Técnicas Actuales de Programación 2024/2025
Examen Final - Enero

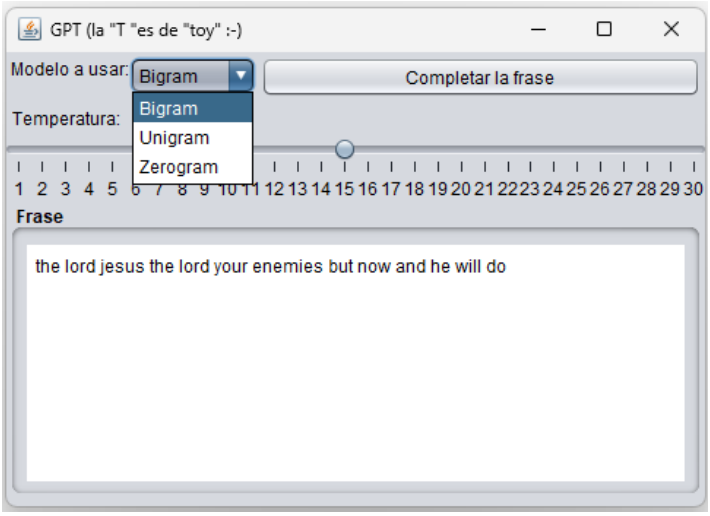
Responda al examen con un proyecto que lleve su nombre (elija Java with Ant -> Java Application).

Además de que el código sea razonablemente correcto y dé los resultados esperados, se valorará que esté limpio de restos de pruebas y de código anulado con comentarios, que esté bien formateado (Nebeans ayuda), y que incluya algún comentario donde se considere necesario. En el GUI se valorará el redimensionamiento razonable.

Recientemente la IA se ha hecho viral gracias a los GPTs (Generative Pretrained Transformers). Con este ejercicio vamos a fabricar y usar unos GPTs, donde la "T" no será de Transformer sino de "Toy". Es decir, en vez de pre-entrenar un modelo muy complejo (una Red Neuronal de tipo Transformer), lo haremos con unos modelos "de juguete".

EL GUI

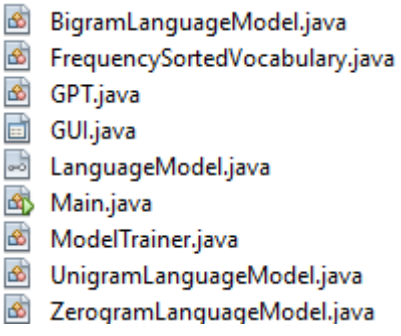
Antes de nada, veamos el GUI con que probaremos el funcionamiento, que podrá ser algo funcionalmente similar a lo que se muestra en la figura.



En el campo de texto podremos comenzar una frase y al pulsar el botón, el modelo seleccionado, con el parámetro "temperatura" -que determina un cierto nivel de aleatoriedad- irá añadiendo palabras hasta que considere que termina la frase.

Estructura de la aplicación

La estructura de la aplicación puede ser la que se ve en la imagen, donde la clase **Main.java** se encarga de:

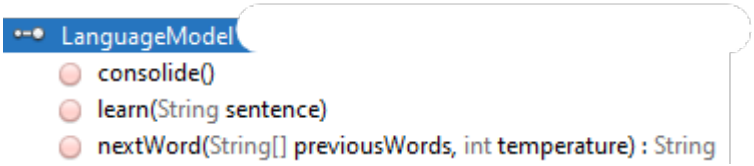


- Generar tres GPTs (**GPT.java**), cada uno con un modelo de lenguaje distinto (**BigramLanguageModel.java**, **UnigramLanguageModel.java** y **ZerogramLanguageModel.java**) que implementan en interface **LanguageModel.java**
- Entrenar los tres modelos mediante la clase **ModelTrainer.java**
- Arrancar el **GUI** proporcionandole los GPTs.

La clase **FrequencySortedVocabulary.java** permitirá generar de un modo sencillo una parte de cada uno de los modelos.

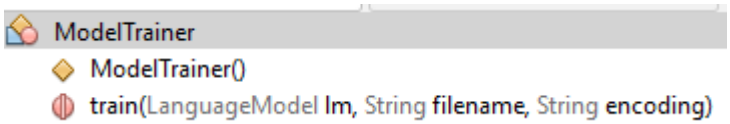
El interface LanguageModel.java

Un modelo de lenguaje vendrá definido por tres acciones:



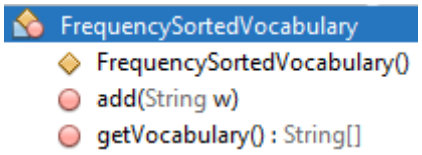
Inicialmente se entrena aportándole numerosas frases (`learn(.)`). Una vez que se le ha aportado todo el material de aprendizaje puede comenzar a usarse, pero es posible que para ello sea adecuado "reorganizar" las estructuras de datos utilizadas en la frase de aprendizaje de un modo que se adapte mejor a la fase de uso (`consolide()`). Y en la fase de uso le aportamos una secuencia de palabras (que puede ser vacía) y responde con la palabra siguiente según ha aprendido, pero con un cierto nivel de aleatoriedad dado por el parámetro "`temperature`" (`nextword(.)`)

El aprendizaje: la clase ModelTrainer



Esta clase contiene únicamente un método que recibe la referencia a un modelo de lenguaje y a un fichero y su codificación, de modo que para todas las frases que lee del fichero llama al método `learn(.)` del modelo de lenguaje, y una vez terminada la lectura de todas las frases llamará al método `consolide()` para que pueda ser usado.

La clase FrequencySortedVocabulary



Esta clase es de utilidad posterior a la hora de generar los modelos de lenguaje. Su función es muy sencilla: admite palabras (`add(.)`) que va almacenando a modo de vocabulario junto con la cuenta del número de apariciones de cada una de ellas. En cualquier momento es capaz de proporcionar un array de palabras ordenadas de mayor a menor número de apariciones (`getVocabulary()`)

Los modelos de lenguaje

Implementaremos tres modelos de lenguaje "de juguete" escribiendo para cada uno de ellos los métodos heredados del interface `LanguageModel`.

El modelo menos "inteligente" (`ZerogramLanguageModel`) será un modelo que únicamente aprenda el vocabulario y proporcione una palabra al azar (un no-modelo en realidad). Para esto nos resultará útil la clase `FrequencySortedVocabulary` resultando indiferente que el vocabulario esté ordenado.

Un modelo un poco más "inteligente" será el que proporcione palabras teniendo en cuenta su frecuencia de aparición en el lenguaje (`UnigramLanguageModel`). En este caso la clase `FrequencySortedVocabulary` encaja exactamente con la estructura de datos que el modelo debe manejar.

Si continuamos "mejorando" el modelo en la misma dirección, podemos pasar a uno que proporcione una palabra teniendo en cuenta su frecuencia de aparición, pero, ahora ya sí, teniendo en cuenta el contexto previo del modo más simple, es decir teniendo en cuenta sólo la última palabra dada (`BigramLanguageModel`). En este caso la estructura de datos necesaria consistirá en un

`FrequencySortedVocabulary` por cada palabra del vocabulario, de modo que en fase de entrenamiento se toman las palabras por pares y la primera determina el vocabulario a entrenar con la segunda.

Detalles de entrenamiento:

Los modelos deben aprender vocabularios teniendo en cuenta que deben predecir la siguiente palabra, y esto incluye la posibilidad de predecir el fin de frase. Para ello lo más sencillo es añadir a cada frase recibida para aprender el modelo una meta-palabra (frase+"<EOS>").

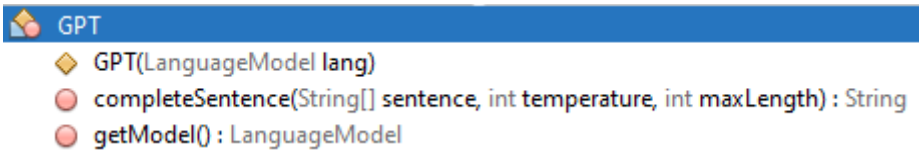
En el caso de los bigramas, como atienden a la palabra anterior, deben ser conscientes también del caso en que esten principio de frase, para lo cual podemos añadir también una meta-palabra inicial ("<BOS>"+frase+"<EOS>").

Detalles de generación de la siguiente palabra: la "Temperatura".

Para que el modelo no sea tal que a un mismo contexto devuelva siempre la misma palabra, sino que tenga cierto nivel de aleatoriedad, el parámetro entero temperatura indica el número de palabras más probables a considerar de modo que se elija una al azar de entre ellas. Si la temperatura es 1 se devolverá la más probable, pero si la temperatura es p.ej. 10, se devolverá aleatoriamente una de las 10 primeras,

El GPT

El GPT es una aplicación de un modelo de lenguaje, por lo tanto se instancia con un modelo (en nuestro caso 3 GPTS, uno con cada modelo) y proporciona métodos de utilidad.



Para nuestra aplicación gráfica basta con un método que reciba la frase en el área de texto y llame al modelo tantas veces como sea necesario para llegar a obtener un fin de frase ("<eos>") devolviendo la frase completa. Pondremos también una limitación forzada (p.ej. 100 palabras) porque, potencialmente, el modelo podría generar frases infinitas.

Los datos

El texto para el aprendizaje se encuentra en el fichero:

https://gtts.ehu.es/German/Docencia/2425/TAP/examenes/bible.clean.txt