, concatenamos el embedding del campo con el de su valor correspondiente, y consideramos este vector como el embedding de un "token" dentro del *feature* campo-valor. Dicho de otro modo, tomaremos el vector concatenado y obtendremos el peso que le corresponde a través de una red de una capa, para luego realizar un promedio ponderado entre todos los vectores concatenados, entre todos los pares campo-valor que hayan en la ventana de tiempo. Con lo que finalmente formamos un vector que representa todos los pares campo-valor y que concatenaremos a la secuencia $x_t$.

En un inicio pensamos realizar un promedio ponderado entre los *tokens* que forman los campos o valores, para continuar con la misma arquitectura de la red. Pero la cantidad de tokens que forman todos los pares dentro de una ventana era demasiado alta, e instanciar cada embedding implicaba utilizar demasiada memoria (en la mayoría de los casos más de la que teníamos disponible). En consecuencia, decidimos utilizar una estructura de Pytorch llamada `EmbeddingBag`, que es capaz de realizar una operación simple (suma, promedio, máximo) entre una colección de embeddings sin instanciarlos, siendo mucho más eficiente en memoria. Por lo tanto, realizamos un promedio simple entre los tokens que forman un campo

o valor.

### Observaciones de la implementación

Un detalle relevante en la implementación de la red es que debimos generar un nuevo mapeo de los índices, pues buscamos que los índices de los *tokens* de un *feature* sean secuenciales para que representen la posición del vector correspondiente en la matriz de embeddings. Los índices antiguos cambian por la mínima frecuencia que decidimos imponer en los *tokens*, lo que provoca que los índices que estén bajo ese valor ahora reciban la posición del *token* especial de palabras fuera de vocabulario, y su índice ya no está presente. Es importante este nuevo mapeo por dos cosas:

1. La conversión de los índices debe ser la misma para los datos de entrenamiento, validación y testeo.
2. El mapeo debe ser el mismo si posteriormente queremos seguir entrenando un modelo por más épocas.

Pues la red utiliza los índices para reconocer los distintos tokens y obtener sus embeddings, por lo que si cambiamos los índices claramente no estaremos usando los mismos vectores y por ende no obtendremos los mismos resultados. Para manipular esta conversión, creamos la clase `PreProcessTokensFeatures` para almacenar los diccionarios que realizan este mapeo, y así, poder compartir la instancia de este objeto entre los distintos conjuntos de datos (entrenamiento, validación y testeo), y poder almacenarla para su uso posterior.

### Formato de la entrada de la red

La entrada a la red debe contener los identificadores de los *tokens* para cada *feature*, para cada una de las ventanas de 12 horas del historial del paciente, para así, promediar con pesos los tokens de cada feature en cada ventana, y obtener las secuencias $x_t$ que serán la entrada a la LSTM. Debimos definir entonces, qué formato tendría la entrada para que contuviese toda esta información.

Es necesario notar que el mayor problema de esta decisión, es que cada paciente puede tener distinto número de ventanas y cada *feature* puede tener distinto número de *tokens*. Por lo que representarlo por un tensor de dimensiones: `(cantidad de ventanas, features, cantidad de tokens)`, implicaría tener que realizar padding en la tercera dimensión para una misma persona, y si consideramos batches con tamaño $> 1$, entonces habría que realizar padding también en la primera dimensión.

Además, hay que tener en consideración, que buscamos realizar operaciones con tensores en vez de iterar sobre arreglos, pues las primeras son altamente más paralelizables. Es por esto que tuvimos en mente dos representaciones:

1. El historial de un paciente es representado por un tensor de dimensiones: (cantidad de ventanas, máximo número de tokens en una ventana, 2). De tal forma que para cada ventana tenemos pares (*feature* ID, *token* ID), y como puede haber distinto número de *tokens* en una ventana, debemos hacer padding y dejar la segunda dimensión con el máximo número de *tokens* entre las ventanas. En esta representación un batch sería

una lista de estos tensores.
2. El historial de un paciente es representado por una lista de tensores, donde cada uno contiene todos los tokens dentro de una ventana. Además, tenemos una lista para cada ventana que indica a qué *feature* corresponden los *tokens* del tensor, para poder realizar el promedio correspondiente.

La principal ventaja de la primera opción es que como tenemos un tensor que representa todos los features dentro de una ventana, podemos realizar el promedio ponderado para todas las ventanas directamente. En cambio en la segunda opción, como cada ventana es un elemento de una lista, debemos iterar sobre ellas. Pero la gran desventaja es que debemos realizar padding en la primera y en la segunda dimensión para batches >1. Probamos primero utilizando la primera representación, pero era necesario acotar demasiado los tamaños de la entrada (detalles en 3.4.2) para que el modelo entrara en la memoria de la GPU, por lo que decidimos utilizar la segunda representación.

#### Tamaño del modelo y tamaño de la entrada

El tamaño de la entrada a la red neuronal queda definido por:

1. Tamaño del batch,
2. la cantidad de ventanas de cada ejemplo,
3. la cantidad total de tokens por ventana,
4. la cantidad de pares campo-valor por ventana,
5. el largo de la lista de *tokens* que compone un campo o un valor.

En la publicación de Rajkomar et al. [42] utilizan un batch de tamaño 128, nosotros probamos con un batch de tamaño 64 y los máximos valores posibles para cada ítem (2,3,4,5), y no cabía en la memoria de la GPU. Por lo que decidimos definir un máximo para cada uno de estos largos para poder asegurar que ningún conjunto de ejemplos fuese a tener problemas de memoria. Estos máximos son considerados en el momento de generar los ejemplos, la red neuronal no ve la información eliminada en ningún momento. Para definir estos valores, calculamos primero la cantidad de ventanas por persona, la cantidad de *tokens* y pares por ventana, y los largos de la lista que compone el campo y el valor. Usando estos valores definimos los máximos tamaños tales que un batch de tamaño 64 con esos máximos entrara en memoria.

Para la cantidad de ventanas de cada ejemplo, definimos un largo máximo de 230, y decidimos que para los historiales que sobrepasaran este número consideraríamos solamente las últimas 230 ventanas (las 230 más cercanas al momento de la predicción). Solamente un 0.24 % de las ventanas sobrepasa el tamaño y por ende es truncada.

Para la cantidad de *tokens* por ventana, es complejo decidir qué *tokens* ignorar en caso de sobrepasar el máximo, por lo que decidimos simplemente eliminar las personas que tienen demasiados *tokens* en una ventana. Establecimos este máximo en 50.000 *tokens*, que corresponde a un 0.08 % de las ventanas que se distribuyen en un 2 % de pacientes distintos. Eliminar estos pacientes de los datos implicó perder un 10 % de los ejemplos (hospitalizaciones válidas), cambiando un poco la distribución de positivos y negativos (Tabla 3.1).

| | Total | Luego de eliminar |
|---|---|---|
| **Positivos** | 18.258 (9.85 %) | 11.459 (6.18 %) |
| **Negativos** | 167.065 (90.15 %) | 173.864 (93.82 %) |

Tabla 3.1: Distribución ejemplos antes y después de eliminar los historiales de pacientes que contienen ventanas que sobrepasan el límite de *tokens*

Luego, consideramos un máximo de 800 pares campo-valor dentro de una misma ventana. Para los casos donde este valor era sobrepasado, ignoramos los pares siguientes. Del total de formularios, solamente un 1.27 % sobrepasa este máximo.

Por último, establecimos un máximo largo de *tokens* que componen un campo, y otro largo máximo para los que componen un valor. Si alguno de los dos sobrepasa su valor, el par completo no es considerado. Para el campo, definimos el máximo en 3000, y para el valor en 5000. La cantidad de veces que estos valores son sobrepasados es: 1.2 % y un 1.1 % respectivamente.

Con estos máximos, y utilizando la heurística 3.5 y los mismos hiperparámetros que en Rajkomar et al. [42], el modelo completo posee 14,39 millones de parámetros entrenables.

## 3.5. Datos finales

### 3.5.1. Conjunto final

La base de datos de la Clínica Las Condes contiene 118.174 pacientes que han sido hospitalizados, y un total de 211.647 admisiones. Luego de seleccionar las hospitalizaciones válidas y eliminar los pacientes que sobrepasaban el máximo de *tokens* en una ventana de 12 horas, el conjunto final de datos contiene:

- 114.685 pacientes
- 185.323 hospitalizaciones
- Readmisiones en 30 días: 11.459 (6,18 %)
- Hospitalizaciones sin readmisión en 30 días: 173.864 (93,82 %)

El conjunto final es más grande y contiene más ejemplos positivos que el utilizado en Huang et al. [22], que contenía 34.560 pacientes, con 2.963 (5,79 %) readmisiones en 30 días, y 48.150 (94,2 %) etiquetas negativas.

Además observamos en primer lugar, que las hospitalizaciones válidas no están concentradas en unos pocos pacientes, de hecho más del 70 % de los pacientes tiene sólo una admisión válida, y alrededor de un 20 % posee 2. De igual forma, notamos que más del 80 % de los pacientes que tienen re-hospitalizaciones en 30 días, tienen solamente 1, lo cual nos indica que los ejemplos positivos no están concentrados en un conjunto pequeño de personas.

### 3.5.2. Conjuntos de entrenamiento, validación y prueba

Utilizamos los porcentajes estándares para los conjuntos de entrenamiento, validación y prueba: 80 %, 10 % y 10 %, respectivamente.

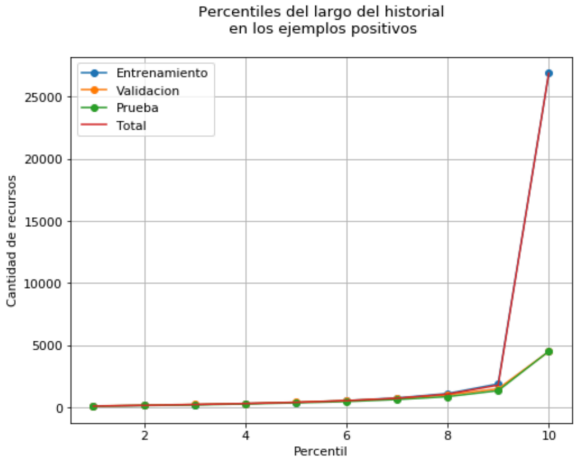



Figura 3.12: Largo del historial de los ejemplos positivos para cada conjunto

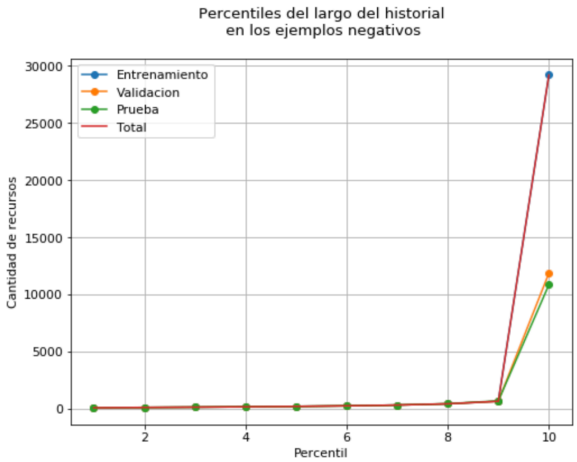



Figura 3.13: Largo del historial de los ejemplos negativos para cada conjunto

Un aspecto importante que tuvimos en consideración fue evitar que algunos ejemplos de una persona se encontraran en el conjunto de entrenamiento, y otros en el de test. Pues en ese caso, el modelo tendría información extra al momento de probarlo. Por ejemplo, si una persona tiene 2 hospitalizaciones válidas, en las fechas $a, b$ con $a > b$, entonces aportará con 2 ejemplos, donde el primero contiene el historial hasta $a$ y el segundo hasta $b$. Por lo tanto los datos del segundo incluyen los del primero, y así, si pusiéramos el primer ejemplo en el conjunto de test, y el segundo en el de entrenamiento, entonces el modelo ya habría visto lo que pasó después en ese historial. Esta separación por pacientes estuvo inspirada en Arango et al. [4].

En consecuencia, lo que hicimos fue tomar aleatoriamente un 80 % de los pacientes para el conjunto de entrenamiento, y un 10 % aleatorio de los restantes para el conjunto de validación, dejando el 10 % restante como el conjunto de testing. Con lo cual, obtuvimos aproximadamente los mismos porcentajes en la cantidad de ejemplos de cada conjunto, y verificamos que la cantidad de positivos y negativos mantuviese la distribución (Tabla 3.2).

| | **Cantidad de ejemplos** | **Porcentaje del total de ejemplos** | **Porcentaje positivos** | **Porcentaje negativos** |
|---|---|---|---|---|
| **Total** | 185.323 | - | 6.183 % | 93.817 % |
| **Entrenamiento** | 148.510 | 80.13 % | 6.237 % | 93.763 % |
| **Validación** | 18.318 | 9.88 % | 5.819 % | 94.181 % |
| **Testeo** | 18.495 | 9.97 % | 6.115 % | 93.885 % |

Tabla 3.2: Cantidades de ejemplos por conjunto

Además, nos fijamos que los ejemplos en cada uno de los conjuntos tuviesen historiales de largo similar. En particular, visualizamos el largo máximo para cada percentil en cada conjunto y observamos que en un 90 % de los datos son prácticamente iguales en largo (Figura 3.5.2).

# Capítulo 4

# Resultados y análisis

## 4.1. Resultados

Entrenar la LSTM basada en embeddings ponderados por una época toma 11 horas, y luego realizar la predicción en el conjunto de validación toma 1 hora. Estos tiempos probablemente se deban a la cantidad de iteraciones que realizamos sobre cada ejemplo del batch, y luego sobre cada ventana y cada *feature*. El modelo BOW basado en TF-IDF toma alrededor de 7 horas y media en realizar una época, y luego predecir el conjunto de validación toma 1 hora. Debemos recordar que este segundo modelo fue entrenado en una CPU, a diferencia del otro que entrenamos en una GPU.

El modelo BOW lo entrenamos durante 8 épocas utilizando el optimizador SGD (gradiente estocástico descendente), pues dio mejores resultados que utilizando algoritmos más sofisticados como Adam. El modelo basado en LSTM fue entrenado por 6 épocas utilizando Adam, tal como especificamos en la Sección 3.4.2. En ambos entrenamientos seleccionamos el modelo que tenía mejores resultados según la métrica AUROC en el conjunto de validación. En la Tabla 4.1 reportamos los resultados obtenidos por esos modelos en el conjunto de prueba. Ambos modelos fueron reentrenados por algunas épocas adicionales (el modelo BOW por 8 épocas más y el LSTM por 3 extras), pero ninguno mejoró sus resultados tal como podemos visualizar en las Figuras 4.3 y 4.6, en las épocas luego de la línea negra vertical. Adicionalmente, entrenamos el modelo LSTM sobre un conjunto de entrenamiento con *oversampling*, es decir, repetimos aleatoriamente ejemplos positivos en el conjunto de entrenamiento hasta llegar a 20 % (número que decidimos intuitivamente, pues nos pareció que uno mayor implicaría repetir más de 4 veces los ejemplos). Los resultados se observan también en la Tabla 4.1.

| | AUROC |
|---|---|
| BOW con TF-IDF | 0.5807 |
| LSTM + embeddings | **0.7626** |
| LSTM + embeddings (oversampling 20 %) | 0.7499 |

Tabla 4.1: Valores AUROC en el conjunto de prueba

También calculamos los valores de precisión y recall por clase para observar el comportamiento de los modelos en cada caso. Estos resultados los encontramos en la Tabla 4.2.

| | Precision clase 0 | Recall clase 0 | Precision clase 1 | Recall clase 1 |
|---|---|---|---|---|
| BOW + TF-IDF | **0.95** | 0.96 | 0.26 | 0.23 |
| LSTM + embeddings | **0.95** | **0.99** | **0.52** | 0.13 |
| LSTM + embeddings (oversampling 20 %) | **0.95** | 0.97 | 0.36 | **0.29** |

Tabla 4.2: Precisión y recall por clase en el conjunto de prueba

## 4.2. Análisis

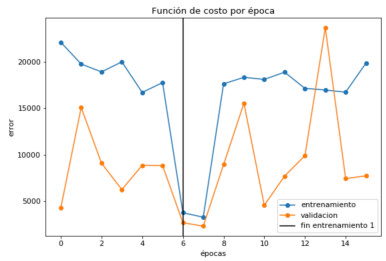



Figura 4.1: Función de costo modelo BOW

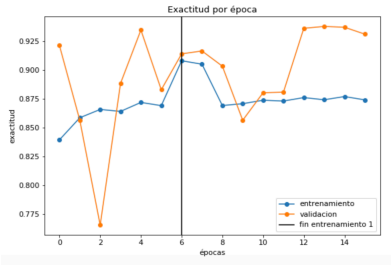



Figura 4.2: Exactitud modelo BOW

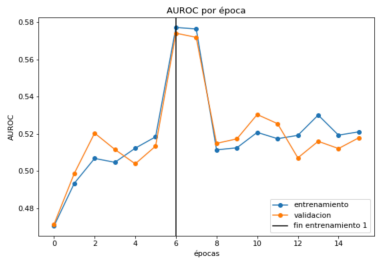



Figura 4.3: AUROC modelo BOW

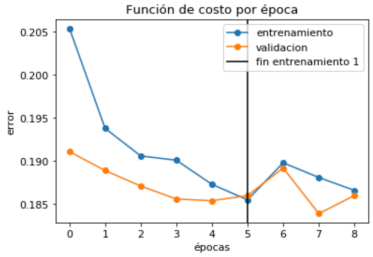



Figura 4.4: Función de costo modelo LSTM

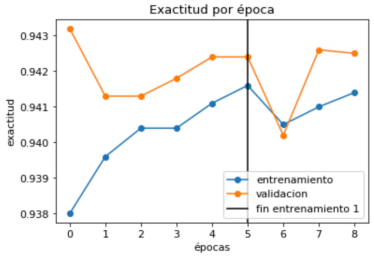



Figura 4.5: Exactitud modelo LSTM

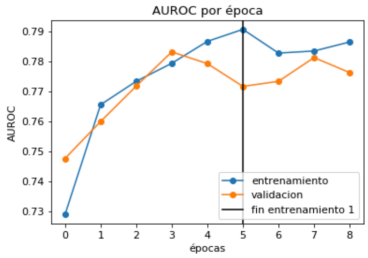



Figura 4.6: AUROC modelo LSTM

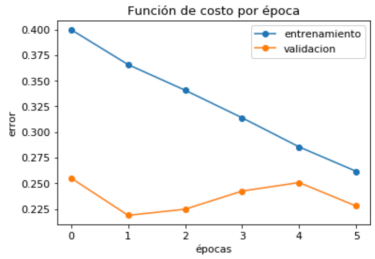



Figura 4.7: Función de costo modelo LSTM + oversampling

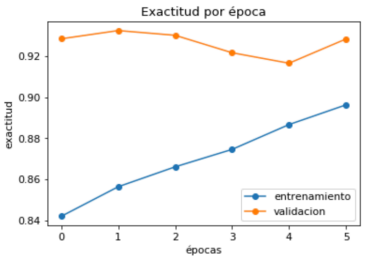



Figura 4.8: Exactitud modelo LSTM + oversampling

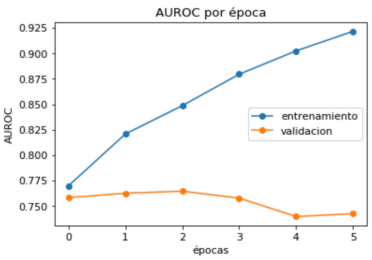



Figura 4.9: AUROC modelo LSTM + oversampling

### Entrenamiento

De las Figuras 4.1 - 4.6, observamos que no hubo mayor mejora en las épocas de entrenamiento extra. Una razón de esto puede ser que en el primer entrenamiento no guardamos el estado del optimizador, y entonces la segunda vez el modelo comenzó con los pesos anteriores, pero con una tasa de aprendizaje incorrecta y no logró encontrar una óptima. En el caso de la LSTM (Figuras 4.4, 4.5 y 4.6), notamos que en la primera época adicional las distintas métricas reciben valores menores, pero luego tienden a los valores logrados anteriormente.

Observamos también en las Figuras 4.1, 4.4 y 4.7, que el error en el conjunto de validación es menor que en el de entrenamiento. Esto tiene sentido pues al evaluar el modelo en el conjunto de validación no utilizamos regularizaciones, y además a lo largo de los batches del entrenamiento el costo va cambiando pues vamos ajustando el modelo. También hay que recordar que el conjunto de validación tiene menos ejemplos positivos (Tabla 3.2), lo que también puede provocar menor error en el caso de que el modelo tenga bajo error en los casos negativos (cosa que ocurre dado que son mayoría).

En cuanto al entrenamiento del modelo LSTM con oversampling, observamos que no se logra generalizar, pues el error en el conjunto de validación no logra ser disminuido mientras que en el conjunto de entrenamiento si baja. De igual forma, la exactitud y el valor AUROC se mantienen casi constantes en el conjunto de validación. De todas formas esto tiene sentido si pensamos que en el entrenamiento puede que se esté mejorando en disminuir el error y aumentar la exactitud en los ejemplos positivos, que son menos en el conjunto de validación, y que por ende podría no verse esta optimización del modelo. De todas formas, pareciera que el modelo se sobreajusta a los ejemplos positivos del entrenamiento, pues en la época de entrenamiento final llegó a obtener una precisión de 0.80 y un recall de 0.64 en la clase positiva, resultando en un AUROC de 0.92, pero que luego al ser probado en el conjunto de validación bajaba a valores de 0.35 (precisión) y 0.26 (recall), y el AUROC llegaba solamente a 0.74 en esa época.

### Métricas

Con respecto a los resultados de la Tabla 4.1 observamos que el modelo basado en LSTM supera considerablemente al basado en BOW. Es decir, a pesar de tener miles de parámetros más, logra una mejor representación del historial de un paciente a través del uso de los embeddings y la LSTM. Además, el valor AUROC obtenido es bueno también en comparación a la publicación que seguimos de guía para diseñar el modelo ([42]), pues allí alcanzaron un 0.77 y 0.76 en dos hospitales distintos. Notamos además que la LSTM con oversampling no logró entregar mejores resultados finales, esto se debe probablemente a que como repetimos alrededor de 4 veces los mismos ejemplos el modelo tendió a sobreajustarse en vez de lograr generalizar. En una segunda iteración podríamos probar realizando un menor porcentaje de oversampling o por el contrario hacer downsampling (disminuyendo aleatoriamente los ejemplos negativos).

No obstante, visualizando más métricas, notamos que el recall del modelo para la clase minoritaria es bastante bajo (Tabla 4.2). Lo que quiere decir que estamos detectando solamente un 13 % de los casos de re-hospitalización. Esta métrica logró ser aumentada utilizando oversampling, lo que nos indica que probablemente utilizando menos porcentaje para evitar

el sobreajuste o realizando downsampling como comentábamos podría ser una buena idea para mejorar en recall. Por otro lado, también podríamos intentar modificar la función de costo para que pesara más cuando no se predice de forma correcta los casos que sí se rehospitalizan. De todas formas, habría que ser cuidadosos en este caso pues probablemente el modelo comenzaría a tener más falsos positivos, pues eso minimizaría más la función de costo. Esto puede o no ser deseable, dependiendo de los objetivos y recursos del centro clínico.

### Posibles mejoras

Además de probar oversampling y downsampling, los resultados también podrían ser mejorados realizando una búsqueda de parámetros específica para nuestros modelos, puesto que para los basados en una LSTM utilizamos los mismos que definieron en Rajkomar et al. [42], y considerando que el largo de la secuencia y la cantidad de *features* es distinta, podría tener un efecto considerable para nuestra arquitectura. En el modelo BOW utilizamos la tasa de aprendizaje que viene por defecto (0.01), y regularización L2 estándar (0.001), así que realizar una búsqueda de hiperparámetros también podría mejorar su rendimiento. Por otro lado, recordemos que para el manejo de los números en el texto de los datos, simplemente reemplazamos por el token '<num>'; Aquí podríamos haber probado algún tipo de agrupación o discretización para aprender embeddings específicos, pero el tiempo no permitió hacerlo.

# Conclusión

Luego de haber propuesto y desarrollado un modelo de deep learning para predecir re-hospitalizaciones no planificadas para el caso particular de Chile, concluimos que no es posible replicar directamente los modelos predictivos propuestos en otros países para el caso chileno. En primer lugar, porque en Chile no existen leyes tan rigurosas acerca de la información que deben almacenar los centros clínicos, por lo que en la Clínica Las Condes no poseen los códigos de tratamientos ni de medicamentos que son ordenados a un paciente. En consecuencia, no es posible por ejemplo modelar el historial de un paciente solamente a través de códigos, como realizan en Nguyen et al. [37]. En segundo lugar, en Chile la terminología de referencia definida por el Ministerio de Salud, SNOMED CT, no sirve para identificar hospitalizaciones no planificadas según la definición de la CMS [3]. Por último, en la Clínica Las Condes utilizan "formularios" como una herramienta flexible que les permite almacenar distintos tipos de información. En consecuencia, requieren de modelos sofisticados que sean capaces de rescatar información de texto libre, números, categorías, entre otros, al mismo tiempo. A pesar de que el historial clínico de un paciente tenga algunas diferencias y peculiaridades en comparación a los datos en el extranjero, la información en la Clínica Las Condes ha sido almacenada de forma estructurada en bases de datos relacionales, y es posible utilizarla directamente por programas automáticos.

Como muestra de nuestro trabajo, concluimos que los modelos y técnicas de deep learning son capaces de aprender a detectar patrones dentro de los datos de los pacientes de la Clínica Las Condes. De hecho, el modelo particular presentado y desarrollado en el trabajo de título alcanza un rendimiento a nivel del estado del arte en predicciones de readmisiones hospitalarias. Además reafirmamos con nuestra experiencia, que a pesar de que el modelo de redes neuronales tenga miles de parámetros más que el lineal, es capaz de aprender a diferenciar mejor, los casos positivos de re-hospitalización de los negativos. Por último, es importante concluir que con las técnicas de deep learning existentes hoy en día, es posible desarrollar este tipo de modelos predictivos con los datos disponibles en Chile. Por ende, utilizando los programas de este trabajo de título y el modelo entrenado, se podrían desarrollar sistemas que permitiesen tener el cálculo de probabilidad de readmisión de un paciente al momento del alta.

Obtener los datos, comprenderlos, pre procesarlos y transformarlos para poder utilizarlos por los modelos predictivos tomó alrededor del 90 % del tiempo de este trabajo. Pues debimos primero tener reuniones con distintos grupos de la Clínica Las Condes para lograr la aprobación del proyecto y el posterior acceso a los datos. Luego, entre otras cosas debimos visualizar los datos para comprenderlos y encontrar posibles errores, y generar programas

específicos para poder tratarlos. En consecuencia, tuvimos que tomar decisiones de diseño para simplificar el trabajo, y no pudimos realizar todos los experimentos que nos hubiesen gustado pues el tiempo del trabajo de título es acotado. No obstante, este trabajo se enmarca dentro de un proyecto de innovación de la Clínica Las Condes, y por ende dejamos propuestas algunas modificaciones (en la metodología y en los experimentos) que podrían mejorar los resultados, entre ellas: considerar los datos que contienen la información de los exámenes, realizar algún tipo de discretización con los números que aparecían en el texto libre y hacer una búsqueda de hiperparámetros en los modelos entrenados. Además, podría realizarse un análisis del modelo de deep learning entrenado, para observar a través de los pesos con que se promedian los embeddings, qué datos son más relevantes o utilizados para la predicción, y qué datos quizás deberían ser modelados de alguna otra forma para que la red lograra captar más información de ellos.

# Bibliografía


[1] Universidad de standford. notas de cs224n: Natural language processing with deep learning. `http://web.stanford.edu/class/cs224n/`. Accessed: 2019-10-24.

[2] Wikipedia.com curva roc. `https://en.wikipedia.org/wiki/Receiver_operating_characteristic`. Accessed: 2019-10-17.

[3] 2016 measure updates and specifications report: hospital-wide all-cause unplanned readmission. *YaleNew Haven Health Services Corporation/Center for Outcomes Research Evaluation*, May 2016.

[4] Aymé Arango, Jorge Pérez, and Barbara Poblete. Hate speech detection is not as easy as you may think: A closer look at model validation. In *Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 45–54. ACM, 2019.

[5] Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Jauvin. A neural probabilistic language model. *Journal of machine learning research*, 3(Feb):1137–1155, 2003.

[6] James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13(Feb):281–305, 2012.

[7] Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. Enriching word vectors with subword information. *Transactions of the Association for Computational Linguistics*, 5:135–146, 2017.

[8] M. Services C. for Medicare. Readmissions reduction program. `https://www.cms.gov/Medicare/Medicare-Fee-for-Service-Payment/AcuteInpatientPPS/Readmissions-Reduction-Program.html`, 2014. [Online; visitado 20-Diciembre-2018].

[9] Edward Choi, Mohammad Taha Bahadori, Elizabeth Searles, Catherine Coffey, Michael Thompson, James Bost, Javier Tejedor-Sojo, and Jimeng Sun. Multi-layer representation learning for medical concepts. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1495–1504. ACM, 2016.

[10] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint*

*arXiv:1412.3555*, 2014.

[11] Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. Fast and accurate deep network learning by exponential linear units (elus). *arXiv preprint arXiv:1511.07289*, 2015.

[12] M.P.A. Committee. Report to congress: Promoting greater efficiency in medicare. `http://www.medpac.gov/docs/default-source/reports/Jun07_EntireReport.pdf`, 2007. [Online; visitado 20-Diciembre-2018].

[13] Healthcare Cost and Utilization Project (HCUP). Clinical Classifications Software (CCS) for ICD-9-CM. `https://www.hcup-us.ahrq.gov/toolssoftware/ccs/ccs.jsp`. [Online; visitado 4-Junio-2019].

[14] Hercules Dalianis. *Clinical text mining: Secondary use of electronic patient records*. Springer, 2018.

[15] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[16] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(Jul):2121–2159, 2011.

[17] Usama Fayyad, Gregory Piatetsky-Shapiro, and Padhraic Smyth. From data mining to knowledge discovery in databases. *AI magazine*, 17(3):37, 1996.

[18] Joseph Futoma, Jonathan Morris, and Joseph Lucas. A comparison of models for predicting early hospital readmissions. *Journal of biomedical informatics*, 56:229–238, 2015.

[19] Benjamin A Goldstein, Ann Marie Navar, Michael J Pencina, and John Ioannidis. Opportunities and challenges in developing risk prediction models with electronic health records data: a systematic review. *Journal of the American Medical Informatics Association*, 24(1):198–208, 2017.

[20] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[21] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[22] Kexin Huang, Jaan Altosaar, and Rajesh Ranganath. Clinicalbert: Modeling clinical notes and predicting hospital readmission. *arXiv preprint arXiv:1904.05342*, 2019.

[23] Hernaldo Zamora Alcántara Javiera Valjalo Verdejo. Variables asociadas a la re hospitalización en dos hostpitales de Santiago de Chile. `https://www.sociedadpoliticaspublicas.cl/archivos/noveno/Funcio_Zamora_Hernaldo.pdf`, 2017. [Online; visitado 22-Octubre-2018].

[24] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[25] Durk P Kingma, Tim Salimans, and Max Welling. Variational dropout and the local reparameterization trick. In *Advances in Neural Information Processing Systems*, pages 2575–2583, 2015.

[26] David Krueger, Tegan Maharaj, János Kramár, Mohammad Pezeshki, Nicolas Ballas, Nan Rosemary Ke, Anirudh Goyal, Yoshua Bengio, Aaron Courville, and Chris Pal. Zoneout: Regularizing rnns by randomly preserving hidden activations. *arXiv preprint arXiv:1606.01305*, 2016.

[27] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436, 2015.

[28] Joshua C Mandel, David A Kreda, Kenneth D Mandl, Isaac S Kohane, and Rachel B Ramoni. Smart on fhir: a standards-based, interoperable apps platform for electronic health records. *Journal of the American Medical Informatics Association*, 23(5):899–908, 2016.

[29] Pedro Paulo Marín, Pamela Chávez, Marcela Carrasco, Homero Gac, Cristina Alonso Bouzón, and Leocadio Rodríguez Mañas. Utilización del servicio de urgencia de un hospital universitario por los adultos mayores en santiago de chile. *Revista Española de Geriatría y Gerontología*, 46(1):27–29, 2011.

[30] Andrew McCallum, Kamal Nigam, et al. A comparison of event models for naive bayes text classification. In *AAAI-98 workshop on learning for text categorization*, volume 752, pages 41–48. Citeseer, 1998.

[31] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[32] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119, 2013.

[33] Riccardo Miotto, Fei Wang, Shuang Wang, Xiaoqian Jiang, and Joel T Dudley. Deep learning for healthcare: review, opportunities and challenges. *Briefings in bioinformatics*, 2017.

[34] Tom M. Mitchell. *Machine Learning*. McGraw-Hill, 1997.

[35] Gabriel Muñoz. La ficha clínica y la protección de datos de salud en chile. *Revista Chilena de Salud Pública*, 21(1):59–67, 2017.

[36] Andrew Y Ng. Feature selection, l 1 vs. l 2 regularization, and rotational invariance. In *Proceedings of the twenty-first international conference on Machine learning*, page 78. ACM, 2004.

[37] Phuoc Nguyen, Truyen Tran, Nilmini Wickramasinghe, and Svetha Venkatesh. Deepr: A convolutional net for medical records. *IEEE journal of biomedical and health informatics*, 21(1):22–30, 2017.

[38] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks. In *International conference on machine learning*, pages 1310–1318, 2013.

[39] Jeffrey Pennington, Richard Socher, and Christopher Manning. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543, 2014.

[40] Matthew E Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. Deep contextualized word representations. *arXiv preprint arXiv:1802.05365*, 2018.

[41] Trang Pham, Truyen Tran, Dinh Phung, and Svetha Venkatesh. Deepcare: A deep dynamic memory model for predictive medicine. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pages 30–41. Springer, 2016.

[42] Alvin Rajkomar, Eyal Oren, Kai Chen, Andrew M Dai, Nissan Hajaj, Michaela Hardt, Peter J Liu, Xiaobing Liu, Jake Marcus, Mimi Sun, et al. Scalable and accurate deep learning with electronic health records. *npj Digital Medicine*, 1(1):18, 2018.

[43] Hinrich Schütze, Christopher D Manning, and Prabhakar Raghavan. Introduction to information retrieval. In *Proceedings of the international communication of association for computing machinery conference*, page 260, 2008.

[44] Benjamin Shickel, Patrick James Tighe, Azra Bihorac, and Parisa Rashidi. Deep ehr: a survey of recent advances in deep learning techniques for electronic health record (ehr) analysis. *IEEE journal of biomedical and health informatics*, 22(5):1589–1604, 2018.

[45] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

[46] University of Regina, CA. Overview of the kdd process, 2018. [Online; visitado 26-Diciembre-2018].

[47] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, pages 5998–6008, 2017.

[48] Christian Weiss, Jaime Santander, Igor Aedo, and Ximena Fuentes. Caracterización de las readmisiones precoces en la hospitalización psiquiátrica. *Revista chilena de neuro-psiquiatría*, 51(4):239–244, 2013.

[49] Cao Xiao, Edward Choi, and Jimeng Sun. Opportunities and challenges in developing deep learning models using electronic health records data: a systematic review. *Journal*

*of the American Medical Informatics Association*, 2018.

# Apéndice A

# Descripción de los datos del historial médico

A continuación describiremos cada uno de los elementos que componen el historial médico de un paciente, y detallaremos los datos que contiene cada uno. Además, especificaremos los campos que decidimos eliminar por ser ruidosos o irrelevantes, y presentaremos los procesamientos realizados según corresponda, para manejar datos erróneos, vacíos o anormales. Notación: Indicamos con una (C) los campos que son de tipo categóricos, es decir que poseen un código y una descripción correspondiente, señalizamos con una (D) los que corresponden a fechas, y con una (T) los que son texto libre. Además, tachamos los que eliminaremos.

## A.1. Datos personales

Corresponden a la información general e identificadora del paciente. Es importante notar que no utilizamos ningún tipo de reconocimiento personal (RUT, dirección, nombres, etc.).

i. Fecha nacimiento (D)
ii. Nacionalidad (C)
iii. Sexo (C)
iv. ~~Indicador si falleció (C)~~
v. ~~Fecha de fallecimiento (D)~~

## A.2. Problemas

Los problemas son diagnósticos de enfermedades que son independientes de una visita o encuentro, es decir que duran más tiempo y se deben tener en consideración en las consultas. Por ejemplo, enfermedades crónicas, dolores permanentes, enfermedades psiquiátricas, etc.

Están compuestos por los siguientes campos:

i. Vocabulario (C)

ii. Problema (C)
iii. Problema anotado (T)
iv. Fecha (D)
v. ~~Área que lo ingresa (C)~~
vi. ~~Estado (C)~~
vii. ~~Ranking (C)~~

Cada problema (ii) está descrito en algún vocabulario médico (i), los dos mayoritarios en la base de datos son: SNOMED CT (80,15 %), ICD-10-CM Chile (13,84 %). Además, el personal que ingresa la información puede escribir el problema de forma más detallada si así lo desea (iii).

No consideraremos el estado (vi), que indica si el problema está activo o no, pues, puede haber sido modificado posterior a la fecha de predicción que se está considerando, y por ende sería información futura que no se tiene en ese momento. También decidimos descartar el área que ingresa el problema (v), pues el $96,4\,\%$ corresponde a la categoría "Médico", y el resto se divide entre otras 4 categorías, por lo que no es una señal tan diferenciadora. Por último, el ranking (vii) fue descartado pues sólo un 0,6 % de los valores era distinto de nulo.

## A.3. Encuentros

Un encuentro es cualquier tipo de visita que realiza un paciente en la clínica, ya sea una consulta médica, una consulta en urgencia, retiro de resultados, una hospitalización etc.

Se compone de los campos:

i. Clase de encuentro (C)
ii. Tipo de encuentro (C)
iii. Unidad médica (C)
iv. ~~Estado (C)~~
v. Fecha de admisión (D)
vi. Fecha de alta (D)
vii. Razón de la visita (T)
viii. ~~Médico tratante (C)~~
ix. ~~Resolución~~
x. ~~Detalle resolución~~

Un encuentro está limitado a las fechas de ingreso (v) y del alta (vi) del paciente. Cuando la persona llega y realiza su admisión, describe la razón de su visita (vii) a la o el secretario que realiza el ingreso. Además, existen clases de encuentros (i) y categorías dentro de cada clase (ii). En las tablas A.1 y A.2 están las clases como encabezados de columnas y sus categorías en las columnas respectivas. Por último, una visita contiene también una unidad médica (iii), que corresponde a el área de la clínica en el cual se desarrolla el encuentro, por

ejemplo: oncología, ginecología y obstetricia, UCI, etc.

Los campos (ix) y (x) describen la situación luego de que el paciente deja el hospital, si se va a su casa, a otro centro médico o si falleció durante el encuentro. El detalle (x) en general especifica si se va en medio propio o en ambulancia, si es llevado al Servicio Médico Legal o es retirado con certificado, etc. Estos campos serán utilizados para definir el conjunto de hospitalizaciones válidas (3.2.2), pero no como datos para la predicción, pues es una información posterior a la decisión del alta.

Por último, decidimos descartar el estado (iv), pues el 0.004 % es distinto de "Dado de alta". También descartamos el médico tratante del encuentro (viii), pues se quiere medir el riesgo de hospitalización futura según el tratamiento que se le otorga y no según quién se lo otorga.

### A.3.1. Pre-procesamiento

Existen 11.276 encuentros que tienen la fecha de registro (v) o de alta vacía (vi). Pero analizando el tipo de encuentro (ii) de cada uno, notamos que todos correspondían a eventos ambulatorios o de urgencia. Comentándolo con los funcionarios del hospital, concluimos que probablemente sucedía porque la duración era muy pequeña, y por ende, decidimos igualar la vacía con la que posee un valor.

Como se dijo anteriormente, un encuentro es asociado a diagnósticos (A.4), órdenes (A.5), formularios (A.6) y notas clínicas (A.7), por lo tanto asumimos que cualquiera de ellos sucede durante el encuentro. Es decir, la fecha debe estar entre la de admisión (v) y la de alta (vi). Si esto no sucede, ignoramos la información (toda la información del diagnóstico, órden, formulario o nota clínica), pues conversando con médicos y técnicos notamos que estas situaciones corresponden a errores. En los casos en que las fechas de estos episodios estén vacías, decidimos completarlas con el promedio de las del encuentro , (v) y (vi), pues dado que está asociado a la visita entendemos que sucedió durante ella.

| **Paciente ambulatorio** | **Paciente interno** | **Preadmitir** |
|---|---|---|
| Ambulatorio | Hospitalización Transitoria | Cir/Proc Ambulatorio |
| Chequeos | Hospitalizado | Preregistro |
| Chequeos Ambulatorios | Programa Trasplante Hospitalizado | |
| Cir/Proc Ambulatorio | Quimioterapia | |
| Consulta Médica | Recién Nacido | |
| Fertilización in Vitro | | |
| Hemodiálisis | | |
| Hospitalizado | | |
| Kinesioterapia Ambulatoria | | |
| ... (más categorías) | | |

Tabla A.1: Categorías de las clases de encuentros

| Recurrente | Urgencia | null |
|---|---|---|
| Diálisis | Urgencia | Proc Ambulatorio No Invasivo<br>Resultados Ambulatorios LIS/RIS |

Tabla A.2: Continuación categorías de las clases de encuentros

## A.4. Diágnosticos

Corresponden a las enfermedades que causan o que se definen durante un encuentro. Se compone de los campos:

i. Vocabulario médico (C)
ii. Diagnóstico (C)
iii. Diagnóstico anotado (T)
iv. Fecha (D)
v. Tipo (C)
vi. Tipo de profesional que lo ingresa (C)
vii. ~~Estado (C)~~
viii. Ranking (C)
ix. Servicio clínico (C)

Al igual que un problema, los diagnósticos están descritos mediante un código (ii), que corresponde a algún vocabulario (i), entre los mayoritarios: SNOMED CT (82,3 %) y ICD-10-CM Chile (17,7 %). De igual forma, el personal puede detallar más el diagnóstico si lo desea (iii), y también poseen la fecha en que fueron ingresados (iv). Además, tienen un tipo (v), que indica si el diagnóstico es el final de egreso, el inicial con que fue admitido, o es algo que se debe tratar durante el encuentro. Los diagnósticos poseen también la información del área que ingresa la información, tanto del tipo de profesional (vi), que podrían ser: médico, enfermería, kinesiología, etc., como del tipo de servicio clínico en específico que lo ingresa (ix), como por ejemplo: centro oftalmológico, consulta de neurocirujía, quimioterapia, etc. Por último se tiene el ranking (viii) que indica la prioridad (primaria o secundaria) del diagnóstico.

Descartamos la variable de estado (vii), que indica si la enfermedad está en estudio, confirmada o descartada. Pues este estado puede ser actualizado posterior a la fecha de creación del diagnóstico, por lo que no sería información que se tendría al momento de la predicción. Además, más del 88 % está confirmado y un 7 % está indefinido, por lo que creemos tampoco posee mayor información.

## A.5. Órdenes

Son los medicamentos, procedimientos o exámenes, que se le prescriben al paciente durante un encuentro. Esto puede ser realizado por ejemplo, a través de una receta en una consulta, o de forma directa en una hospitalización. Existen dos tablas principales, una que contiene la información de lo que se receta, y otra de lo que se administra. Dado que este trabajo

de título es una primera aproximación, decidimos no utilizar por esta vez los datos de la administración de medicamentos, pues creímos no poseía tanta información la hora exacta a la que se le entrega el remedio. Con lo cual, los campos de las órdenes son:

i. Fecha (D)
ii. Unidad que lo ordena (C)
iii. Catálogo clínico (C)
iv. Tipo (C)
v. Indicación (C)
vi. Sinónimo (C)
vii. ~~Estado~~
viii. Detalle
ix. Medio de comunicación

Las órdenes son realizadas en una fecha específica (i), y por una unidad en particular (ii), tales como: urgencia, urología, obstetricia, UTI, etc. La indicación (v), se refiere a la orden en sí, el procedimiento o remedio en específico que se le receta al paciente. Además, esta indicación, es clasificada en algún tipo (iv), que a su vez es clasificado en algún catálogo (veremos algunos en ejemplos en A.3). También, la indicación posee un sinónimo (vi) y detalles (viii), que a veces corresponden al genérico del medicamento, u otras veces incluye más detalle de la dosis, de la persona que realizó el procedimiento, entre otros (tal como podemos observar en la tabla A.4). Por último, una orden es realizada a través de algún medio: verbal, escrito, protocolo, etc.

El estado de una orden indica si ya está completada, cancelada o descontinuada. Pero al igual que mencionamos en los diagnósticos, no tiene sentido incluir esta variable, pues es información que fue ingresada de forma posterior, y que podría no estar ingresada cuando se estuviera usando el modelo predictivo en la realidad.

| **Catálogo Clínico** | **Tipo** | **Indicación** |
|---|---|---|
| Medicación | Farmacia | Salbutamol |
| Indicaciones de enfermería | Cuidado de paciente | Transfundir Producto Sanguíneo |
| Laboratorio | Laboratorio | Cultivo de orina + Antibiograma |
| Medicación | Farmacia | Paracetamol |

Tabla A.3: Ejemplos de categorización de órdenes

## A.6. Formularios

Corresponden a información de distinto tipo que se ingresa a través de un formulario. Por ejemplo se pueden tener evaluaciones psicológicas o nutricionales, mediciones de signos vitales, documentación necesaria para una cirugía, test respiratorios, controles de kinesiología, entre otros.

Un formulario está compuesto por un título de tipo categórico (que indica el tipo de

| Sinónimo | Detalle |
|---|---|
| salbutamol 100 mcg/inhalacion | salbutamol 2 puff, INH, 1/día (a la hora de acostarse) |
| -Transfundir Producto Sanguíneo | Transfundir Producto Sanguíneo 21/10/2009 15:27, 2, bolsa(s), plasma |
| Cultivo de Orina + Antibiograma | Cultivo de Orina + Antibiograma Urgente tomar, Tomado Sí/No, 22/10/09 15:58 por BAGINSKY GUERRERO,MARGARITA, Fecha Finalizar 22/10/09 15:58, Tomado por la enfermera |
| Paracetamol,500mg comp SU | paracetamol 500 mg = 1 comp(s), Comp., VO, Por 1 vez, STAT, Fecha Iniciar 22/10/2009 17:18, Fecha Finalizar 22/10/2009 17:18 |

Tabla A.4: Ejemplos de sinónimos y detalles de órdenes (continuación de tabla A.3)

documento que se llenará), una fecha de creación, un estado que indica si está correcto o no, y una lista de campos y valores. Esta lista contiene cada pregunta o información a completar, y las respuestas correspondientes. Además, cada par campo-valor, posee un estado que indica si es correcta esa información o no.

A diferencia del estado en una orden, decidimos mantener los campos de estado del formulario, y de cada uno de los pares campo-valor. Pues si bien, pueden haber sido modificados posteriormente, en general indican si un valor (o el formulario completo) fue ingresado por error, si nunca se completó, o si es válido. Lo cual normalmente sucede en el momento de llenado del formulario.

Los valores pueden ser una categoría seleccionada (de una lista desplegada), uno o varios números o texto libre. Observamos algunos ejemplos a continuación en las tablas A.5 y A.6, y en la tabla A.7 presentamos un ejemplo de los distintos valores que toma un campo, de un formulario específico.

## A.7. Notas clínicas

Son los archivos de texto que genera el personal médico para escribir observaciones, informar del estado actual, y dejar por escrito la información de una consulta, una cirugía o una evaluación de algún tipo. Es importante notar que las notas clínicas son texto libre, que puede variar en estilo y uso de vocabulario entre las distintas especialidades y doctores. También, pueden contener nombres de los pacientes, sus familiares o de los tratantes, por lo que debimos realizar un pre-procesamiento para anonimizar esta información (extrajimos todos los nombres utilizando el tokenizador citado en la Sección 3.3.3). Así, están compuestas por:

i. Fecha (D)
ii. Título (C)
iii. Contenido (T)

| **Médico Genérico Urgencias SU** | |
|---|---|
| *Campos* | *Valores* |
| Ambulatorio, Escala de Evaluación | EVA |
| ED Examen Físico | |
| ED Examen Neurológico | |
| Escala Numérica del Dolor | 0=Sin Dolor |
| ESI Especialidad de la Atención | Pediatría |
| MG Evaluar Glasgow | Si |
| Presencia de dolor | No |
| Puntuación Glasgow | 15 |
| Respuesta Motora | Obedece Ordenes |
| Respuesta Ocular | Espontánea |
| Respuesta Verbal | Orientado |

Tabla A.5: Ejemplo formulario "Médico Genérico Urgencias SU"

| **Evaluación Auxiliar Enfermería SU** | |
|---|---|
| *Campos* | *Valores* |
| Circulación | Pulsos Periféricos Palpables |
| Descripción de la Respiración | Regular |
| FIO2 | 21 |
| Frecuencia Cardíaca Central | 84 |
| Neurológio Urgencia | Alerta |
| PA Diastólica | 93 |
| PA Sistólica | 124 |
| Permeabilidad de Vía Aérea | Permeable |
| Presión Arterial Media (PANI) | 103 |
| Requiere Oxígenoterapia | No |
| SpO2 | 99 |
| SU Control de Presión Arterial | Si |
| SU Control de Presión Arterial | 36.2 |

Tabla A.6: Ejemplo formulario "Evaluación Auxiliar Engenrmería SU"

| **Entrega Turno Enfermería Paciente Crítico** |
|---|
| *Clasificación del Riesgo* |
| Sin Riesgo ( >ó = 19 puntos) |
| Riesgo Moderado (13 a 14 Puntos) |
| Riesgo Bajo (15 a 18 Puntos) |
| Riesgo Alto (<ó =12 puntos) |

Tabla A.7: Ejemplo valores que toma el campo "Clasificación del Riesgo" en el formulario "Entrega Turno Enfermería Paciente Crítico"

iv. Estado (C)

Las notas clínicas entonces, están compuestas principalmente por la fecha (i) en que fueron creadas, el título (ii), su contenido en texto libre y el estado (iv), que indica si es errónea, si está actualizada o si está correcta. Al igual que el estado del formulario, el estado de una nota clínica es una validación del sistema, que indica que la nota es válida y no fue generada por error. Por lo tanto, a pesar de que puede haber sido cambiado posteriormente, normalmente es ingresado al momento de crear la nota clínica.

Las notas clínicas están almacenadas en el sistema de Cerner archivos tipo HTML o RTF. En la Clínica Las Condes nos dieron acceso a el texto plano de estos archivos, perdiéndose la información de imágenes y subrayados o negrita.