Resúmenes de texto con modelos seq2seq y mecanismo de atención

José Jorge Rodríguez Salgado c-511, David Castillo López c-511

MATCOM, Universidad de la Habana
{j.salgado,d.castillo}@estudiantes.matcom.uh.cu

Abstract. En la actualidad es enorme la cantidad de información que está disponible, sobre todo en la web. Gran parte de esta información se presenta en forma de texto por lo que cada día se hace más necesario poder acceder a la misma en manera resumen. En el presente trabajo se proponen dos modelos de redes neuronales basados en la arquitectura Sequence to Sequence (seq2seq) junto con una capa de atención para realizar resúmenes abstractos de información textual. Para evaluar sus resultados se implementó un modelo basado en semántica latente que realiza resúmenes de manera no supervisada utilizando solamente palabras del texto original. Los resultados han mostrado ser satisfactorios sobre un dataset de opiniones de comida en inglés.

Key words: Resumen de texto abstracto, Redes Neuronales Recurrentes, Modelo seq2seq, Mecanismo de Atención

1 Introdución

La ingente cantidad de información textual que existe en la web y que en ocasiones debe ser procesada por humanos, hace necesario un avance en la realización de resúmenes automáticos para disminuir el tiempo de dicho procesamiento. A lo largo de los años se han desarrollado numerosos algoritmos y modelos para resolver este problema.

Estos intentos se pueden dividir en dos grandes grupos: los que realizan resúmenes extractivos, que como su nombre indica construyen los resúmenes extrayendo fragmentos del texto original; y los que realizan resúmenes abstractos, los cuales son capaces de elaborar resúmenes utilizando palabras que no tienen por qué aparecer en el texto pero que conforman una idea equivalente a la expresada en el texto original.

En el presente trabajo se proponen dos modelos para la generación de resúmenes abstractos basados en la arquitectura seq2seq junto a un mecanismo de atención para que los modelos sean capaces de procesar textos largos. La diferencia de ambos modelos viene dada porque en uno el codificador es puramente recurrente con tres capas LSTM, y el otro sustituye todas las capas recurrentes por capas convolucionales de una sola dimensión, excepto la última

capa LSTM que es necesaria mantener para procesar textos y resúmenes de longitudes variables.

Los resultados de ambos modelos son satisfactorios sobre un dataset de reviews de productos comestibles en inglés. Estos resultados son mucho mejores que los que se obtienen con la implementación de un modelo de semántica latente para el mismo conjunto de datos.

El presente documento está estructurado de la siguiente manera: en la sección 2 se realiza un recorrido por los principales trabajos realizados hasta el momento en el campo de los resúmenes de texto automáticos, en la sección 3 se define el problema y algunas de sus características básicas, en 4 y en 5 se comentan las características esenciales de los modelos Sequence to Sequence y se justifica el uso del mecanismo de atención el cual se comenta en 6, luego en 7 se explica el proceso de entrenamiento de los modelos y se detallan las características del conjunto de datos entrenantes. En 8 se describe el preprocesamiento necesario para realizar la correcta generación de los resúmenes, finalmente en las sección 9 se describen los modelos implementados y en 10 se resumen los resultados, las recomendaciones y conclusiones del trabajo.

2 Trabajos Relacionados

Debido a su éxito reciente, los modelos de redes neuronales han sido utilizados con resultados muy positivos en *abstractive summarization*. Este problema fue abordado primeramente con un modelo de atención neuronal, logrando resultados relevantes en múltiples *datasets*. Este modelo ha sido extendido usando una red neuronal recurrente(RNN) como decodificador. La mejora inmediata fue utilizar un *framework* donde tanto el codificador como el decodificador eran RNN, también empezaron a tener más relevancia las características léxicas y estadísticas. Uno de los mayores rendimientos en la actualidad se alcanza mediante la codificación selectiva de palabras como un proceso de destilación de información relevante.

Los modelos neuronales de *abstractive summarization* han sido utilizados para resumir cada vez textos más largos. Primero se utilizaron los modelos de extracción de palabras, los cuales mostraron peores resultados que los modelos de extracción de oraciones. Redes neuronales jerárquicas basadas en atención también han sido aplicadas en este problema. Finalmente los modelos basados en distracción se propusieron para permitir la opción de transitar por el contenido del texto y obtener una idea general de su significado. Los mejores resultados se han alcanzado utilizando redes neuronales *atencionales*, considerando los factores claves en resúmenes de texto como son fluidez, novedad y prominencia.

3 Resumen Automatizado de Textos

El resumen automatizado de textos es la tarea de producir un resumen conciso y fluido que mantenga la información clave y el significado general del texto original.

Existen dos formas de abordar este problema:

- Resumen extractivo
- Resumen abstracto

3.1 Resumen extractivo

Como su nombre indica se basa en la extracción de partes del texto. Se identifican oraciones, frases o palabras importantes y se extraen solamente esas para conformar el resumen. **El resumen consiste solamente de las partes extraídas del texto**.

3.2 Resumen abstracto

Este enfoque es más interesante y es el que se estará utilizando para resolver el problema que nos ocupa. La idea principal es generar nuevas oraciones a partir del texto original. Por tanto las oraciones que conformarán el resumen pueden estar presentes o no en el texto original.

4 Modelo Sequence to Sequence(*Seq2Seq*)

El modelo *Sequence to Sequence* se puede utilizar en problemas que involucren información secuencial. Ejemplos de estos son *Clasificación de Sentimientos, Traducción y Reconocimiento de Entidades Nombradas*. En el caso de la traducción la entrada es un texto en un idioma y la salida es el mismo texto en otro idioma. En el problema de las **entidades nombradas** la entrada es una secuencia de palabras y la salida una secuencia de etiquetas asociadas a las palabras de entrada.

El problema que se intenta resolver tiene como entrada una gran secuencia de palabras(el texto original) y su salida es una secuencia (más corta) que resuma el contenido del texto de entrada. Por tanto, cae dentro de la clasificación de problema con información secuencial y se puede aplicar el modelo *Seq2Seq*.

Existen dos componente principales en el modelo *Seq2Seq*:

- **Encoder**
- **Decoder**

La arquitectura *Encoder-Decoder* es mayormente utilizada para resolver problemas de tipo *Seq2Seq* cuando las longitudes de la entrada y la salida difieren. Existen varias variantes de *Redes Neuronales Recurrentes(RNN)*; para las componentes *Encoder* y *Decoder* las preferibles son *Gated Recurrent Neural Network(GRU)* o las *Long Short Term Memory(LSTM)*.

4.1 Encoder

Un *Encoder LSTM* lee la secuencia completa de entrada, recibiendo una palabra a la vez. Procesa la información recibida hasta el momento y captura la información contextual presente en la secuencia de entrada. Después de cada palabra recibida y procesada la información mencionada previamente queda en unos estados h_i y c_i a los cuales llamaremos *hidden state* y *cell state*, respectivamente. Los estados que se produzcan después de analizar la última palabra se utilizan para inicializar el *Decoder*.

4.2 Decoder

El *Decoder* es una red neuronal *LSTM* que se entrena para predecir la siguiente palabra en función de la palabra anterior. Recibe como entrada la salida del *Encoder* y un *token* especial denominado *Start*, que indica la primera palabra del *output* para así poder predecir la siguiente hasta que se produzca el *token End* o se llegue a la longitud máxima establecida.

5 Limitaciones de la Arquitectura *Encoder-Decoder*

El *Encoder* convierte la secuencia de entrada en un vector de longitud fija y luego el *Decoder* predice la secuencia de salida. Esto funciona mejor cuando la secuencia de entrada es corta ya que el *Decoder* tiene que mirar la entrada completa para hacer la predicción. La dificultad para el *Encoder* radica en memorizar secuencias muy largas en vectores de tamaño fijo.

La forma de lidiar con este problema se llama *mecanismo de atención*, el cual consiste en intentar predecir una palabra teniendo en cuenta solo algunas partes de la entrada y no su totalidad.

6 Mecanismo de Atención

Como se mencionó anteriormente, en modelos tradicionales de *Sequence to Sequence* se descartan todos los vectores de estado intermedios del *Encoder* y solo se utiliza el último vector de estado para inicializar el *Decoder*. Esto funciona bien para textos pequeños, pero para textos grandes tener solo en cuenta el último vector como salida del *Encoder* se convierte en una limitante. Esta observación se ha comprobado en la práctica al ver como el rendimiento de estos modelos disminuye drásticamente a medida que el tamaño del texto de *input* aumenta. Este problema se intenta resolver con algo conocido como *mecanismo de atención*, que sí tiene en cuenta los estados intermedios del *Encoder* construyendo un vector de contexto que va a utilizar el *Decoder* en cada paso para generar su salida.

La *capa de atención* va a hacer que la salida del *Decoder* en cada paso sea dependiente de la entrada (que es el token *start* o la salida anterior del *Decoder*), del estado interno de la *LSTM* (que inicialmente es la salida del *Encoder*) y del

vector de contexto que es un vector igual a la suma de todos los vectores que dió como salida el *Encoder*, multiplicados cada uno por un peso que regula que tan importante es dicho vector para la predicción del paso actual del *Decoder*. La capa de atención es una red neuronal tradicional que, utilizando propagación hacia atrás y descenso de gradiente, va a ser entrenada para dar valores correctos a los pesos que multiplican a cada uno de los vectores de estado que generó el *Encoder*.

Ahora cada vez que se vaya a evaluar el *Decoder* se va a crear el vector de contexto, que es la combinación lineal de los vectores $h_1, h_2, h_3, \ldots, h_n$, donde h_i es la salida del *Encoder* en el paso i, multiplicados por los escalares $e_1, e_2, e_3, \ldots, e_n$, donde e_i es el peso que la red neuronal de la capa de atención ya entrenada predice le corresponde a h_i. Tener en cuenta que a $e_1, e_2, e_3, \ldots, e_n$ se les aplica la función *softmax* lo cual hace que $0 \leq e_i \leq 1$ para todo i, y que $e_1 + e_2 + e_3 + \cdots + e_n = 1$, dándole un concepto probabilístico a e_i. Luego el vector de contexto sería igual a $h_1 * e_1 + h_2 * e_2 + \cdots + h_n * e_n$. Ahora el vector de contexto se concatena al *input* que recibe el *Decoder*, permitiendo que este utilice toda la información que generó el *Encoder*, de acuerdo a un concepto de atención que se le debe prestar a cada uno, de una forma elegante y eficiente.

7 Datos y proceso de entrenamiento

Los datos utilizados provienen de un dataset llamado *Amazon Fine Food Reviews*, el cual contiene 100000 opiniones sobre determinados productos de comida junto a un resumen del texto de dicha opinión. Todo ello realizado por humanos.

Para realizar el entrenamiento se realizó primeramente un preprocesamiento de los textos que incluye el proceso de *tokenización* para convertir la entrada en tensores numéricos. El preprocesamiento se detalla en la siguiente sección.

Se utilizaron el 10% de los datos para construir los conjuntos de validación y de prueba. El conjunto de prueba está constituido por 500 opiniones.

Se utilizó un *early stopping* para detener el entrenamiento cuando los modelos comenzaran a dar signos de *overfiting*.

Todo el proceso de entrenamiento se llevó a cabo en un CPU *Intel Core i-5* con $8GB$ de memoria RAM. Los tiempos varían considerablemente de acuerdo al modelo empleado por eso es en la sección 9 donde se comentará sobre los tiempos de demora de los entrenamientos.

A continuación se detalla el preprocesamiento realizado a los datos para lograr que estos tuvieran un formato acorde para realizar un proceso de aprendizaje adecuado por parte de las redes.

8 Preprocesamiento

La realización de un preprocesamiento es muy importante antes de pasar a la construcción del modelo. El uso de datos sin procesar puede resultar en un

desastre. En esta fase dejaremos fuera todos los símbolos no deseados que no afectan el objetivo de nuestro problema.

En general, las primeras acciones a realizar son las siguientes:

- Convertir todos los caracteres a minúsculas.
- Realizar el mapeo de las contracciones.
- Eliminar los apóstrofos.
- Eliminar todo texto entre paréntesis.
- Eliminar signos de puntuación y caracteres especiales.
- Eliminar *stopwords*

También se desechan los datos que contengan textos o resúmenes vacíos, lo que reduce un poco el tamaño del dataset.

Más del 90% de los datos constan de textos de menos de 30 palabras y de resúmenes con menos de 8 palabras. Es por ello que en se desechan los datos que no cumplen con esta condición ya que sería contraproducente incluirlos en el entrenamiento pues son ejemplos raros.

Luego se realiza el proceso de *tokenización* para transformar en una representación numérica la información textual. Se añaden además los tokens especiales de inicio de resumen y fin de resumen que son necesarios para tener una entrada inicial en el proceso de generación así como una marca final respectivamente.

Por último se procede a extraer del vocabulario las palabras *raras* que son las que se repiten menos que una cantidad determinada a priori de veces. Por supuesto que el umbral para los textos no es el mismo que para los resúmenes, y las palabras que caen en la categoría de raras no son las mismas en ambos casos. Por tanto el vocabulario de los textos no es el mismo que el de los resúmenes, y este último incluye los tokens especiales de inicio y fin de resumen.

Se debe recordar que sobre el vocabulario de los resúmenes es donde se realiza la distribución de probabilidad en cada momento para predecir cuál será la próxima palabra y así generar los nuevos resúmenes de manera automática.

En la siguiente sección se explica cómo fueron construidos los modelos que logran realizar este objetivo.

9 Modelos implementados

En la presente sección se describe la arquitectura de los modelos implementados y sus principales características, así como los tiempos de demora en el entrenamiento de ambos.

A pesar que existen diferencias grandes en ambos modelos, estos comparten un diseño similar que responde al paradigma *seq2seq*.

9.1 Diseño general de los modelos y función de pérdida

En general ambos modelos constan de un codificador y un decodificador unidos por una capa de atención de la cual ya se comentaron sus características así como su necesidad.

El objetivo del decodificador es generar en cada momento la siguiente palabra del resumen (o el final del mismo), por lo que en ambos modelos este es invariante. Toma como entrada su propio estado anterior y la salida de la capa de atención, esa entrada pasa a ser procesada por una capa LSTM para poder asimilar entradas de longitud variable y tener la capacidad de *recordar* estados anteriores, y luego se tiene una capa de distribución de probabilidad temporal que en definitiva calcula la palabra más probable a salir en función de las anteriores y del texto de entrada.

La función de pérdida utilizada fue la *sparse categorical cross-entropy* en lugar de la *categorical cross-entropy* por cuestiones de memoria.

Como se puede apreciar el interés principal está en variar la forma de codificar los textos de entrada, y es precisamente allí donde se realizaron las variaciones más importantes, implementándose dos modelos: uno con un codificador puramente recurrente y otro con un codificador convolucional. A continuación se describen ambos.

9.2 Modelo con codificador puramente recurrente

En el caso de este modelo el codificador consiste en una capa de *embedding* que sirve para aprender representaciones de menor dimensión de la entrada. Luego se tienen tres capas LSTM, una a continuación de la anterior, cada capa, intuitivamente, cumple la misma función de capturar información del texto como un todo; la causa por la que se incluyen tres de estas capas es para aumentar la capacidad del modelo que, de esta forma, puede aprender transformaciones mucho más elaboradas, lo que se hace necesario por la complejidad de la tarea.

El proceso de entrenamiento de este modelo dura, aproximadamente, cuatro horas en las condiciones descritas en la sección 7.

Los resultados son buenos pues se producen resúmenes coherentes y que capturan las ideas claves de manera general. Esta evaluación se realizó en el conjunto de prueba lo que demuestra el buen poder de generalización del modelo.

9.3 Modelo con codificador convolucional

En este caso, luego de la necesasria capa de embedding, en el codificador se sustituyeron las capas recurrentes por capas convolucionales de una dimensión, con excepción de la última que sigue siendo una capa LSTM que es necesaria pues sus estados forman parte de la entrada del decodificador.

Intuitivamente las capas convolucionales capturan ventanas de la secuencia de entrada, lo que les permite aprender patrones como frases comunes y grupos de palabras con significados. De esta manera se pueden codificar determinadas frases importantes con lo que se enriquece contextualmente esta codificación de una manera distinta a como lo hacen las capas LSTM.

Se debe señalar que entre las dos capas convolucionales se añade una capa de *max pooling* que no hace más que seleccionar, de cada grupo de palabras que va procesando la convolucional, la mayor activación. Esto permite filtrar frases más largas cada vez.

Esta vez el proceso de entrenamiento demora solamente veinte minutos y los resultados son incluso más coherentes que los del modelo recurrente puro. Esto es debido al tipo de texto sobre el que se evalúa el cual es procesado de manera más efectiva a partir de convoluciones.

Ambos modelos fueron comparados con un modelo base basado en semántica latente, que se tomó como punto de partida al cual las redes neuronales debían superar para mostrar ser una opción aceptable. A continuación se describe la implementación de este modelo base.

9.4 Modelo basado en Análisis de Semántica Latente (LSA)

Este modelo es un ejemplo de aprendizaje no supervisado. En esencia no es ni generativo ni abstracto pues solamente emplea las propias palabras del texto las cuales agrupa de manera conveniente y extrae para conformar un resúmen. En este caso se divide cada texto en bigramas y estos se toman como las oraciones.

Primeramente se contruye la matriz *término-oración* la cual contiene un 1 en la posición (i, j) si el término j se encuentra en la oración i y 0 en otro caso. Luego se procede a descomponer en valores simples esa matriz con lo que se obtiene una relación de oraciones significativas.

Para este caso se define que existirán a lo sumo dos posibles temas diferentes por lo que el algoritmo puede extraer bigramas que caractericen dos temas y se espera que esto produzca una idea que resuma el texto.

En la práctica, para este tipo de datos, este enfoque no reporta buenos resultados pues, aunque las palabras que extrae son importantes para definir la idea central del texto, estas no se articulan de una manera adecuada y constituyen solamente un grupo de palabras que dan una idea pero no conforman una oración coherente. De ahí que se valoren más los resultados de los modelos supervisados implementados pues estos no solamente son capaces de resumir la idea central del texto sino también generan oraciones con una adecuada sintaxis.

10 Conclusiones

En el presente trabajo se construyeron dos modelos de redes neuronales basados en la arquitectura seq2seq para realizar resúmenes automáticos de textos, especialmente de opiniones sobre productos de comida en idioma inglés.

Ambos modelos logran resultados satisfactorios aunque difieren grandemente en sus tiempos de entrenamiento. Se puede verificar que el modelo convolucional, además de ser más rapido de entrenar, genera resúmenes con mayor coherencia. Esto es debido al tipo de texto con el que se trabaja. Ambos modelos además son capaces de generar resúmenes más elaborados que los que extrae el modelo de semántica latente.

Sería recomendable experimentar usando capas LSTM Bidireccionales que, en teoría, logran una mejor asimilación del contexto aunque añaden más del doble de pesos a aprender. Además explorar un modelo basado en generación de punteros que es una alternativa a los modelos seq2seq.

References

1. Chandola, V., Banerjee, A. and Kumar, V.: Anomaly detection: A survey. In: ACM Computing Surveys (CSUR), Vol. 41, pp. 15, 2009.
2. Akoglu, L., Tong, H. and Koutra, D.: Graph based anomaly detection and description: a survey. In: Data Mining and Knowledge Discovery, Vol. 29, pp. 626–688, 2015.
3. Beyer, K., Goldstein, J., Ramakrishnan, R. and Shaft, U.: When is "nearest neighbor" meaningful?. In: Database Theory?ICDT?99, Springer, 1999.
4. Muller, E., Shez, P. I., Mulle, Y. and Bohm, K.: Ranking outlier nodes in subspaces of attributed graphs. In: Data Engineering Workshops (ICDEW), 2013 IEEE 29th International Conference on Data Engineering, pp.216–222, 2013
5. Knorr, E. M.: Outliers and Data Mining: Finding Exceptions in Data. The University of British Columbia, 2002.
6. Breunig, M. M., Kriegel, H.-P., Ng, R. T. and Sander, J.: LOF: identifying density-based local outliers. In: ACM Sigmod Record, Vol. 29, Number 2, pp. 93-104, 2000.
7. Papadimitriou, S., Kitagawa, H., Gibbons, P. B. and Faloutsos, C.: LOCI: Fast Outlier Detection Using the Local Correlation Integral. In: ICDE, pp. 315-326, 2003.
8. Xiong, Y., Zhu, Y., Yu, P. S. and Pei, J.: Towards Cohesive Anomaly Mining. In: AAAI, 2013.
9. Liu, F. T., Ting, K. M. and Zhou, Z.-H.: Isolation-based anomaly detection. In: ACM Transactions on Knowledge Discovery from Data (TKDD), Vol. 6, Number 1, pp. 3, 2012.
10. Liu, F. T., Ting, K. M. and Zhou, Z.-H.: On Detecting Clustered Anomalies Using SCiForest. In: ECML/PKDD (2), Vol. 6322, pp. 274-290, Springer, 2010.
11. Aggarwal, C. C. and Yu, P. S.: Outlier detection for high dimensional data. In: ACM Sigmod Record, Vol. 30, Number 2, pp. 37–46, 2001.
12. Lazarevic, A. and Kumar, V.: Feature bagging for outlier detection. In: Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining, pp. 157–166, 2005.
13. Akoglu, L., McGlohon, M. and Faloutsos, C.: Oddball: Spotting anomalies in weighted graphs. In: Advances in Knowledge Discovery and Data Mining, pp. 410–421, Springer, 2010.
14. Eberle, W. and Holder, L.: Discovering structural anomalies in graph-based data. In: Data Mining Workshops, 2007. ICDM Workshops 2007., pp. 393–398, 2007.
15. Noble, C. C. and Cook, D. J.: Graph-based anomaly detection. In: Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining, pp. 631–636, 2003.
16. Gao, J., Liang, F., Fan, W., Wang, C., Sun, Y. and Han, J.: On community outliers and their efficient detection in information networks. In: Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining, pp. 813–822, 2010.
17. Moser, F., Colak, R., Rafiey, A. and Ester, M.: Mining Cohesive Patterns from Graphs with Feature Vectors. In: SDM, Vol. 9, pp. 593–604, 2009.
18. Günnemann, S., Farber, I., Boden, B. and Seidl, T.: Subspace clustering meets dense subgraph mining: A synthesis of two paradigms. In: Data Mining (ICDM) 10th International Conference on Data Mining, pp. 845–850, 2010.
19. Akoglu, L., Tong, H., Meeder, B. and Faloutsos, C.: PICS: Parameter-free Identification of Cohesive Subgroups in Large Attributed Graphs. In: SDM, pp. 439–450, 2012.

20. Blondel, V. D., Guillaume, J.-L., Lambiotte, R. and Lefebvre, E.: Fast unfolding of communities in large networks. In: Journal of statistical mechanics: theory and experiment, Vol. 2008, Number 10, pp. P10008, 2008.
21. He, X. and Cai, D. and Niyogi, P.: Laplacian score for feature selection. In: Advances in neural information processing systems, pp. 507–514, 2005.
22. Xu, X., Yuruk, N., Feng, Z. and Schweiger, T. A.: Scan: a structural clustering algorithm for networks. In: Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining, pp. 824–833, 2007.