**TF-IDF**

El término de frecuencia/frecuencia inversa del documento es **una técnica estadística para saber la importancia de una palabra en un documento**. Lo hace en función de las veces que aparece cierta palabra en el documento contra la frecuencia inversa de la palabra en un conjunto de documentos. Usos:

* Clasificar resultados de motor de búsqueda
* Hacer resúmenes de texto
* Chatbots inteligentes

Funcionamiento TF-IDF

Esta técnica asigna puntuaciones a una palabra:

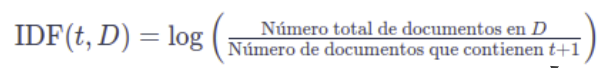
* TF: Si aparece con frecuencia en un documento entonces esta palabra es mas importante que otras que aparecen con menos frecuencia, tendrá una puntuación alta.
* IDF: Si una palabra aparece muchas veces en el documento y en el conjunto de documentos entonces NO es una palabra relevante, tendrá una puntuación baja.

Pasos para crear el modelo

1. Calcular la frecuencia de un término en el documento.



1. Calcular la frecuencia inversa de un término en el corpus (conjunto de documentos)



1. Combinar 1 y 2 para crear la matriz TF-IDF



Implementación

La librería scikit-learn proporciona 2 formas para crear la matriz TF-IDF

1. TfidfVectorizer: Crear directamente la matriz TF-IDF
2. CountVectorizer + TfidfVectorizer: Obtener TF y IDF por separado

Prácticas

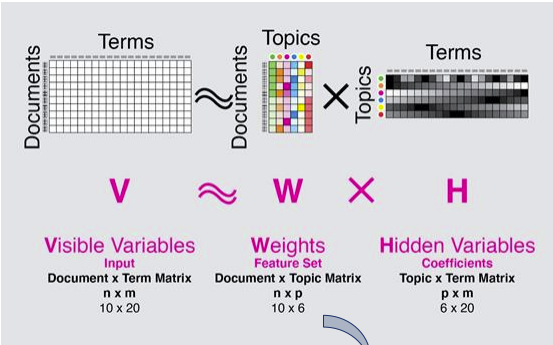
* 3.3: Analizar noticias de los medios

**Topic Modeling**

El Topic Modeling es una **técnica no supervisada de NLP para detectar y extraer relaciones entre términos en grandes volúmenes de información.** Un tema (tópico) es un conjunto de palabras que suelen aparecer juntas en los mismos contextos, expresan una distribución de probabilidades de aparición de las distintas palabras del vocabulario.

El algoritmo LDA (Latent Dirichlet Allocation) implementa TopicModeling y puede ser utilizado a través de scikit-learn. Este algoritmo genera una matriz de términos del documento. Tiene dos objetivos principales:

* Cada término hará una contribución positiva a un conjunto de temas
* Cada tema puede ser descrito por una combinación de términos



Prácticas

* 3.4: Topic Modeling vs K-Means

**Embeddings**

Las incrustaciones de palabras (embeddings) son **representaciones vectoriales de una palabra.** Su objetivo es transformar toda la información de una palabra (significado, gramatical, etc.…) y convertirla en una forma numérica para un ordenador.

Los vectores resultantes permiten resumir información de un objeto (palabra) usando números. Contra mayor es el número de dimensiones del vector más información guardan.

De esta forma (y a diferencia de una vectorización One-hot) se pueden realizar operaciones entre los vectores y calcular la distancia entre las palabras.

Word2Vec

Algoritmo de aprendizaje estadístico para hacer embeddings. Recibe un corpus y genera una representación vectorial de cada palabra, acercando palabras con contextos similares. Un ejemplo de aplicación puede ser predecir la siguiente palabra de una frase.

**LLMs (Large Language Models)**

Los grandes modelos de lenguaje (LLMs) son modelos de aprendizaje profundo (redes neuronales con transformadores) para comprender y generar texto en lenguaje natural. El término “grande” hace referencia a que están pre-entrenados con un gran número de parámetros y un gran corpus. Resuelven tareas de PLN como generación de texto, resúmenes, traducción, pregunta-respuesta, etc.…

Transformadores

Los transformadores permiten capturar patrones del lenguaje complejos en un gran conjunto de datos. Su mecanismo clave es la **atención**, la cual permite otorgar cierta importancia a las palabras en función de su contexto. Las redes neuronales basadas en transformes combinan 2 elementos:

* Encoder: toma una entrada y produce una secuencia de estados ocultos con información sobre el contexto a través de embeddings
* Decoder: dado un texto, utiliza un codificador para generara una serie de predicciones.

1. Ingesta de las palabras y hace el encodding para generar los estados ocultos
2. Utilizando la atención curzada, toma las palabras y sus estados para proponer un equivalente
3. Itera para actualizar el decoder hasta la secuencia final

Tipos de LLMs

* Modelos autoregresivos: predicen la siguiente palabra. Entrenados con la probabilidad de cada palabra en un conjunto de datos. Ejemplos: GPT
* Modelos de autocodificado: reconstruir texto incorrecto y detectar palabras ausentes. Ejemplos: BERT (Bidirectional Encoder Representations from Transformers)
* Modelos híbridos: Combinan autoregresivos con autocodificado. Ejemplos: T5 (Text-to-text Transformer)

**LangChain**

Potente framework para crear aplicaciones relacionadas con el PLN. Permite ampliar las funcionalidades de los LLMs. Componentes:

* Modelos: LLMs entrenados con gran cantidad de texto
* Prompts: texto que guía al LLM para generar el resultado adecuado. LangChain permite definir plantillas para dar forma a los prompts. Ejemplo: especificar un formato de salida, proporcionar contexto, imitar personas, limitar la salida, etc...
* Chains: secuencia de instrucciones para realizar una tarea. Permite conectar componentes de LangChain
* Memoria: almacenamiento de información para que el LLM pueda acceder después. Ejemplo: resultado de la cadena anterior, contexto, etc.
* Índices: documentación externa (no incluida en el entrenamiento)
  + Cargadores de documentos: en la nube, web, bases de datos, etc.
  + Base de datos vectoriales: datos desestructurados almacenados como embeddings
  + Divisores de texto: Separar el texto en partes más pequeñas para que pueda ser procesados de forma más eficiente
  + Recuperadores: recuperar e integrar la información externa recolectada
* Agentes y herramientas: Los agentes utilizan un modelo para determinar que acciones realizar. Las herramientas permiten a los agentes de LangChain interactuar con el mundo real (Google Search, Wikipedia, etc.)

Prácticas

* 3.5: Creación de un asistente con Langchain
* 3.6: Chatbot para consultas a documentos PDF

**RASA**

Los chatbots antiguos utilizaban patrones de coincidencia, dando respuestas predefinidas y limitadas. No utilizaban aprendizaje profundo por lo que no mejoraban con el tiempo.

RASA es un framework para la creación de asistentes y chatbots en Python y de código abierto. Basa su funcionamiento en 2 componentes principales:

* NLU: analizar el texto para poder comprenderlos
* Core: tomar decisiones y dar seguimiento a la conversación

Comandos

* Inicialización del entorno: rasa init
* Ejecución del entorno: rasa Shell
* Entrenamiento del modelo: rasa train
* Manejar acciones personalizadas (lanzar servidor): rasa run actions
* Entrenamiento personalizado: rasa interactive

Estructura del proyecto

* Configuración y pipelines de entrenamiento: /config.yml
* Enpoints: /endpoints.yml
* Conexiones externas: /credentials.yml
* Declaración de intents, respuestas, slots, formulario: /domain.yml
* Historias del ususario: /data/stories.yml
* Reglas: /data/rules.yml
* Definir intents ususario: /data/nlu.yml
* Acciones definidas del usuario: /actions/actions.py

Gestión de desvíos

* Mensajes fuera de contexto (reglas específicas)
* Recuperaciones (fallbacks): mecanismos para manejar mensajes en los que el modelo NLU tiene poca confianza al clasificar el mensaje. Se puede implementar de 2 formas
  + FallbackClassifier: crear una regla para que el intent ‘nlu\_fallback’ tenga cierta respuesta
  + RulePolicy: especificar un threshold y una acción por defecto. Agregar una respuesta ‘utter\_default’

Acciones

Cada acción tiene su propia clase con 2 funciones

* Name: nombre de la acción definido en domain.yml
* Run: código a ejecutar. Parámetros:
  + Dispatcher: para enviar mensajes al usuario
  + Tracker: acceder a slot, intents, entidades, etc…
  + Domain: acceso al fichero domain.yml

Para habilitar las acciones que he hacerlo en endpoints.yml y lanzar el servidor

Formularios

RASA cuenta con slots para almacenar datos

Para utilizar un formulario:

* Los formularios se tienen que declara en domain.yml bajo ‘forms’
* Las respuestas de un formulario tienen que seguir ‘utter\_ask\_slot’
* En rules hay reglas para activar y desactivar el formulario

Los formularios pueden desviarse:

* Referencia repentina a otras intenciones: Para casos puntuales se hace referencia a las reglas
* Petición de cancelación: un usuario pide salir del formulario.
  + Agregar una historia en stories.yml
  + Agregar un intent en nlu.yml
  + Domain.yml: agregar respuesta para confirmar la cancelación