Neural Information Retrieval

David Guaty Domínguez C512
Adrián Rodríguez Portales C512
Rodrigo D. Pino Trueba C512

June 2022

1 Introducción

A lo largo de la historia de la humanidad, el proceso de almacenar y recuperar la información ha sido una tarea bastante difundida. Sin embargo con el surgimiento de Internet y del *Big Data*, los volúmenes de información actuales son cada vez más grandes y se hace necesario el surgimiento de herramientas más sofisticadas para poder satisfacer estas necesidades. El estudio de esta área se le conoce como Recuperación de Información (*Information Retrieval*). Entre los modelos clásicos utilizados se encuentran el booleano y el vectorial. Ambos modelos presentan ventajas en determinadas situaciones, pero en muchos escenarios no logran cumplir con los requerimientos del sistema.

La aparición del aprendizaje automático (*machine learning*) y ,en especial, del aprendizaje profundo (*deep learning*), ha permitido revolucionar la industria y la ciencia . Los sistemas de recuperación de información no se han quedado atrás. El enfoque conocido como *Learning to Rank* [1] usa técnicas de aprendizaje automático para la tarea de recuperar información. Existen tres formas principales de resolver este problema:

- *pointwise*: Este enfoque es el más sencillo de implementar y fue el primero en proponerse para las tareas de *Learning to Rank*.
- *pairwise*: El problema con el enfoque anterior es que se necesita la relevancia absoluta de un documento con respecto a una consulta. Sin

embargo, en muchos escenarios esta información no estás disponible, entonces lo que se puede saber es, por ejemplo, que documento tiene mayor relevancia de una lista según la selección de un usuario

- *listwise*: Este enfoque es el más difícil, pero también el más directo. A diferencia de los dos anteriores , donde el problema se reduce a una regresión o clasificación, este trata de resolver el problema de ordenación directamente.

Dentro de los modelos de aprendizaje automático más usados recientemente para el proceso de *ranking*, se encuentran las redes neuronales. A la aplicación de este conjunto de técnicas en la recuperación de información se le conoce como *Neural Information Retrieval* [3]. En este trabajo usaremos este enfoque y compararemos los resultados con un modelo de recuperación de información clásico (vectorial).

2 Diseño del sistema de recuperación de información

La aplicación cuenta con un interfaz visual que permite acceder a la API interna. Se cuenta actualmente con 5 definciones de modelos:

- Modelo Vecotrial
- Learn to Rank implementado como Clasificador y Regresor
- Learn to Rank utilizando Redes Recurrentes implementado como Clasificador y Regresor

2.1 Preprocesamiento de los datos

Los dataset utilizados fueron preprocesados para obtener aquellas palabras que tienen mayor semántica. Durante este proceso se realizaron dos tareas:

- Eliminación de *stopwords*:los *stopwords* son términos muy comunes que brindan poca información semántica en un texto o documento. Generalmente suelen ser adverbios, pronombres y artículos.

- *Lemmatization*: este proceso consiste en llevar a las palabras a su raíz gramatical, haciendo análisis morfológico de los términos y eliminando cualquier inflexión. Esto permite indexar términos con igual semántica como uno solo .

Ambas tareas se hicieron con la biblioteca de Python *Spacy*

2.2 Modelo vectorial

El modelo vectorial[2] es un modelo algebraico que representa los documentos y la consulta como vectores con pesos. Cada término indexado representa una dimensión del vector, por lo que el vocabulario del corpus es quien define la dimensión del espacio vectorial.

Para el cálculo de los pesos existen varias formas. Una de las más usadas es *tf-idf*. Esta tiene entre sus ventajas que es fácil de computar. Sin embargo, como está basado en el modelo *bag of words*, no tiene en cuenta el orden de las palabras en el texto.

Para calcular el *tf* se utilizó la siguiente fórmula:

$$tf_{ij} = K + \frac{(1-K)freq_{ij}}{\max_i freq_{ij}}$$

En el caso del *idf* se utilizó:

$$idf_i = \log \frac{N}{n_i}$$

donde N es el total de documentos en el corpus y n_i es la cantidad de documentos donde aparece el término i. Luego con estos valores los pesos se obtenían como $w_{ij} = tf_{ij} * idf_{ij}$.

Finalmente para obtener la similitud entre la consulta y los documentos se usó el coseno del ángulo :

$$sim(d_j, q) = \frac{d_j \cdot q}{|d_j||q|}$$

2.3 Modelo con redes neuronales

Para determinar qué tan relevante es un documento para una consulta dada se implementaron modelos de redes neuronales. En total, en la aplicación existen cuatro de estos modelos.

La función del modelo de red neuronal es determinar, para una consulta q y un documento d, un ranking $N(q, d)$, donde N es la red neuronal. Este valor de ranking varía de acuerdo al conjunto de datos con el cual el modelo se entrenó. Por ejemplo, en el conjunto de datos *Cranfield* existen los niveles de relevancias $\{-1, 1, 2, 3, 4\}$, los cuales se mapearon a $\{0, 1, 2, 3, 4\}$. Un valor de relevancia 0 significa que el documento d es totalmente irrelevante a la consulta q y un valor de relevancia de 4 significa que el documento es una total respuesta a la consulta por el usuario.

Entonces se tienen un conjunto de datos de entrenamiento que tiene la forma $(documento, consulta, relevancia)$, es el trabajo de la red neuronal aprender estas relaciones.

Los cuatros modelos son:

- Un clasificador y un regresor mediante redes convolucionales
- Un clasificador y un regresor mediante redes recurrentes con capas LSTM

Investigaciones recientes han empleado redes neuronales convolucionales o redes neuronales recurrentes para la clasificación de textos motivados por el notable éxito del aprendizaje profundo [4].

Para la implementación de los modelo se utilizó keras debido a sus facilidades para el preprocesamiento de texto y para elaborar arquitecturas de manera sencilla.

Para cada uno de lo tipos de redes, las arquitecturas del clasificador y el regresor son en esencia las mismas; el cambio fundamental es en la capa de salida y en la función de pérdida.

Dicho esto se explica la arquitectura general seguida para la red convolucional:

- Capa de Embedding: Para representar los vectores de documento y consulta en un espacio vectorial continuo de dimensión fija pequeña. Esto es útil pues aumenta el rendimiento con respecto al procesamiento de lenguaje natural y puede capturar algo de la semántica de la palabra. Para el embedding se usó un modelo preentrenado: GloVe. Mediante glove se puede obtener la matriz de la capa del embedding de la red.

```
embeddings_index = {}
with open(path_to_glove_file) as f:
        for line in f:
```

```
            word, coefs = line.split(maxsplit
                =1)
            coefs = np.fromstring(coefs, "f",
                sep=" ")
            embeddings_index[word] = coefs
    print("Found %s word vectors." % len(
        embeddings_index))
```

- Capa de convolución
- Capa(s) densamente conectadas
- Capa de salida

En código, generalmente el modelo convolucional es de la siguiente manera:

```
input_doc = keras.Input(shape=(None,), dtype="int64")
input_query = keras.Input(shape=(None,), dtype="int64")

embedding_layer = Embedding( #GloVe
        num_tokens,
        embedding_dim,
        embeddings_initializer=keras.initializers.
            Constant(embedding_matrix),
        trainable=False,
)

embedded_sequences = embedding_layer(input_doc)
x = layers.Conv1D(128, 5, activation="relu")(
    embedded_sequences)
x = layers.GlobalMaxPooling1D()(x)


embedded_sequences = embedding_layer(input_query)
y = layers.Conv1D(128, 5, activation="relu")(
    embedded_sequences)
y = layers.GlobalMaxPooling1D()(y)
```

```
combined = layers.Concatenate()([x, y])

# Para clasificador
z = layers.Dense(16, activation="relu")(combined)
z = layers.Dense(number_of_relevance_levels, activation
    ="softmax")(z)

#Para regresor
z = layers.Dense(16, activation="relu")(combined)
z = layers.Dense(1)(z)
```

La arquitectura de la red recurrente tiene la siguiente forma:

- Una capa de Embedding, anteriormente mencionada.
- Una capa LSTM bidireccional: Esta capa propaga su entrada hacia adelante y hacia atrás y luego concatena la salida final.
- Capas densamente conectadas: Después de que la capa anterior haya convertido la secuencia en un solo vector, estas capas realizan un procesamiento final y convierten esta representación vectorial en un solo logit como salida de clasificación o regresión.

En código, generalmente el modelo recurrente es de la siguiente manera:

```
embedding = tf.keras.layers.Embedding(
  input_dim=len(encoder.get_vocabulary()),
  output_dim=64,
  mask_zero=True,
)

input_doc = keras.Input(shape=(None,), dtype="int64")
input_query = keras.Input(shape=(None,), dtype="int64"

doc_embedding = embedding(input_doc)
query_embedding = embedding(input_query)

doc_bid = tf.keras.layers.Bidirectional(
  tf.keras.layers.LSTM(64, return_sequences=True)
)(doc_embedding)
```

```
query_bid = tf.keras.layers.Bidirectional(
  tf.keras.layers.LSTM(64, return_sequences=True)
)(query_embedding)

x = keras.layers.Concatenate()([doc_bid, query_bid])

x = tf.keras.layers.Dense(64, activation="relu")(x)
x = tf.keras.layers.Dropout(0.5)(x)

output = keras.layers.Dense(1)(x)
```

2.4 Ventajas y desventajas de las redes neuronales

A menudo, es posible realizar un entrenamiento de extremo a extremo en aprendizaje profundo para una aplicación, lo que quiere decir que se pueden diseñar modelos cuya entrada sea el texto en lenguaje natural introducido por el usuario. Esto se debe a que estos modelos ofrecen una rica representabilidad y la información en los datos se puede codificar de manera efectiva en el modelo.

Como ventajas de las redes neuronales para el procesamiento de texto se tiene:

- Puede reconocer patrones en el texto
- Puede aprender representaciones del texto, como por ejemplo los embedding mencionado
- Mejora su rendimiento cuanto más datos existan
- Captura mejor la semántica del texto que otros métodos

En las tareas de procesamiento de texto mediante redes neuronales existe una falta de fundamento teórico, así como falta de interpretabilidad del modelo y el requisito de una gran cantidad de datos y recursos computacionales potentes.

Como desventajas se tienen:

- Se requiere grandes cantidades de datos para ser entrenadas

- Dependiente de hardware en su eficiencia para entrenarse
- Los modelo de redes neuronales son cajas negras difíciles de analizar
- Si el conjunto de entrenamiento no esta bien depurado pueden producirse errores en el aprendizaje y no generalizar como se quiere

2.5 Problemas encontrados en entrenamiento

Al entrenar las redes neuronales mediante el conjunto de datos de Cranfield se encontró con el problema del overfitting. Los modelos aprenden muy bien el conjunto de entrenamiento pero tienen bajos rendimiento en el conjunto de validación. En la figura 1 se puede observar el diagnóstico del entrenamiento de la red y como se produce el overfitting.

Para tratar de solucionar este problema se intentó:

- Simplificar el modelo poniendo menos capas intermedias y con menos neuronas
- Añadir capas de Dropout
- Para el entrenamiento cuando las dos curvas mostradas en la figura 1 empiezan a divergir
- Cambiar la función de activación en las capas intermedias
- Aumentar la dimensión del embedding

Lamentablemente ninguna de estas vías solucionó el problema del overfitting, por lo que es recomendable usar otro conjunto de datos como corpus para entrenar los modelos.

3 Análisis de los resultados

Los resultados se encuentran lejos de ser satisfactorios. Hubo muy bajo nivel de recobrado y precisión en todos los modelos, no existe una diferencia notable entre estos. La precisión se encuentra sobre los *0.1* mientras que el recobrado está sobre el *0.07*. Vale notar que todos los entrenamientos y pruebas se realizaron sobre el *Cranfield* que representa un dataset díficil de recuperar.

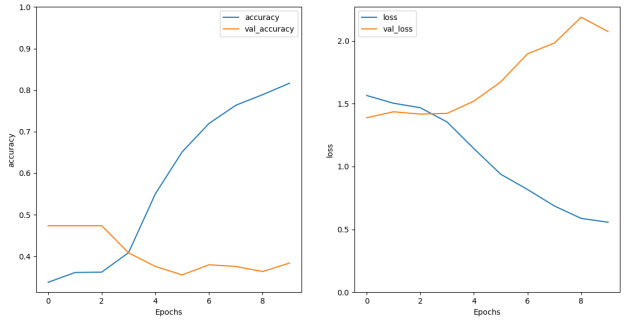

Figure 1: Overfitting encontrado en entrenamiento

Nunca se logró un accuracy mayor que 0.4 en el set de validación, incluso después de reducir las épocas en un intento futil de disminuir el overfitting.

Relevance Metric	Score
Precission	0.1
Relevance	0.07
F1 Score	0.08

4 Conclusiones

En este trabajo se implementó un sistema conocido como *Learning To Rank* en el cual se usan redes neuronales para predecir la relevancia que tiene un documento dada una consulta del usuario. Tratar de hallar la relevancia para todos los documentos del corpus mediante la red neuronal es muy costoso, por lo que se implementó un modelo vectorial que da un conjunto de documentos sobre los cuales la red neuronal halla las relevancias. En el entrenamiento de las redes neuronales se encontró el problema del overfitting el cual no pudo ser resuelto por las técnicas intentadas.

Se recomienda la búsqueda de mas conjuntos de entrenamiento que tengan mayor cantidad de datos y analizar si se produce overfitting en estos casos. También se podrían implementar otras arquitecturas de redes neuronales, como por ejemplo Transformers.

References

[1] Learning to rank: A complete guide to ranking using machine learning — by francesco casalegno — towards data science.

[2] Vector space model - wikipedia.

[3] B. Mitra and N. Craswell. An introduction to neural information retrieval. *Foundations and Trends® in Information Retrieval*, 13(1):1–126, December 2018.

[4] H. Wu and S. Prasad. Convolutional recurrent neural networks for hyperspectral data classification. *Remote Sensing*, 9(3):298, 2017.