

Trabajo de Laboratorio Final:
NLP Aplicado a Curriculum Vitae
y Ofertas de Trabajo

- **Materia:** Inteligencia Artificial.
- **Carrera:** Ingeniería en Informática.
- **Docente:** Ing. Federico Gabriel D'Angiolo.
- **Estudiante:** Calonge, Federico.

Índice.-

1- Objetivo.	(Pág. 3)
2- Introducción: Conceptos previos.	(Pág. 4)
2.1-NLP.	(Pág. 4)
2.1.1-Definición.	(Pág. 4)
2.1.2-Aplicaciones.	(Pág. 5)
2.1.3-Librerías de NLP en Python.	(Pág. 5)
2.2-Conceptos NLP.	(Pág. 6)
2.2.1-Corpus.	(Pág. 6)
2.2.2-Stop y Rare Words.	(Pág. 6)
2.2.3-Tokenización.	(Pág. 6)
2.2.4-POS (Parts of Speech) Tagging	(Pág. 6)
2.2.5-Stemming o lematización	(Pág. 7)
2.3-Modelos para extraer features/vocabulario de nuestros CORPUS.	(Pág. 7)
2.3.1-Bag of Words (BOW)	(Pág. 7)
2.3.2-TF-IDF	(Pág. 8)
2.3.3-N-GRAMS y Feature Hashing.	(Pág. 8)
2.3.4- Word Embeddings.	(Pág. 9)
2.3.4.1-Funcionamiento Word2Vec para obtener nuestros Word Embeddings.	(Pág. 11)
2.4-Cómo Clasificar Texto.	(Pág. 14)
3- Desarrollo.	(Pág. 15)
3.1- Análisis de CVs de Candidatos en formato PDF y obtención de Perfiles de Candidatos.	(Pág. 15)
3.1.1-Data Set.	(Pág. 15)
3.1.2-Objetivo del análisis.	(Pág. 16)
3.1.3-Extracción de datos.	(Pág. 17)
3.1.4-Preprocesamiento de Texto.	(Pág. 18)
3.1.5- Word Embeddings e implementación del modelo Word2Vec.	(Pág. 19)
3.1.6-Resultados – Candidate Profile.	(Pág. 20)
3.2- Análisis de Ofertas/Posiciones de Trabajo y obtención de Perfiles de Candidatos.	(Pág. 21)
3.2.1-Data Set.	(Pág. 22)
3.2.2-Objetivo del análisis.	(Pág. 23)
3.2.3-Extracción de datos.	(Pág. 24)
3.2.4-Construcción de lista de Keywords predefinida.	(Pág. 24)
3.2.5-Tratamientos a la columna 'job description'.	(Pág. 26)
3.2.6-Resultados para Data Scientists.	(Pág. 27)
3.2.7-Resultados para los demás Job Posts – Candidate Profile V2.	(Pág. 29)
4-Conclusiones.	(Pág. 30)
5-Mejoras.	(Pág. 32)
6-Bibliografía.	(Pág. 33)

1- Objetivo.

Este Trabajo de Laboratorio consistirá, como un objetivo secundario, introducirnos en el mundo de NLP (Procesamiento de Lenguaje Natural o Natural Language Processing) para explicar en qué consiste el mismo, conceptos, aplicaciones y los distintos algoritmos y modelos implicados en esto.

Y, como objetivo principal, realizaremos dos análisis:

1. **Análisis de Curriculum Vitae / CVs / Resumes de Candidatos en formato PDF y obtención de Perfiles de Candidatos.**
2. **Análisis de Ofertas/Posiciones de Trabajo y obtención de Perfiles de Candidatos.**

En el *1er análisis* le proporcionaremos al sistema distintas keywords que creemos que son relevantes para analizar (statistics, Language, machine_learning, Deep, Python, data) y el sistema calculará las palabras más similares a dichas Keywords. Estas palabras similares se obtendrán mediante la **obtención de word embeddings** mediante el modelo **Word2Vec** usando como entrenamiento de un **Corpus** de texto obtenido de artículos de Wikipedia que hablan sobre Machine Learning y temas similares. Luego de obtener dichas palabras similares a las Keywords, se machearán las mismas con los CVs en PDF y así obtendremos un **Perfil** de Candidato.

En cambio, en el *2do análisis* no utilizaremos ningún modelo para entrenar nuestros datos (como Word2Vec). Simplemente consiste en analizar Ofertas de Trabajo (nuestro **Corpus**) publicadas en Indeed al día de la fecha (06/07/2020) para distintas posiciones (Data Scientist, Machine Learning, Data Engineer, Java Programmer, HCM Consultant) mediante técnicas de NLP (tokenización, stemming/lematización, POS Tagging, etc.). Luego, teniendo una lista de Keywords predefinidas de Skills y Tools requeridas mediante consideraciones propias, el objetivo de esto es:

- Por un lado, analizar cuáles son las 20 principales skills y tools mediante visualizaciones, y cuál es el nivel de educación mínimo para los puestos publicados como “Data Scientist”;
- y por el otro lado, obtener un TOP 5 de Skills y Tools para las demás posiciones (Machine Learning, Data Engineer, Java Programmer, HCM Consultant). Estas Skills y Tools las utilizaremos como nuevas Keywords para machear directamente contra los CVs de los Candidatos; y así obtener, como en el análisis 1, un Perfil de Candidato; pero esta vez será en base a Keywords predefinidas y NO en base a palabras similares a keywords obtenidas mediante Word2Vec.

2- Introducción: Conceptos previos.

En esta Sección describiremos los temas teóricos y matemáticos para tratarlos a lo largo del desarrollo del Informe y llevarlos a cabo mediante los Algoritmos de Python en el ambiente de Anaconda.

2.1-NLP.

2.1.1-Definición.

NLP (Natural Language Processing o Procesamiento del Lenguaje Natural) es un campo de las ciencias de la computación, inteligencia artificial y lingüística que estudia las interacciones entre las computadoras y el lenguaje humano. Su **objetivo** principal es **desarrollar algoritmos que permitan a los sistemas/ aplicaciones/ máquinas comprender textos y comprender el lenguaje natural/humano**.

Para abordar este problema, se comenzaron a desarrollar diversas aplicaciones utilizando enfoques de aprendizaje automático. Este objetivo se puede lograr **recolectando una gran cantidad de texto y luego entrenando al algoritmo para realizar diversas tareas como categorizar texto y/o modelar temas**.

Gracias a los millones de GB y TB diarios generados por blogs, sitios web sociales y páginas web hoy en día es muy beneficioso e interesante usar NLP. Hay muchas empresas que reúnen todos estos datos para comprender a los usuarios y sus pasiones y dar estos informes a las empresas para ajustar sus planes.

Estos datos podrían mostrar que la gente de Brasil está contenta con el producto A, que podría ser una película o cualquier cosa, mientras que la gente de EE. UU. está contenta con el producto B. Y esto podría ser instantáneo (resultado en tiempo real). Al igual que los motores de búsqueda, dan los resultados adecuados a las personas adecuadas en el momento adecuado.

Podemos resolver muchísimos problemas con NPL, tales como:

- **Clasificación de textos** (puede ser en muchas categorías o binaria). Un ejemplo de clasificación binaria es hacer **análisis de sentimiento** (decir por ej. si un review es positivo -1- o negativo -0-).
- **Entity Recognition**: esto es reconocer **entidades** en los textos y **tagearlas** (por ejemplo personas, empresas, fechas, lugares, etc.).
- **Topic modeling**: de esta manera podemos entender cuáles son los temas principales de un texto y extraerlos.
- Y muchísimos más.

2.1.2-Aplicaciones.

Podemos encontrar muchísimas aplicaciones dentro del mundo de NLP:

- **Motores de búsqueda:** como Google.
- **Feeds de los sitios web sociales:** como la fuente de noticias de Facebook. El algoritmo de feed de noticias entiende tus intereses mediante NLP y muestra anuncios y publicaciones relacionadas con mayor probabilidad que otras publicaciones.
- **Interfaces de conversación (reconocimiento de voz y traducción del habla):** por ejemplo el motor de voz de Apple, Siri.
- **Procesadores de documentos:** NLP permite comprender oraciones completas y hasta incluso escribir oraciones y párrafos completos gramaticalmente correctos.
- **Filtros de Spam:** por ejemplo los filtros de spam de Gmail: No se trata solo del filtro de spam habitual, ahora los filtros de spam entienden qué hay dentro del contenido del correo electrónico y ver si es un correo no deseado o no.
- **Traducción de textos automática.**
- **Resúmenes automáticos:** dado un texto nos devuelve el resumen condensado.
- **Modelos de recomendación.**
- Y muchísimas más.

2.1.3-Librerías de NPL en Python.

Las librerías de NPL más populares para Python son:

- Natural language toolkit (NLTK): es la biblioteca más de Python más popular y fácil de usar para el procesamiento del lenguaje natural (NLP) → <https://www.nltk.org/>
- Spacy: librería NLP para Python más rápida que NLTK.
- Gensim: librería de Python que nos provee acceso al modelo Word2Vect y otros algoritmos para entrenar y obtener nuestros word embeddings. También nos provee de word embeddings ya pre-entrenados para descargarlos y utilizarlos (los cuales están entrenados con las noticias de Google).
- Apache OpenNLP.
- Stanford NLP suite.
- Gate NLP library.

Estas librerías nos proveen distintas funciones para poder interactuar con el lenguaje humano: tokenizar o lematizar palabras, buscar relaciones entre palabras más similares, buscar sinónimos, antónimos, etc. Para algunas de estas funciones se utilizan modelos de redes neuronales implementados en dichas librerías. En nuestros análisis utilizamos las 3 primeras librerías (NLTK, Spacy y Gensim).

2.2-Conceptos NLP.

A continuación explicaremos varios conceptos de NLP que tomaremos en cuenta para en análisis de nuestros textos en Python.

2.2.1-Corpus.

También llamado “Corpus Lingüístico”, es un conjunto amplio y estructurado de ejemplos reales de uso de la lengua. En nuestro caso utilizamos textos extraídos de Wikipedia (para el análisis 1) y job posts extraídos de Indeed.com (para el análisis 2). Estos corpus deben ser un **conjunto de textos que deben ser relativamente grande**, creado independientemente de sus posibles formas o uso... osea que la estructura, variedad y complejidad del corpus debe reflejar dicha lengua de la forma más exacta posible. **La idea es que representen al lenguaje de la mejor forma posible para que los modelos de NLP puedan aprender los patrones necesarios para entender el lenguaje.**

2.2.2-Stop y Rare Words.

- Stop words: palabras muy frecuentes que tenemos que decidir qué hacer con estas (hay listas de stop words precompiladas que nos dice cuáles son las mismas para un determinado idioma). Por ejemplo: “the”, “a”, “this” en inglés.
- Rare words: palabras “raras” que no aparecen en cinco o más documentos por ejemplo. Estas podríamos querer filtrarlas también como las stop words.

2.2.3-Tokenización.

Cuando tratamos con texto, necesitamos dividirlo en partes más pequeñas para su análisis. Por esto, **al tokenizar**, lo que estamos haciendo es separando el texto de entrada en entidades más pequeñas o palabras, llamadas *tokens*, con las que trabajaremos luego. En un tokenizador debemos evaluar muchas cosas: qué hacemos con los signos de puntuación, guiones, underscores, fechas, si utilizamos los signos de puntuación como token o no, si le damos o no importancia a las mayúsculas y si unificamos o no palabras similares en un mismo token.

Por esto, usar un buen tokenizador es muy importante. No es recomendable hacer nuestros propios tokenizadores, ya hay muchos tokenizadores que funcionan bastante bien y sirven para cualquier idioma.

Para ambos análisis utilizamos el tokenizador provisto por la librería NLTK (“word_tokenize”).

2.2.4-POS (Parts of Speech) Tagging

POS Tagging es un método de NLP que consiste en ETIQUETAR/TAGEAR a las palabras en sustantivos (noun), adjetivos (adjectives), verbos (verbs), etc. Lo utilizamos en el 2do análisis en los “job descriptions” para identificar algunos tags y así filtrar las palabras que NO tengan dichos tags; ya que no son informativas para nuestro análisis.

2.2.5-Stemming o lematización

Word Stemming es un método de NLP que nos permite convertir palabras en una raíz común (o también llamado "root"/"base"/"stem"). Por ejemplo las palabras "models" y "modeling" provienen/tienen la misma raíz ("model"). De esta manera, podemos considerar dichas palabras (models y modeling) como una sola... como la raíz (model). Aplicaremos esta técnica en el 2do análisis en los "job descriptions".

2.3-Modelos para extraer features/vocabulario de nuestros CORPUS.

Podemos usar muchos modelos que utilizan distintos algoritmos/métodos para extraer features de nuestros textos; y así tener un vocabulario que nos sirva para resolver distintos problemas. De esta manera, **los modelos permiten tomar un input de texto y obtener un output numérico de salida**, por ejemplo, nos permiten identificar palabras más similares entre sí o poder predecir qué palabra colocar luego de otra en base al contexto.

Para todos los modelos el resultado final es un VECTOR que representa nuestro texto, el cual lo podremos usar luego en un modelo de ML.

En nuestro caso, solo utilizamos estos modelos para el análisis 1, donde **obtendremos un Word embedding mediante el entrenamiento del modelo Word2Vec** con nuestros textos obtenidos de Wikipedia para obtener palabras similares a nuestras keywords. Para el análisis 2 no aplicamos ninguno de estos modelos, simplemente utilizamos técnicas de NLP (tal como aplicar tokenización, POS (Parts of Speech) Tagging y Stemming).

Como dijimos, para el 1er análisis el modelo que utilizaremos es el conocido como 'Word embedding', pero hay muchísimos más, los cuales veremos a continuación.

Para todos los modelos que nombraremos, debemos tener en cuenta el texto que ingresaremos al mismo y tomar distintas decisiones de cómo indexar y qué hacer con nuestras palabras obtenidas del texto: cómo tokenizar el texto, si tenemos o no en cuenta las Stop y Rare Words, si aplicamos o no POS Tagging y Stemming, etc.

2.3.1-Bag of Words (BOW)

Vamos a ver nuestra primera aproximación a cómo extraer features de un Corpus de texto mediante el modelo BoW. BoW es uno de los modelos más simples, nos permite extraer un vocabulario/léxico de nuestro texto. Vocabulario = todos los términos que intervienen en el texto.

Luego de tener este vocabulario lo que vamos a hacer es crear por cada texto un **vector binario** de un tamaño muy grande, donde cada componente del vector va a ser un 0 o un 1 según aparezca la palabra en el texto. De esta manera nuestro texto estará representado por un vector binario.

El principal problema de BoW es que si tenemos miles de términos entonces nuestro vector tendrá miles de columnas. Pero la ventaja es su sencillez. Este modelo de BoW es usable siempre y cuando tengamos una cantidad de columnas manejable.

2.3.2-TF-IDF

Utilizar TF-IDF es el primer “refinamiento” / variante que se nos puede ocurrir aplicar sobre BoW.

- TF = term frequency. Fórmula para TF \rightarrow **ftd** (frecuencia del término en el documento \rightarrow cuántas veces aparece). Hay otras fórmulas que implican aplicar log a ftd o aplicar log a ftd.
- IDF = inverse document frequency. Fórmula para IDF \rightarrow **$\log (N / ft)$** \rightarrow logaritmo de la cantidad de documentos “N” / la cantidad de documentos en donde aparece el término (“ft”).

Entonces ahora, a nuestro vector de BoW lo que hacemos es que en vez de que esté formado por 1s y 0s... que esté formado por:

- Si el término está en el documento el valor será de TF x IDF.
- Y si no está va a ser 0.

2.3.3 -N-GRAMS y Feature Hashing.

El modelo de N-gramas consiste en contabilizar las frecuencias de combinaciones de N palabras. Con N=1 se trata simplemente de contar las frecuencias de las palabras tal y como hicimos antes. Con N=2 contamos las frecuencias de las palabras de acuerdo a la palabra anterior, es decir usando una palabra como contexto. Con N=3 usamos dos palabras como contexto y así sucesivamente.

Al modelar N-GRAMAS, ahora en nuestro vector no solo vamos a tener UNI-GRAMAS (1-GRAMS) también vamos a tener BI-GRAMAS (2-GRAMS), TRI-GRAMAS (3-GRAMS) y demás. Por ejemplo, para el texto “The quick brown fox jumped over the lazy dog” los bi-gramas son los siguientes:

- The quick.
- Quick Brown.
- Brown fox.
- Fox jumped
- Jumped over
- Over the
- The lazy
- Lazy dog.

Este vector de N-gramas puede ser MUY largo, entonces podemos elegir cuales son los N-grams que vamos a utilizar ahí. De esta manera podemos excluir los más raros que aparecen en muy pocos textos y excluyendo los más populares porque NO nos interesan.

Podríamos aplicar una mejora a esto mediante **Feature hashing**. De esta manera, lo que hacemos es fijar una cantidad de K columnas fija. Entonces, tanto los unigramas como los bi-gramas, tri-gramas o lo que queramos utilizar... los vamos a HASHEAR en un número entre 0 y K-1. Con este esquema podemos solucionar el problema que teníamos de tener un vector muy largo. Y de esta manera, en un vector de tamaño fijo podemos representar n-gramas en una forma más compacta.

2.3.4- Word Embeddings.

Es una de las variantes más populares para representar textos y extraer features a través de nuestros textos. Un embedding es un VECTOR que representa a una palabra. Si nuestro texto tiene N palabras, entonces tendremos N vectores embeddings para representar a nuestro texto. Estos vectores son **DENSOS** (NO son dispersos como teníamos en los modelos basados en BoW).

Estos word embeddings son una de las representaciones de un documento de vocabulario más popular. Son vectores "one-hot vector" y son previamente entrenados; de esta manera son capaces de capturar los contextos de una palabra en el documento, similitud semántica y sintáctica, relaciones con otras palabras, etc.

-Para obtener/construir estos vectores embeddings hay muchas alternativas... las 3 más populares hoy en día son estos modelos:

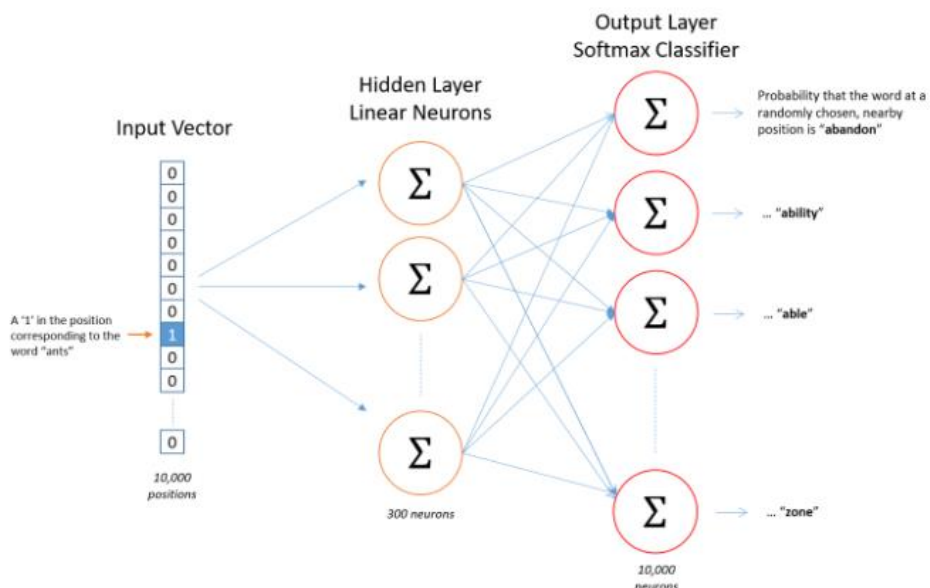
- **Word2Vec.**
- Glove.
- Starspace.

→ En estas 3 alternativas estos modelos de embeddings se ENTRENAN mediante ANNs. Hay 2 formas para entrenar y obtener estos modelos:

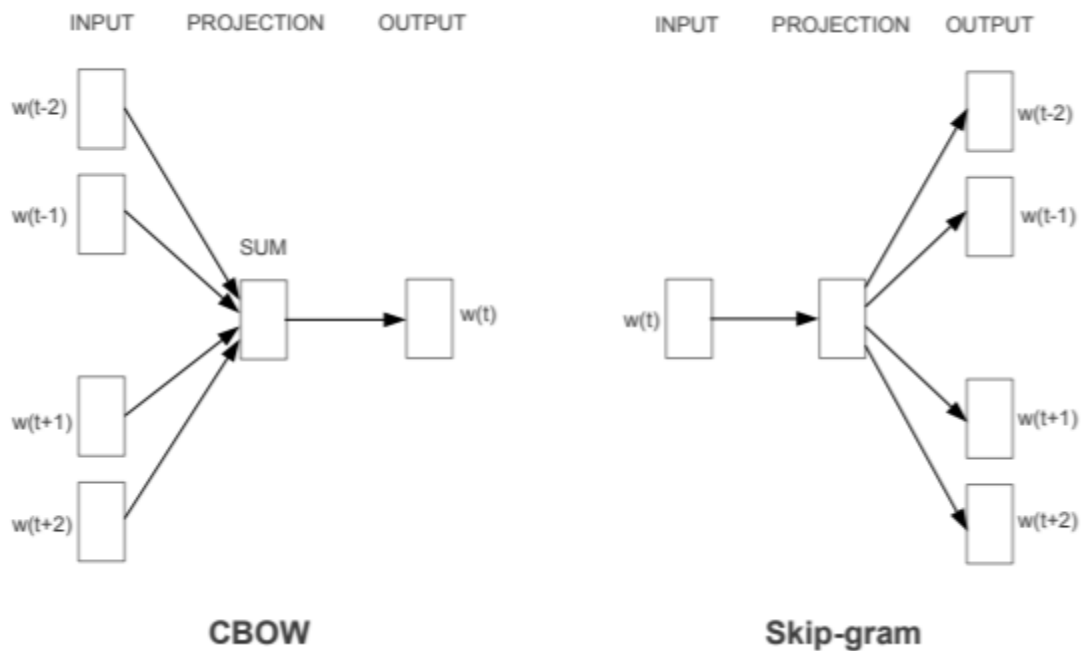
1-CBOW (Continuous Bag Of Words): Se trata de predecir la **palabra target** de un texto **en base** a las palabras más cercanas/vecinas (en base al **contexto**).

2-Skipgrams: **Opuesto a CBOW**. En base a la palabra se trata de predecir cuales son los vecinos (contexto)

→ De esta manera con 1 o 2 entrenamos los distintos modelos; y en definitiva vamos a obtener un embedding por cada palabra.



Modelo de Red Neuronal utilizado para entrenar embeddings



Entrenamiento mediante CBOW vs Skip-gram



CBOW – Predecimos la palabra “target” en base al contexto



Skip Gram – Predecimos el contexto en base a la palabra “target”.

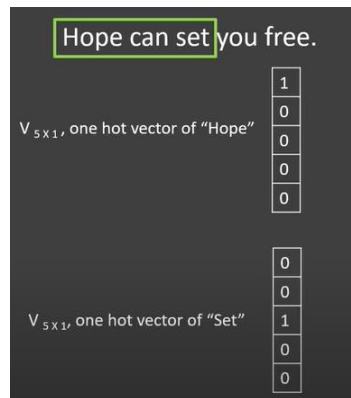
2.3.4.1-Funcionamiento Word2Vec para obtener nuestros Word Embeddings.

En nuestro 1er análisis utilizaremos **CBOW** como entrenamiento de nuestro modelo Word2Vec para obtener nuestros Word Embeddings. Pero, como vimos, también está el entrenamiento mediante **Skip Grams**. Veamos un ejemplo de cómo obtener nuestros Word Embeddings:

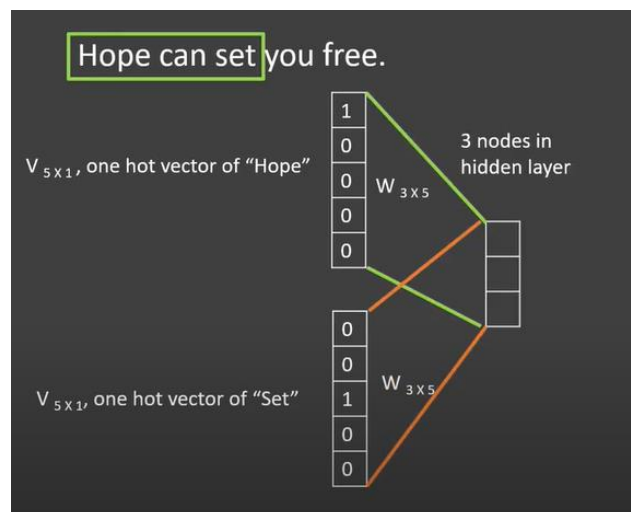
- **Mediante CBOW:**

Consideremos que tenemos un vocabulario simple (“**Hope can set you free**”) con el que entrenaremos nuestra red neuronal (propia del modelo Word2Vec).

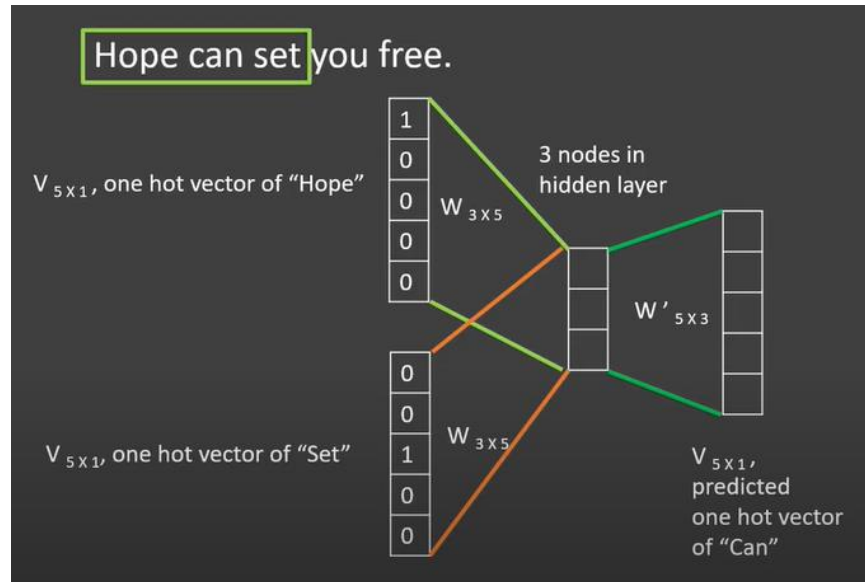
1. Pasamos nuestro vocabulario a un “one hot vector” (donde tendrá un 1 en la posición que tiene la palabra en el texto y un 0 en las demás posiciones; y el tamaño del vector será igual al número de palabras de nuestro lenguaje -nuestro vocabulario-). Luego de esto seleccionamos un tamaño de ventana (en nuestro caso seleccionaremos 3). Como en CBOW tratamos de predecir la palabra en base al contexto, entonces usamos como entrada “hope” y “set” (contexto) para predecir la palabra target del medio (“can”).



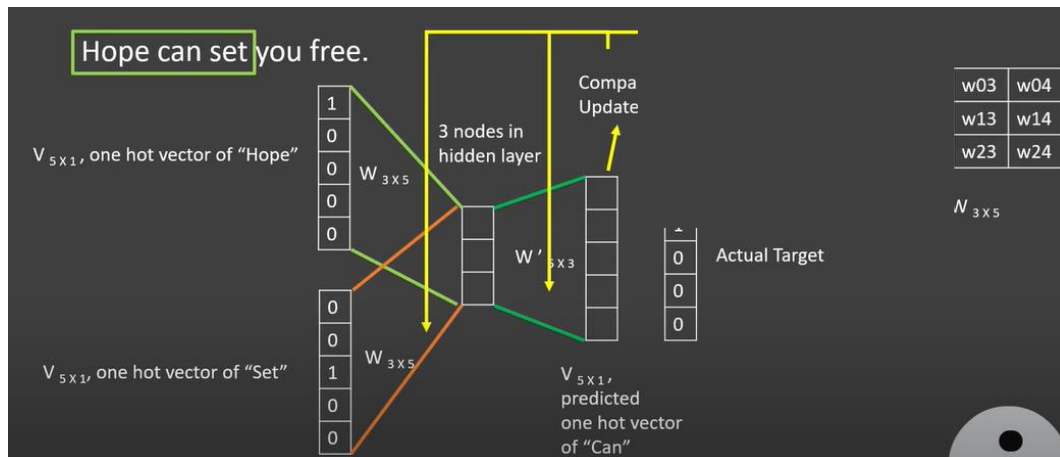
2. Colocamos los vectores ‘one hot’ de “Hope” y de “Set” como entradas a nuestra red neuronal, la cual tendrá el 3 nodos en la capa oculta.



3. Entonces nuestra red neuronal tratará de predecir la palabra “can”. Estos valores pasarán por la función de la capa de salida SOFTMAX para obtener así probabilidades.



4. Dichas probabilidades obtenidas las comparamos con el vector "actual target" (que son los valores del vector "can") y en base a esto actualizamos los pesos de nuestra red. Y luego de realizar iteraciones/epochs (otro parámetro a considerar) nuestra red neuronal tendrá una matriz con nuevos pesos para la 1ra iteración de la ventana de tamaño 3.



5. Luego continuamos con la 2da posición, hacemos lo mismo que vimos previamente, y luego lo mismo para la 3ra posición.

Hope can set you free.

2da posición

Hope can set you free.

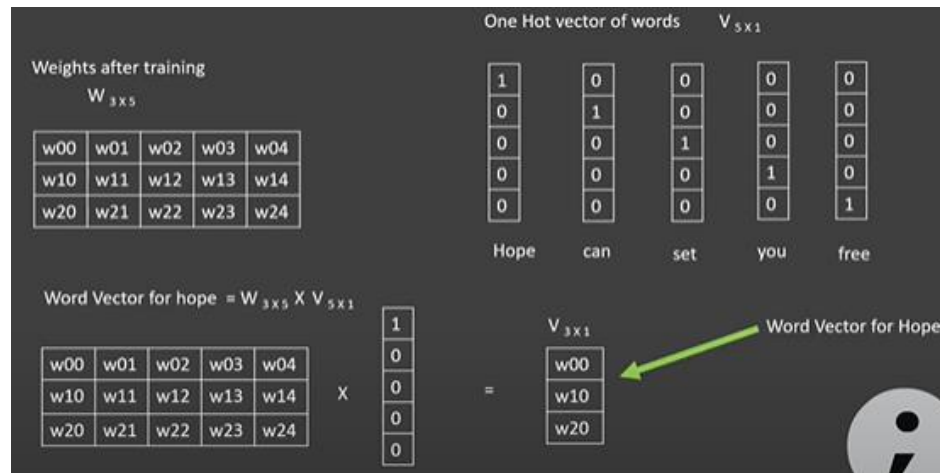
3ra posición

6. Y por último obtenemos esta matriz con los pesos de nuestra red neuronal luego del entrenamiento.

w00	w01	w02	w03	w04
w10	w11	w12	w13	w14
w20	w21	w22	w23	w24

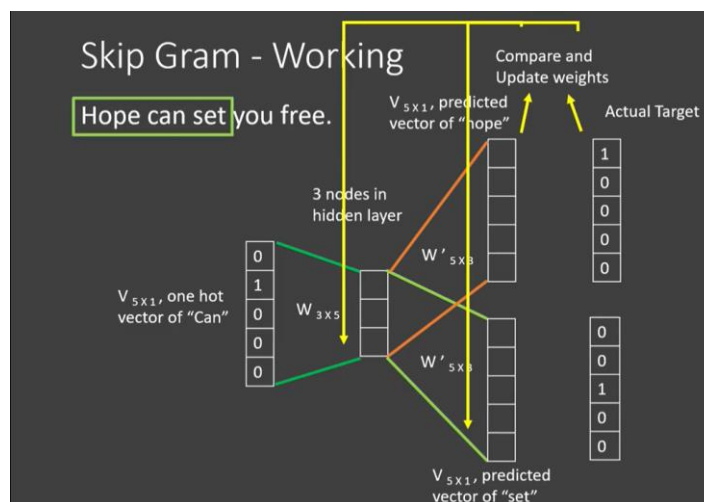
$W_{3 \times 5}$

7. Luego, para obtener finalmente nuestros word embeddings para cada palabra, lo que hacemos es multiplicar la matriz obtenida en 6 por el vector 'one hot' asociado a dicha palabra. En el caso de abajo observamos nuestro word embedding vector para la palabra "Hope".



- **Mediante Skip Gram:**

En este caso, a diferencia de CBOW, tratamos de predecir el contexto en base a la palabra target. Por esto, nuestra entrada será el vector 'one hot' can y en base a esto prediciremos las palabras 'hope' y 'set'. Pero **nuestro resultado será el mismo** (matriz con los pesos asociados a nuestra red neuronal del paso 6- en CBOW).



- En aplicaciones reales generalmente entrenamos nuestro texto con millones de palabras.

2.4-Cómo Clasificar Texto.

NO es nuestro caso, pero si quisiéramos CLASIFICAR texto, lo que podemos hacer es:

1. Con nuestro corpus de texto ya pasado a n Vectores Embedding (1 por cada palabra de nuestro texto), podemos transformar estos N-embeddings en 1 solo mediante una **vectorización de embeddings**. Hay 2 opciones para esto:
 - Promediar (AVERAGE) los embeddings y **quedarnos con un único embedding que represente todo nuestro texto.**
 - Usamos “**Doc2Vec**” o “**Sentence2Vec**” que también nos va a procesar todas las palabras y **nos va a dar un único embedding.**

La diferencia de esta representación con la que se puede llegar a obtener de una representación de BoW, es que a diferencia de esta nos generará un vector de embedding mucho más compacto (de menos dimensiones que BoW).

2. Luego de esto, teniendo ya vectorizados con embeddings nuestros textos (o sea nuestro Corpus representado con 1 solo vector de embeddings) podemos utilizar distintos modelos de Deep Learning para procesar el texto en lenguaje natural y obtener dicha clasificación. Para esto tenemos distintas opciones:
 - Redes Neuronales Convolucionales en 1 Dimensión (“**CONV 1D**”): permite aplicar una serie de N filtros con un cierto tamaño K de palabras. Aplica dichos filtros a todo el vocabulario, obtiene max poolings y a cada resultado le puede aplicar más convoluciones. De esta manera, se realiza una convolución de convoluciones de filtros. De esta manera se pueden reconocer si los textos tenían bi-gramas o tri-gramas, etc. y se empiezan a construir features más complejos en base a los textos. **De la misma forma que una CNN sobre imágenes iba trabajando primero con bordes, luego con formas y demás... esto va trabajando primero con bigramas, luego con trigramas para entender semánticamente sentencias y demás, hasta llegar finalmente a una capa fully connected y finalmente a una SOFTMAX que nos dice a qué Clase pertenece nuestro texto.**
 - CHAR-CONV 1D → es lo mismo que CONV 1D solo que en vez de hacer los embeddings one hot a nivel de palabra se hacen a nivel de carácter. De esta manera tenemos un vector “a” por ej. de 100 posiciones (si hay 100 caracteres posibles) que tiene todos 0s y un 1 en alguna posición.
 - Bidireccional LSTM (B+- LSTM).
 - Modelos de atención (Attention Models).
 - RNNs.
 - CHAR-RNNs.
 - LSTM.

La variante que funcione mejor depende del tipo de problema, del texto y del procesamiento que queramos hacer con ellos.

3- Desarrollo.

3.1- Análisis de CVs de Candidatos en formato PDF y obtención de Perfiles de Candidatos.

En el *1er análisis* le proporcionaremos al sistema distintas keywords que creemos que son relevantes para analizar (statistics, Language, machine_learning, Deep, Python, data) y el sistema calculará las palabras más similares a dichas Keywords. Estas palabras similares se obtendrán mediante la **obtención de word embeddings** mediante el modelo **Word2Vec** usando como entrenamiento de un **Corpus** de texto obtenido de artículos de Wikipedia que hablan sobre Machine Learning y temas similares. Luego de obtener dichas palabras similares a las Keywords, se machearán las mismas con los CVs en PDF y así obtendremos un **Perfil** de Candidato.

3.1.1-Data Set.

Para el *1er análisis*, dispondremos de 2 dataset con los cuales demostraremos el funcionamiento de NLP en Python con el ambiente Anaconda. Estos 2 dataset son:

- Dataset de CVs en PDF. Estos datos NO los usamos para entrenar, simplemente los usamos para machear con las keywords ya previamente “filtradas” por un modelo entrenado. Son nuestros ejemplos de prueba con la información de nuestros Candidatos y están en formato PDF. Se disponen de 6 CVs. Se muestra un ejemplo en la **Imagen 1**.
- Dataset de Artículos de Wikipedia: Corpus de texto extraído mediante Sketch Engine de distintos artículos de Wikipedia, el cual lo utiliza nuestro modelo para entrenar y devolvernos las palabras más similares a keywords definidas por nosotros. Ver **Imagen 2**.

Meghna Lohani		
Campus Address LHA, VIT Chennai Chennai, Tamil Nadu	Meghna.Lohani2016@vitstudent.ac.in Cell-8511192673	Permanent Address F-203, Shangri La Apartments Vadodara
OBJECTIVE	To apply my technical skills for the growth of the company	
EDUCATION	Vellore Institute of Technology, Chennai (T.N.) Bachelor of Computing Science and Engineering CGPA: 9.24/10.0 (Till V semester), to be awarded May 2020 St. Patrick's Junior College, Agra HSC (Class XII), ICSE Percentage: 97, awarded May 2015 St. Patrick's Junior College, Agra SSC (Class X), ICSE Percentage: 95 awarded May 2013	
DATE OF BIRTH	12-May-1998	
LANGUAGES KNOWN	Hindi and English	
COURSEWORK	Object Oriented Programming Database Management Systems Data Structures and Algorithms Discrete Mathematics Machine Learning	Digital Logic and design Software Engineering Computer Architecture Statistics for Engineer
COMPUTER SKILLS	Languages: Python, C, C++, R (Beginner), Java Skills: HTML, PHP, JavaScript, SQL, Data Structures, Data Science, Machine learning, Android	
EXPERIENCE	Developer at VIT Technovit Website Team Won two hackathons Co-ordinator of the android club	

Imagen 1 - Ejemplo de CV

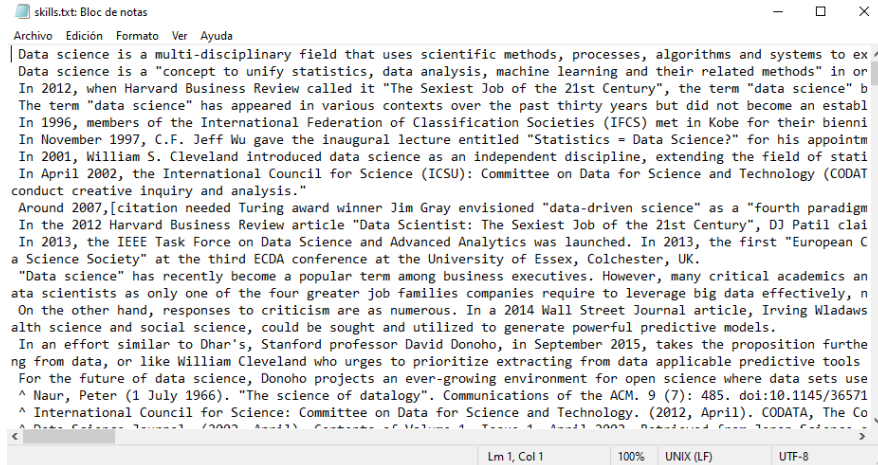


Imagen 2 – Dataset de Artículos de Wikipedia relacionados a Machine Learning.

3.1.2-Objetivo del análisis.

El objetivo en sí del sistema es obtener una visualización de perfiles de candidatos.

El **Workflow** que utilizamos para llevar esto a cabo en Python es el siguiente:

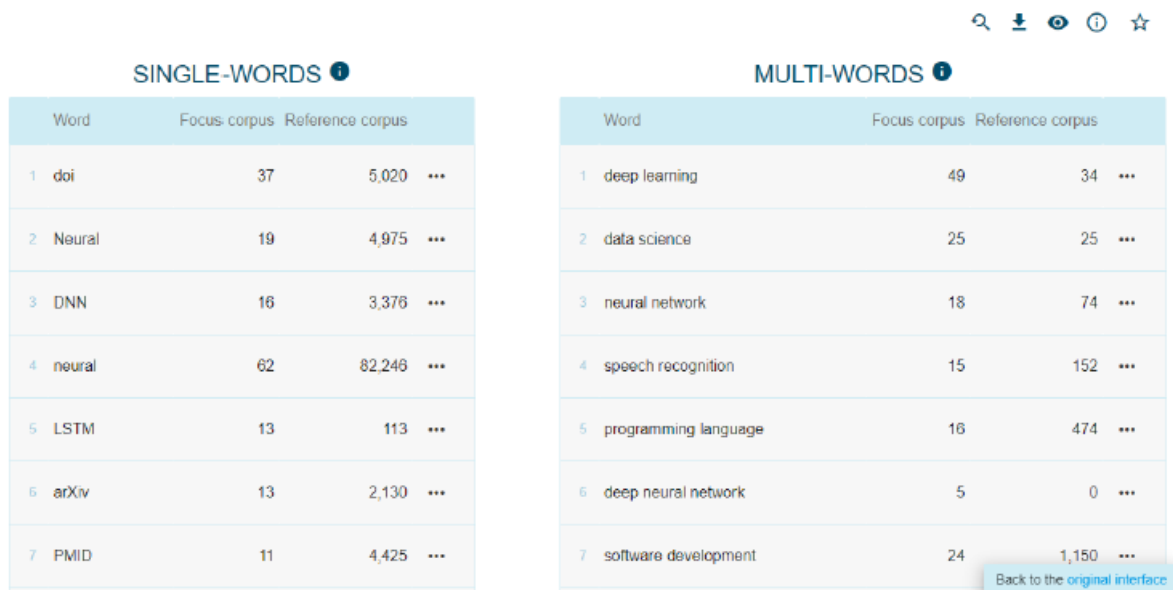
1. Creamos el 'CORPUS' mediante Sketch Engine.
2. Preprocesamiento.
3. Removemos las palabras comunes que no son requeridas.
4. Creamos frases de 'Bigrams' (2 palabras que frecuentemente ocurren juntas).
5. Creamos 'word embedding' mediante nuestro modelo 'word2vec' mediante la librería Genism
6. Extraemos los CVs/Resumes usando PyPDF y los convertimos en string.
7. Construimos el perfil del candidato (candidate profile) usando 'model.most_similar(skills)' (donde 'skills' es un array de los skills requeridos). Y luego creamos un 'Matcher' usando Spacy para machear las palabras en el CV con 'most_similar(skills)'.
8. Contamos las palabras para cada categoría e imprimimos nuestro 'candidate profile'.
9. Visualizamos el candidate profile con matplotlib.

***Archivo con el código: CV_Scoring_V1.ipynb**

3.1.3-Extracción de datos.

Utilizaremos Sketch Engine (<https://www.sketchengine.eu/>) para obtener nuestro Corpus de texto con artículos Web de Wikipedia. De esta manera, obtendremos nuestro dataset de los siguientes artículos:

- https://en.wikipedia.org/wiki/Machine_learning
- https://en.wikipedia.org/wiki/Deep_learning
- https://en.wikipedia.org/wiki/Data_science
- <https://en.wikipedia.org/wiki/Programmer>
- https://en.wikipedia.org/wiki/Software_engineer
- https://en.wikipedia.org/wiki/Software_development



SINGLE-WORDS ⓘ			
	Word	Focus corpus	Reference corpus
1	doi	37	5,020 ...
2	Neural	19	4,975 ...
3	DNN	16	3,376 ...
4	neural	62	82,246 ...
5	LSTM	13	113 ...
6	arXiv	13	2,130 ...
7	PMID	11	4,425 ...

MULTI-WORDS ⓘ			
	Word	Focus corpus	Reference corpus
1	deep learning	49	34 ...
2	data science	25	25 ...
3	neural network	18	74 ...
4	speech recognition	15	152 ...
5	programming language	16	474 ...
6	deep neural network	5	0 ...
7	software development	24	1,150 ...

Back to the original interface

Imagen 3 – Sketch Engine

3.1.4-Preprocesamiento de Texto.

Leemos el dataset “skills.txt” con todo nuestro Corpus de textos, tokenizamos y sacamos las Stop Words. Además, removemos palabras comunes que no son requeridas (estas palabras las colocamos en un archivo “commons.txt” donde colocamos palabras comunes que NO son necesarias para el macheo de skills técnicos; son palabras que eliminaremos de las palabras tokenizadas... por ejemplo ‘may’ o ‘need’.

De esta manera pasamos de tener esto (por ejemplo para el párrafo en la posición 28):

```
content[28]
```

```
'In deep learning, each level learns to transform its input data into a slightly more abstract and composite representation. In an image recognition application, the raw input may be a matrix of pixels; the first representational layer may abstract the pixels and encode edges; the second layer may compose and encode arrangements of edges; the third layer may encode a nose and eyes; and the fourth layer may recognize that the image contains a face. Importantly, a deep learning process can learn which features to optimally place in which level on its own. (Of course, this does not completely obviate the need for hand-tuning; for example, varying numbers of layers and layer sizes can provide different degrees of abstraction.)'
```

A tener esto:

```
['deep', 'learning', 'level', 'learns', 'transform', 'input', 'data', 'slightly', 'abstract', 'composite', 'representation', 'image', 'recognition', 'application', 'raw', 'input', 'matrix', 'pixels', 'first', 'representational', 'layer', 'abstract', 'pixels', 'encode', 'edges', 'second', 'layer', 'compose', 'encode', 'arrangements', 'edges', 'third', 'layer', 'encode', 'nose', 'eyes', 'fourth', 'layer', 'recognize', 'image', 'contains', 'face', 'importantly', 'deep', 'learning', 'process', 'learn', 'features', 'optimally', 'place', 'level', 'course', 'completely', 'obviate', 'handtuning', 'example', 'varying', 'numbers', 'layers', 'layer', 'sizes', 'provide', 'different', 'degrees', 'abstraction']
```

Luego, lo que hacemos es **unir palabras bigramas** como: machine_learning, big_Data deep_learning (Que antes estaban separadas pero que en realidad van juntas; así, objeto Phraser detecta esto y te lo devuelve como 1 sola palabra).

3.1.5- Word Embeddings e implementación del modelo Word2Vec.

La librería Gensim nos provee una simple API al algoritmo de Google 'word2vec' el cual es usado para crear word embeddings. Como vimos previamente, word2vec son modelos de redes neuronales superficiales de dos capas que están entrenadas para reconstruir contextos lingüísticos de palabras.

Podemos traer word embeddings pre-entrenados con noticias de Google por ejemplo, donde se entrenarán con 3 millones de palabras. Pero, nosotros lo que hacemos es utilizar nuestro vocabulario para entrenar y crear nuestros propios embeddings.

La entrada de la red neuronal es una palabra representada como un vector “one-hot”, es decir, un vector con tantas posiciones como tamaño tenga el vocabulario.

Por ejemplo, si queremos representar la palabra “sol” de un vocabulario de 4 palabras (el sol está cayendo), usaremos un vector de esta dimensión (5) con cero en todas las posiciones menos la correspondiente a la palabra “sol” que tendrá un uno → [0,1,0,0]

Y la **salida de la red neuronal** será otro vector “one-hot” llamado “**Word embedding**” de las mismas 4 posiciones que representará las probabilidades de cada una de las palabras sean vecinas de la palabra representada en la entrada. [0.12, 1, 0.4, 0.1]-> Es 1 en la 2da posición ya que es la misma posición de la palabra.

Word embedding es una de las representaciones de un documento de vocabulario más popular. Es capaz de capturar los contextos de una palabra en el documento, similaridad semántica y sintáctica, relaciones con otras palabras, etc.

Mediante la siguiente línea de código entrenamos al modelo Word2Vec mediante nuestras 1275 palabras contenidas en “all Sentences”:

```
model=gensim.models.Word2Vec(all_sentences,size=5000,min_count=2,workers=4>window=4)
```

Además, como no especificamos el parámetro “sg” entonces se entrenará mediante el método de entrenamiento CBOV. Y workers son los epochs/las iteraciones de entrenamiento.

Gracias a nuestros words embeddings obtenidos por el entrenamiento en el modelo word2vec, ahora podemos realizar funciones como 'most similar'. Para esto testamos nuestro modelo obteniendo las palabras más similares (con vectores más similares a “machine learning”):

```
ModeloPrueba=model.wv.most_similar("machine_learning")

print(ModeloPrueba)

[('deep_learning', 0.16754990816116333), ('software', 0.1469424068927765), ('new', 0.14542436599731445), ('computer', 0.14073587954044342), ('learn', 0.13994109630584717), ('images', 0.12864306569099426), ('data', 0.12860141694545746), ('neural', 0.12691855430603027), ('use', 0.12555642426013947), ('image', 0.12384194135665894)]
```

Nos dio que tiene una similitud del 0.16 con “Deep_learning”.

3.1.6-Resultados – Candidate Profile.

Entonces, como dijimos previamente, lo que hacemos con los word embeddings es obtener las palabras más similares a las keywords que les pasamos como parámetro (las cuales son: statistics, Language, machine_learning, Deep, Python y data). Y macheamos todas estas palabras en los CVs en PDF de los candidatos (mediante el macheador “PhraseMatcher” de la librería Spacy) para obtener un Perfil de candidato.

Ejemplo de obtención de palabras más similares a ‘statistics’:

```
sim_words_NLP=[k[0] for k in model.wv.most_similar("statistics")]  
  
sim_words_NLP  
['data',  
 'image',  
 'new',  
 'software',  
 'using',  
 'software_development',  
 'data_science',  
 'deep_learning',  
 'computer',  
 'methods']
```

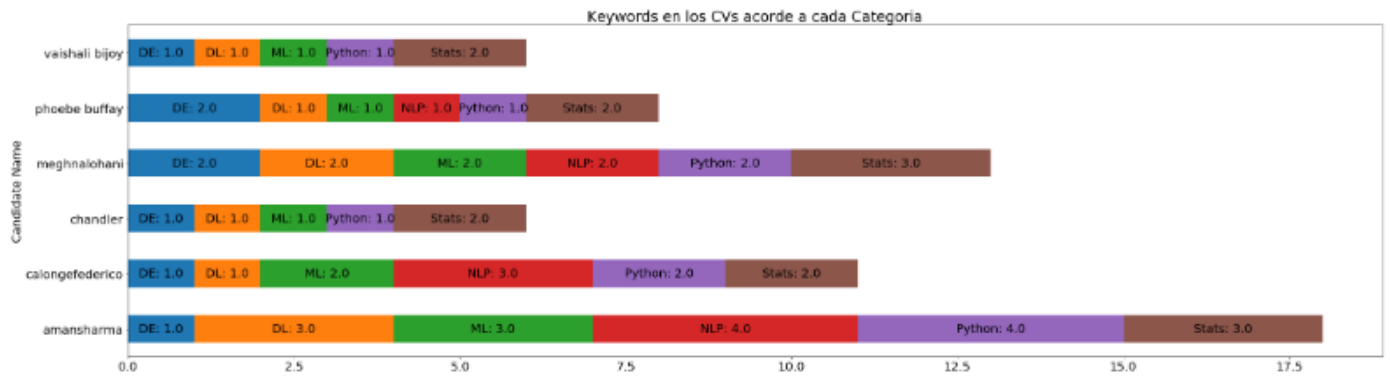
Ejemplo de las Keywords que macheó el candidato “calongefederico”:

	Candidate Name	Subject	Keyword	Count
0	calongefederico	ML	data	1
1	calongefederico	NLP	data	1
2	calongefederico	Stats	data	1
3	calongefederico	DL	data	1
4	calongefederico	Python	data	1
5	calongefederico	Stats	computer	1
6	calongefederico	ML	computer	1
7	calongefederico	NLP	systems	1
8	calongefederico	Python	systems	1
9	calongefederico	NLP	learning	1
10	calongefederico	DE	learning	1

Profile de todos los Candidatos (realizando SCORES – sumando todos los ‘count’ del cuadro anterior para cada ‘subject’):

	Candidate Name	DE	DL	ML	NLP	Python	Stats
0	amansharma	1.0	3.0	3.0	4.0	4.0	3.0
1	calongefederico	1.0	1.0	2.0	3.0	2.0	2.0
2	chandler	1.0	1.0	1.0	0.0	1.0	2.0
3	meghnalohani	2.0	2.0	2.0	2.0	2.0	3.0
4	phoebe buffay	2.0	1.0	1.0	1.0	1.0	2.0
5	vaishali bijoy	1.0	1.0	1.0	0.0	1.0	2.0

Y por último obtenemos la visualización de los perfiles:



En este gráfico final observamos que el último candidato por ejemplo tiene conocimientos más amplios que los demás. Y que Chandler y Bijoy son los que menos Keywords machearon en su CV.

Igualmente esta última visualización NO es fiables. Ver **Conclusión**.

3.2- Análisis de Ofertas/Posiciones de Trabajo y obtención de Perfiles de Candidatos.

En este 2do análisis no se utilizó ningún modelo para entrenar nuestros datos (como Word2Vec en el caso del análisis 1). Simplemente analizamos Ofertas de Trabajo (nuestro **Corpus**) publicadas en Indeed al día de la fecha (06/07/2020) para distintas posiciones (Data Scientist, Machine Learning, Data Engineer, Java Programmer, HCM Consultant) mediante técnicas de NLP (tokenización, stemming/lematización, POS Tagging, etc.).

Luego, teniendo una lista de Keywords predefinidas de Skills y Tools requeridas mediante consideraciones propias, el fin de esto es:

- Por un lado, analizar cuáles son las 20 principales skills y tools mediante visualizaciones, y cuál es el nivel de educación mínimo para los puestos publicados como “Data Scientist”;
- y por el otro lado, obtener un TOP 5 de Skills y Tools para las demás posiciones (Machine Learning, Data Engineer, Java Programmer, HCM Consultant). Estas Skills y Tools las utilizaremos como nuevas Keywords para machear directamente contra los CVs de los Candidatos; y así obtener, como en el análisis 1, un Perfil de Candidato; pero esta vez será en base a Keywords predefinidas y NO en base a palabras similares a Keywords obtenidas mediante Word2Vec.

3.2.1-Data Set.

Dispondremos nuevamente de 2 dataset con los cuales demostraremos el funcionamiento de NLP en Python con el ambiente Anaconda. Estos 2 dataset son:

- Dataset de CVS: son los mismos que utilizamos en el *análisis 1*. Como dijimos previamente, propiamente dicho no son Datasets, sino que estos datos NO los usamos para entrenar, simplemente los usamos para machear con las keywords ya previamente **definidas por nosotros**. Son nuestros ejemplos de prueba con la información de nuestros Candidatos. Son los mismos 6 CVs utilizados en el análisis 1.
- Dataset de Job Posts: Son datos de posts de puestos de trabajo vacantes extraídos de Indeed.com mediante Selenium para distintas búsquedas laborales, los cuales luego utiliza nuestro sistema para analizar el contenido de la columna ‘job_description’ y realizar los macheos con keywords predefinidas.

	job_title	company	location	job_description
0	Data Scientist I	AMZN CAN Fulfillment Svcs, ULC	- Vancouver, BC	Master or PhD in Computer Science, Machine Lea...
1	Data Scientist	AbCellera Biologics	- Vancouver, BC	Full-timeVancouver, BC\nJUNE 9, 2020\nJob ID...
2	Data Scientist	Rock FOC Technology	- Windsor, ON	Minimum Qualifications\nMaster's degree or abo...
3	Data Scientist Consultant, Cloud Support Data ...	Google	- Waterloo, ON	Minimum qualifications:\n3 years of experience...
4	Data Scientist	Lightspark	- Vancouver, BC	Who We Are\nLightspark is an innovative cleant...
5	Junior Data Scientist	Fiddlehead	- Moncton, NB	What you'll do\nResponsibilities\nIn this ro...
6	Data Scientist	Suncor Energy Services	- Calgary, AB	LOCATION: Calgary, Alberta (CA-AB)\nJOB NUMBER...
7	Data Scientist	SkipTheDishes	- Winnipeg, MB	We're revolutionizing the way humanity eats, a...
8	Data Scientist - Programmer	IQVIA	- Kirkland, QC	IQVIA™ is the leading human data science compa...
9	DATA SCIENTIST	Tematrix	- Ottawa, ON	The quantitative research team is looking for ...

Dataset de Job Posts de Indeed – 10 primeras columnas para Job Post = “Data Scientist”

3.2.2-Objetivo del análisis.

Tuvimos 2 objetivos en este análisis: analizar los TOP 20 TOOLS, TOP 20 SKILLS y MÍNIMO DE EDUCACIÓN para los job post de 'data scientist' mediante el macheo en su 'job_description'. Y, como segundo objetivo obtener, al igual que en el 1er análisis, una nueva visualización de perfiles de candidatos.

El Workflow que utilizamos para llevar todo esto a cabo en Python es el siguiente:

1. Extracción mediante Web Scrapping para obtener Data Frames de Job Postings
2. Guardamos lo Extraído en archivos CSVs y luego cargamos los Data Frames
3. Removemos registros duplicados.
4. Formamos nuestra lista de Keywords.
5. Tokenizamos la columna 'job_descriptionn
6. Aplicamos POS (Parts of Speech) Tagging y Word Stemming a los 'Job_Description'
7. Macheo de la lista de keywords y job descriptions:
8. Visualización de los resultados para 'Data Scientist'.
9. Obtención de nuevas Keywords (TOP 5 Skills y TOP 5 Tools) de las demás posiciones.
10. Extraemos los CVs/Resumes usando PyPDF y los convertimos en string.
11. Contruimos el perfil del candidato (candidate profile) mediante el macheo de las keywords obtenidas del paso 9- al texto contenido en el CV.
12. Contamos las palabras para cada categoria e imprimimos nuestro 'candidate profile'.
13. Visualizamos el candidate profile con matplotlib.

***Archivos con el código:**

- Para los pasos 1-9: Job_Positions_Extract&Analysis.ipynb
- Para los pasos 10-13: CV_Scoring_V2.ipynb

3.2.3-Extracción de datos.

Utilizaremos Selenium para ejecutar Google Chrome automáticamente y realizar las búsquedas para las distintas posiciones de trabajo. Selenium es un entorno de pruebas de software para aplicaciones basadas en la web. Selenium provee una herramienta de grabar/reproducir para crear pruebas sin usar un lenguaje de scripting para pruebas.

Por ejemplo, para el caso de 'Java Programmer':

```
#Hacemos lo mismo para la posición de "Java Programmer":  
  
loc = '' #Ponemos vacío para que busque en TODAS las localizaciones.  
#loc = 'Toronto%2C+ON' #Este es el caso si queremos solo obtener los job posts de Toronto.  
q = 'title%3A%28java+programmer%29' #Lo que queremos buscar:  
df_java_programmer = scrapeindeed(q, loc, 15, True) #Armamos nuestro dataframe y posteriormente lo pasamos a csv y PKL.  
  
Job Results: 10  
Pages: 1  
Completed Post 1 of Page 1 - Java Programmer  
Completed Post 2 of Page 1 - Java Programmer analyst  
Completed Post 3 of Page 1 - Java programmer  
Completed Post 4 of Page 1 - Corrective Maintenance Java Programmer  
Completed Post 5 of Page 1 - Java, C# Programmer  
Completed Post 6 of Page 1 - Senior Programmer/Developer - Java AWS Lead - Full Time Remote Work in Canada  
Completed Post 7 of Page 1 - Senior Programmer/Developer - Java AWS Lead - Mississauga  
Completed Post 8 of Page 1 - Java Programmer Developer
```

3.2.4-Construcción de lista de Keywords predefinida.

Antes de buscar en los "job_description" necesitamos una lista de keywords que representen las tools/skills/degrees (herramienta/habilidades/nivel de educación).

Para este análisis, usaremos una simple aproximación mediante listas de keywords. Estas listas están hechas a nuestro criterio, con las palabras que vimos que contenían la mayoría de los job posts:

- Para la lista de keywords para "tools" se pensó en hacer una lista basada en conocimiento acerca de data science: Python, R, Hadoop, Spark, etc. Y luego, ir viendo algunos job posts al azar y agregar las tools que aparecían ahí y que faltaban en nuestra lista. Además, separamos las keywords en una lista de palabras "simples" y otras de palabras "compuestas" (bi o trigramas). Esto lo hacemos ya que necesitamos machear estas 2 listas de Keywords en los 'job_description' de 2 distintas maneras.

Hay keywords con una única letra (como el lenguaje R o C). En estos casos, como "c" y "r" es común que aparezcan dentro de palabras... "can", "clustering", "random", etc... necesitamos procesarlas mediante TOKENS para machear así cuando aparezca SOLO la letra C o R en los 'job_description'.

- Y para la lista de keywords para "skills" se realizó el mismo procedimiento... en base a nuestro propio criterio y con palabras que se vieron en job posts vistos al azar.
- Para el caso del nivel de educación ("degree") utilizamos un procedimiento distinto. Como se quiere buscar el mínimo nivel requerido de nivel de educación, entonces necesitamos un valor NUMÉRICO para hacer los macheos y rankings. De esta manera utilizamos 1 para representar a 'bachelor' y 'undergraduate', 2 para representar a 'master' y 'graduate', etc. De esta manera rankeamos los degrees en números del 1 al 4, donde mayor sea este número será mejor el nivel de educación.

De esta manera, por ejemplo, nuestra lista de keywords para tools será:

```
# Obtenemos algunos keywords:
tool_keywords1 = ['python', 'pytorch', 'sql', 'mxnet', 'mlflow', 'einstein', 'theano', 'pyspark', 'solr', 'mahout',
'cassandra', 'aws', 'powerpoint', 'spark', 'pig', 'sas', 'java', 'nosql', 'docker', 'salesforce', 'scala', 'r',
'c', 'c++', 'net', 'tableau', 'pandas', 'scikitlearn', 'sklearn', 'matlab', 'scala', 'keras', 'tensorflow', 'clojure',
'caffe', 'scipy', 'numpy', 'matplotlib', 'vba', 'spss', 'linux', 'azure', 'cloud', 'gcp', 'mongodb', 'mysql', 'oracle',
'redshift', 'snowflake', 'kafka', 'javascript', 'qlik', 'jupyter', 'perl', 'bigquery', 'unix', 'react',
'scikit', 'powerbi', 's3', 'ec2', 'lambda', 'ssrs', 'kubernetes', 'hana', 'spacy', 'tf', 'django', 'sagemaker',
'seaborn', 'mllib', 'github', 'git', 'elasticsearch', 'splunk', 'airflow', 'looker', 'rapidminer', 'birt', 'pentaho',
'jquery', 'nodejs', 'd3', 'plotly', 'bokeh', 'xgboost', 'rstudio', 'shiny', 'dash', 'h2o', 'h2o', 'hadoop', 'mapreduce',
'hive', 'cognos', 'angular', 'nltk', 'flask', 'node', 'firebase', 'bigtable', 'rust', 'php', 'cntk', 'lightgbm',
'kubeflow', 'rpython', 'unixlinux', 'postgresql', 'postgresql', 'postgres', 'hbase', 'dask', 'ruby', 'julia', 'tensor',
# Estos de abajo son paquetes R pero que al final parece que no afectaron mucho.
'dplyr', 'ggplot2', 'esquisse', 'bioconductor', 'shiny', 'lubridate', 'knitr', 'mlr', 'quanteda', 'dt', 'rcrawler', 'caret', 'rmarkdown',
'leaflet', 'janitor', 'ggvis', 'plotly', 'rcharts', 'rbokeh', 'broom', 'stringr', 'magrittr', 'slidify', 'rvest',
'rmysql', 'rsqllite', 'prophet', 'glmnet', 'text2vec', 'snowballc', 'quantmod', 'rstan', 'swirl', 'datasciencr']

# Otros keywords de más de 1 palabra (bi o tri gramas):
tool_keywords2 = set(['amazon web services', 'google cloud', 'sql server'])
```

Y nuestra lista de skills:

```
# Skills/knowledge requerido:
skill_keywords1 = set(['statistics', 'cleansing', 'chatbot', 'cleaning', 'blockchain', 'causality', 'correlation', 'bandit',
'anomaly', 'kpi', 'dashboard', 'geospatial', 'ocr', 'econometrics', 'pca', 'gis', 'svm', 'svd', 'tuning', 'hyperparameter',
'hypothesis', 'salesforcecom', 'segmentation', 'biostatistics', 'unsupervised', 'supervised', 'exploratory',
'recommender', 'recommendations', 'research', 'sequencing', 'probability', 'reinforcement', 'graph', 'bioinformatics',
'chi', 'knn', 'outlier', 'etl', 'normalization', 'classification', 'optimizing', 'prediction', 'forecasting',
'clustering', 'cluster', 'optimization', 'visualization', 'nlp', 'c#', 'regression', 'logistic', 'nn', 'cnn', 'glm',
'rnn', 'lstm', 'gbm', 'boosting', 'recurrent', 'convolutional', 'bayesian', 'bayes'])

# Otros keywords de más de 1 palabra (bi o tri gramas):
skill_keywords2 = set(['random forest', 'natural language processing', 'machine learning', 'decision tree', 'deep learning',
'experimental design', 'time series', 'nearest neighbors', 'neural network', 'support vector machine', 'computer vision',
'machine vision', 'dimensionality reduction', 'text analytics', 'power bi', 'a/b testing', 'ab testing', 'chat bot',
'data mining'])
```

Y por último, nuestra lista de educación mínima requerida:

```
#Educación:
degree_dict = {'bs': 1, 'bachelor': 1, 'undergraduate': 1,
'master': 2, 'graduate': 2, 'mba': 2.5,
'phd': 3, 'ph.d': 3, 'ba': 1, 'ma': 2,
'postdoctoral': 4, 'postdoc': 4, 'doctorate': 3}

degree_dict2 = {'advanced degree': 2, 'ms or': 2, 'ms degree': 2, '4 year degree': 1, 'bs/': 1, 'ba/': 1,
'4-year degree': 1, 'b.s.': 1, 'm.s.': 2, 'm.s': 2, 'b.s': 1, 'phd/': 3, 'ph.d.': 3, 'ms/': 2,
'm.s/': 2, 'm.s./': 2, 'msc': 2, 'master/': 2, 'master\'s/': 2, 'bachelor\'s/': 1}
degree_keywords2 = set(degree_dict2.keys())
```

3.2.5-Tratamientos a la columna 'job description'.

Luego aplicamos tokenización, POS (Parts of Speech) Tagging y Word Stemming/Lematización a la columna 'job_description' creando una nueva columna 'job_description_word_set', la cual tendrá nuestros job_descriptions tokenizados, filtrados, ordenados y en minúscula para el correcto macheo con nuestras Keywords. .

Ejemplo de esta nueva columna para un HCM Consultant:

	job_title	company	location	job_description	job_description_word_set
0	SAP HCM/PY Consultant	Ouest	- Vancouver, BC	Greater Vancouver Area/r/nFull-time/r/nRole de...	{pm, chang, function, other, time, minimum, ye...
1	HCM Implementation Consultant	OneSource Virtual	- Quebec City, QC	Organizations across the globe rely on OneSour...	{written, %, facilit, oper, ', process, execut...
2	Delivery Consultant – HCM Time & Labor / Absen...	Oracle	- Canada	Delivery Consultant – HCM Time & Labor / Absen...	{product/technolog, time, best, written, %, co...
3	Oracle Cloud HCM Technical Consultant	EAInfoBiz	- Toronto, ON	Job Brief/r/nOracle Cloud HCM Technical Consul...	{cloud/fus, best, time, ip, counterpart, progr...
4	SAP HCM Time management Consultant	IBM	- Halifax, NS	Introduction/r/nAs a Package Consultant at IBM...	{univers, iran, condit, first, readi, time, re...
5	Senior SAP HCM Consultants	PAPINS Unlimited	- Newcastle, ON	We are constantly seeking Senior SAP HCM Consu...	{certifi, travel, particip, 3, minimum, writte...
6	Oracle HCM Cloud Time & Labour Consultant	Cloudworks	- Toronto, ON	Cloudworks is a leading cloud technology consu...	{off-sit, time, part, recruit, written, progra...
7	Oracle HCM Cloud Functional Payroll Consultant...	Cloudworks	- Toronto, ON	Cloudworks is a leading cloud technology consu...	{off-sit, recruit, part, time, best, program, ...

3.2.6-Resultados para Data Scientists.

Luego realizamos el macheo de las listas de keywords para SKILLS, TOOLS y DEGREE en los job descriptions:

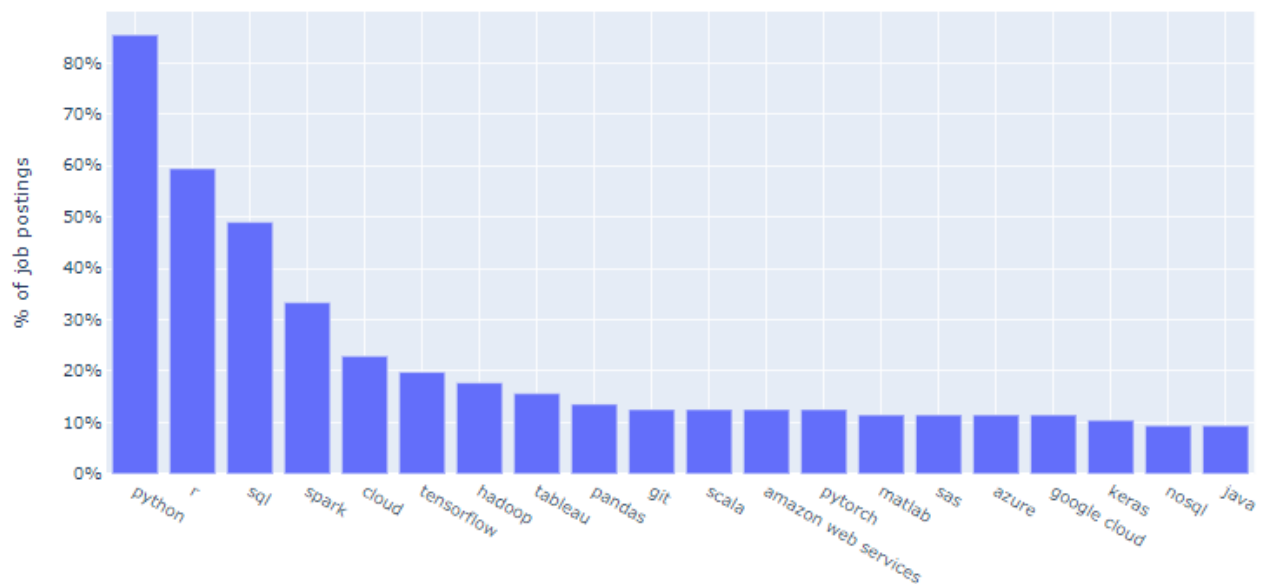
- Para TOOLS y SKILLS vimos que tenemos 2 tipos de listas de keywords: las keywords de sola palabra y las keywords formadas por más de una palabra (bi o trigramas). A cada una la tratamos de distinta manera.
- Para el nivel de educación lo que hacemos es usar el mismo método que usamos en tools y skills para machear las keywords. Obtendremos siempre el nivel MÍNIMO... de esta manera, cuando las keywords "bachelor" y "master" existan en el 'job_description', solo machearemos "bachelor" ya que se entiende que es el MÍNIMO de educación requerida para este trabajo.

Y por último realizamos la visualización de datos/resultados para el puesto de 'Data Scientist'. Utilizamos bar charts para sumarizar los resultados.

Para cada keyword particular de tools/skills/education_level, lo que hacemos es contar el número de job_descriptions que machearon con esta keywords. Luego calculamos el porcentaje sobre TODOS los job descriptions para realizar los gráficos.

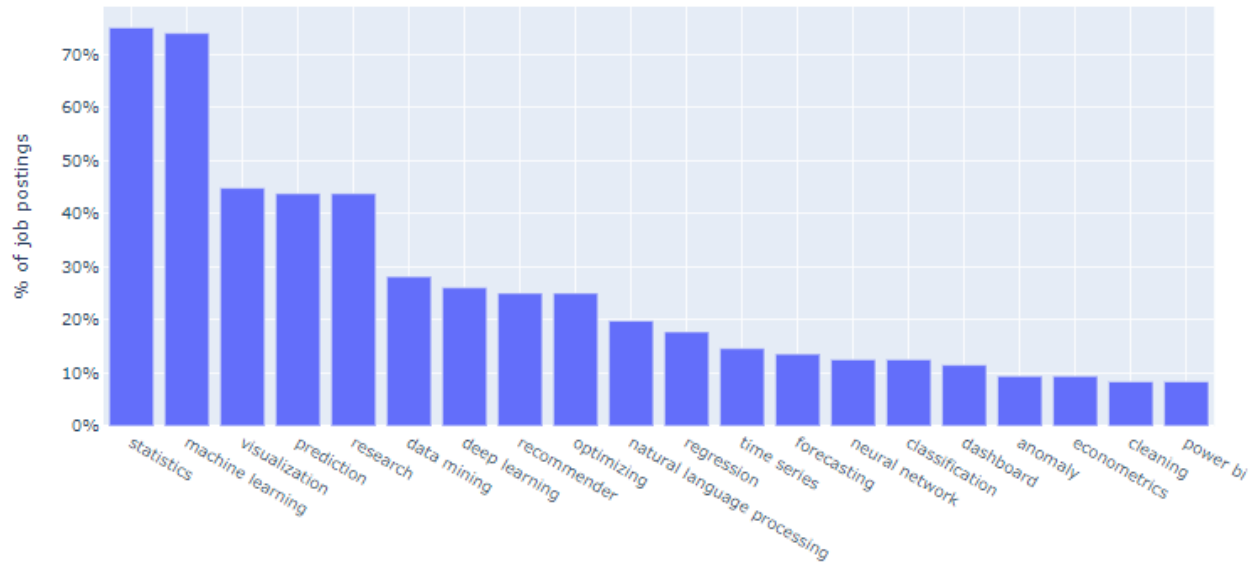
Para la lista de tools y skills presentaremos un TOP 20 de los más populares; y para el nivel de educación sumaremos de acuerdo al mínimo nivel requerido.

El gráfico correspondiente al TOP 20 de Tools para Data Scientists es el siguiente:



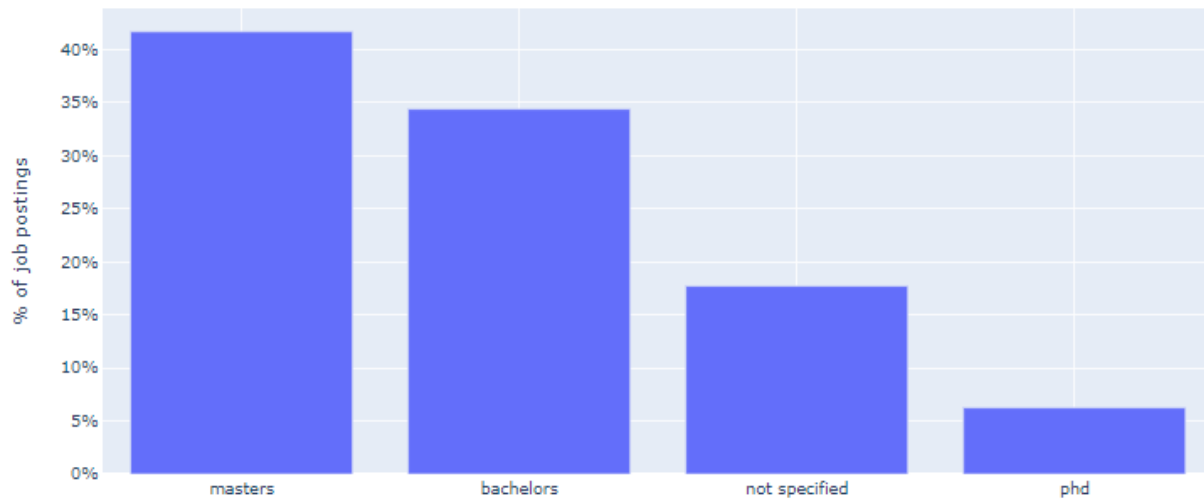
De esta manera, observamos que en dicha sección ('Tools') Python es el indiscutido ganador con un **85%** de los job postings totales. También hay otras tools y paquetes de Python que están en los primeros puestos como tensorflow, pytorch, keras. También aparece SQL (lenguaje de base de datos), Java y C++ (otros lenguajes que son más útiles para "data engineers"), R (el competidor de Python para análisis de datos), Linux (sistema operativo) y herramientas de Big Data: Spark, Cloud, AWS y Hadoop.

El gráfico correspondiente al TOP 20 de Skills para Data Scientists es el siguiente:



En esta sección, statistics (con un 75%) y machine learning (con un 74%) lideran los Skills necesarios para un Data Scientist.

El gráfico correspondiente al mínimo de educación requerida para Data Scientists es el siguiente:



Por último, observamos que hay un gran porcentaje de los jobs que pide **mínimamente** un nivel de master para postularse (un 42%).

3.2.7-Resultados para los demás Job Posts – Candidate Profile V2.

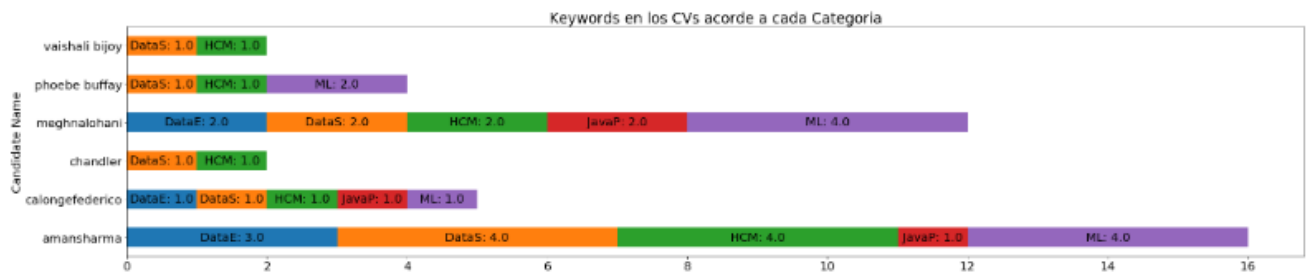
Obtuvimos los TOP 5 skills y TOP 5 tools más usados para cada disciplina. Estas disciplinas son:

- Data Scientist (obtenida previamente).
- Machine Learning.
- Data Engineer.
- Java Programmer.
- HCM Consultant.

Estos 5 skills y 5 tools los cargamos en listas de Keywords para cada disciplina y luego los macheamos directamente mediante el PhraseMatcher de Spacy con los CVs de los Candidatos, obteniendo nuevamente un Perfil de Candidatos (como en el análisis 1):

	Candidate Name	DataE	DataS	HCM	JavaP	ML
0	amansharma	3.0	4.0	4.0	1.0	4.0
1	calongefederico	1.0	1.0	1.0	1.0	1.0
2	chandler	0.0	1.0	1.0	0.0	0.0
3	meghnalohani	2.0	2.0	2.0	2.0	4.0
4	phoebe buffay	0.0	1.0	1.0	0.0	2.0
5	vaishali bijoy	0.0	1.0	1.0	0.0	0.0

Visualización de los Perfil de Candidatos (al igual que en el análisis 1):



Esta visualización nos muestra que el 1er y el 4to candidato tienen menos conocimiento que el resto y no tienen Keywords asociadas a conocimientos en el campo de ML, de Data Engineer o de Java Programmer. También, podemos observar que el último candidato es el que más macheos consiguió.

Igualmente esta última visualización NO es fiables. Ver **Conclusión**.

4-Conclusiones.

Esta práctica permitió adentrarnos en el mundo de NPL. Se analizaron técnicas y modelos de NPL para el análisis de Candidatos y de Posiciones Laborales, mediante la implementación en Python.

Además, nos permitió entender cómo funciona el modelo Word2Vec, qué son los Word Embeddings y qué podemos hacer con estos.

Para el caso del análisis 2 se pudieron comprobar las principales SKILLS, TOOLS y niveles mínimos de educación a tener en cuenta para un trabajo de “Data Scientist”.

En cuanto a las aplicaciones para machear keywords en los CVs de los candidatos y obtener Perfiles de los mismos, ambos resultados (ambos análisis) nos dieron visualizaciones que NO son muy fiables ni acertadas en cuanto a la clasificación del perfil. Esto es debido a varios problemas:

- **Problema 1 Análisis 1:** En el caso del análisis 1 es debido que los vectores de palabras más similares para las distintas categorías/keywords son MUY PARECIDOS. Veamos por ejemplo los casos para palabras más parecidas a “machine learning” y a “stadistics”:

```
In [18]: #Según nuestro modelo para "machine learning" sim_words_ML serian las palabras similares.
sim_words_ML=[k[0] for k in model.wv.most_similar("machine_learning")]
sim_words_ML

#Ver como descartar cuando la probabilidad es menor y porque me devuelve las 10 más parecidas... ver este parametro.
#('systems', 0.15548837184906006)

Out[18]: ['deep_learning',
          'software',
          'new',
          'computer',
          'learn',
          'images',
          'data',
          'neural',
          'use',
          'image']

In [19]: sim_words_NLP=[k[0] for k in model.wv.most_similar("statistics")]
sim_words_NLP

Out[19]: ['data',
          'image',
          'new',
          'software',
          'using',
          'software_development',
          'data_science',
          'deep_learning',
          'computer',
          'methods']
```

Vemos que casi todas las palabras coinciden en ambos vectores. Por esta razón, si una de estas palabras perteneciente por ejemplo al vector de “machine_learning” machea con el CV del candidato, también macheará esa misma palabra del vector “stadistics” por tener la misma palabra en la lista.

- **Problema 2 Análisis 2:** no tenemos en cuenta ninguna de las características de un programador en Java o un HCM Consultant. Las listas de keyword (Skills y Tools) que nosotros generamos para realizar los macheos fue solo con palabras relacionadas a Machine Learning y Data Scientists, no tuvimos en cuenta otras terminologías utilizadas (por ejemplo para un HCM Consultant es muy común que se pida conocimiento en distintos módulos: Absences, Time & Labor y en distintas herramientas: HCM Data Loader, HCM Extractions, SOAP, etc.; SKILLS y TOOLS que NO tenemos en cuenta al confeccionar nuestras listas). Y tener que agregar nuevos términos a estas listas

cada vez que quisiéramos agregar una nueva posición para considerarla, es MUY tedioso, por esto ver la solución ofrecida a este problema en **Mejoras**.

- **Problema 3 – Ambos análisis:** En ambos análisis utilizamos el matcheador de la librería Spacy (para machear palabras obtenidas mediante 'most.similar()') con las palabras del CV). Existe un problema en dicho matcheador, ya que no machea correctamente en los CVs. Por ejemplo para el caso de Meghna Lohani y Vaishali Bijoy vemos que “Java” lo detectó y macheo solo en MegnaLoaini:

COMPUTER SKILLS	<i>Languages:</i>	Python, C, C++, R (Beginner), Java
	<i>Skills:</i>	HTML, PHP, JavaScript, SQL, Data Structures, Data Science, Machine learning, Android

Parte del CV de Meghna Lohani

Skills	
C++	Java
Python	Arduino
Android	HTML
PHP	Javascript
MySQL	R studio

Parte del CV de Vaishali Bijoy

Candidate	Name	Subject	Keyword	Count
0	meghnalohani	JavaP	java	1
1	meghnalohani	ML	java	1
2	meghnalohani	HCM	machine learning	1
3	meghnalohani	DataS	machine learning	1
4	meghnalohani	ML	machine learning	1
5	meghnalohani	DataE	machine learning	1
6	meghnalohani	ML	c++	2
7	meghnalohani	HCM	cloud	2
8	meghnalohani	DataS	cloud	2
9	meghnalohani	JavaP	cloud	2
10	meghnalohani	ML	cloud	2
11	meghnalohani	DataE	cloud	2

Candidate	Name	Subject	Keyword	Count
0	vaishali bijoy	HCM	prediction	1
1	vaishali bijoy	DataS	prediction	1

Resultado de macheo de Skills para Vaishali Bijoy

Resultado de macheo de Skills para Meghna Lohani

5-Mejoras a desarrollar.

- Solución al “**Problema 1 Análisis 1**” (ver **Conclusiones**): Hay 2 posibles soluciones a esto:
 - Una opción es **utilizar un vocabulario/corpus más extenso y amplio**. Obtener más artículos de Wikipedia. De esta manera, con un corpus más grande, nuestro modelo de Word2Vec entrenará con más datos y nos devolverá word embeddings más “dispersos/aleatorios”. Ya que, como vimos en el problema, los vectores de palabras más similares de ‘machine_learning’ y ‘statistics’ tienen palabras repetidas entre sí, y deberían ser distintas para identificar bien a cada keyword, ya que después en el matcheo de skills no nos diferencia muy bien una skill de otra y no es muy preciso a razón de esto. Por esto se necesitaría un corpus mucho más grande para que se diferencien bien esto.
 - Otra opción, en vez de utilizar un corpus más grande y entrenar nuestros words embeddings, es directamente **utilizar/traer embeddings de Gensim pre-entrenados con las noticias de Google** y ver así qué sucede con las palabras de dichas listas, que en teoría deberían tener palabras más distintas, y ahí si realizaríamos los matcheos en los CVs correctamente. Ya que **actualmente nuestro vocabulario es de 1275 palabras, es muy pobre. Podemos obtener word embeddings de Google con longitudes de 3 millones.**
- Solución al “**Problema 2 Análisis 2**” (ver **Conclusiones**): necesitamos una **lista de keywords más extensa** para que realice un mejor matching. Pero, agregar palabras a la lista cada vez que quisiéramos tener en cuenta otras posiciones o cada vez que surjan nuevas tecnologías es bastante tedioso, por esto lo mejor sería:
 - Utilizar como vimos en el análisis 1 un corpus extenso sacado de artículos de Wikipedia y entrenarlo en Word2Vec (pero como vimos anteriormente, deberá ser **más grande del que utilizamos**).
 - O sino **utilizar, como dijimos previamente, embeddings ya entrenados de Google.**
- Solución al “**Problema 3 – Ambos análisis**” (ver **Conclusiones**): Una mejora/solución a esto es analizar bien en detalle cómo se producen los matcheos para comprobar efectivamente si se trata de un error del matcheador de Spacy, un error de código o un error en los PDFs (puede que el matcheador no tome en cuenta algunas columnas). Si sucediera este último caso, se podría optar por una opción donde el Candidato llene manualmente campos predefinidos del CV... de “Experiencia”, “Educación”, “Información Personal”, etc. y allí analizarlos sin tener estos problemas de matcheo.
- Hasta ahora nuestro sistema solo realiza análisis de Keywords y matcheos directos en los CVs, sin importar en qué sección esté. La idea es analizar CADA SECCIÓN del CV en forma separada y específica y realizar estos matcheos.

6-Bibliografía.

- Apunte Materia Organización de Datos – UBA:
 - <https://github.com/lrargerich/Apunte>
- Análisis 1:
 - <https://github.com/meghnalohani/Resume-Scoring-using-NLP>
- Análisis 2:
 - <https://gist.github.com/liannewriting/a08e549f186067837856494513250ff1>
 - <https://towardsdatascience.com/how-to-use-nlp-in-python-a-practical-step-by-step-example-bd82ca2d2e1e>
 - <https://towardsdatascience.com/what-are-the-in-demand-skills-for-data-scientists-in-2020-a060b7f31b11>
- <https://medium.com/@gruizdevilla/introducci%C3%B3n-a-word2vec-skip-gram-model-4800f72c871f>
- <https://www.aprendemachinelearning.com/procesamiento-del-lenguaje-natural-nlp/>