ISSN 1870-4069

Sistemas de clasificación aplicados a la detección de paráfrasis

Francisco Fernando López Ponce,
Gerardo Sierra, Gemma Bel Enguix

Universidad Nacional Autónoma de México,
Instituto de Ingeniería, Grupo de Ingeniería Lingüística,
México

francisco.lopez.ponce@ciencias.unam.mx,
{gsierram, gbele}@iingen.unam.mx

Resumen. En este artículo analizaremos una de las múltiples tareas del PLN que es la clasificación de textos, en particular la clasificación binaria aplicada a la detección de paráfrasis. Implementamos una variedad de modelos de clasificación basados en el aprendizaje automático, desde modelos tradicionales como la regresión logística o Naive Bayes, hasta el estado del arte que son modelos de redes neuronales usando la arquitectura Transformer. Hacemos uso de distintas representaciones matemáticas del texto para realizar el preprocesamiento adecuado previo a la implementación de los modelos. Para el proceso de entrenamiento y evaluación usamos los datasets MRPC, donde los modelos se terminan por evaluar usando las métricas estándar para problemas de clasificación binaria como son Accuracy, Precision, Recall y F1, obteniendo una F1 máxima de 0.89.

Palabras clave: PLN, paráfrasis, redes neuronales, transformers, clasificación binaria.

Classification Systems Applied to Paraphrase Detection

Abstract. In this paper we analyze the NLP task of text classification, in particular binary classification applied to paraphrase detection. We implemented a myriad of classification models based on machine learning, from traditional models like logistic regression or Naive Bayes to state-of-the-art neural models based on the Transformer architecture. For feature extraction we use multiple mathematical representations of text as well as adequate preprocessing. When it comes to training and testing we used the MRPC as well as standard classification metrics such as Accuracy, Precision, Recall, and F1, obtaining a maximum F1 score of 0.89.

Keywords: NLP, paraphrase, neural networks, transformers, binary classification.

1. Introducción

La paráfrasis es el fenómeno que describe la similitud entre distintos textos que han sido modificados mediante sustituciones léxicas o reformulaciones estructurales [22], decimos que un texto es paráfrasis del otro si después de estas modificaciones se preserva el contenido dentro de los textos. Podemos observar ejemplos de este fenómeno en los pares 1.1, 1.2, y 2.1, 2.2.

1.1 El gato come feliz.

1.2 El minino come alegre.

1.3 El niño camina velozmente.

1.4 Rápidamente anda el infante.

Este fenómeno puede analizarse de manera profunda desde el ámbito lingüístico al definir distintos tipos de modificaciones que se hacen a un texto para llegar a una nueva instancia parafraseada. Si bien estas modificaciones, generalmente clasificadas como omisión, sustitución, modificación morfológica, y modificación en el orden, son herramientas que permiten analizar el contenido lingüístico de la paráfrasis, no presentan las mejores herramientas a la hora de implementar modelos computacionales que busquen analizar este fenómeno.

A nivel computacional este problema se trabaja como uno de clasificación binaria en donde un sistema computacional debería de ser capaz de determinar la existencia (o falta) de este fenómeno al comparar pares de textos. Generalmente este problema se simplifica trabajando con pares de oraciones para facilitar la creación e implementación de modelos entrenados con este tipo de datos.

La detección automática de paráfrasis es un tema de estudio de alta relevancia debido a sus aplicaciones como la detección de estilo o detección de plagio. En este artículo presentaremos una implementación de múltiples modelos computacionales que hacen uso del PLN y el aprendizaje automático para enfrentarse a este problema. Estos modelos nos terminan por generar un sistema de clasificación basado en diversas representaciones matemáticas del texto y algoritmos de aprendizaje automático.

Implementaremos modelos clásicos del aprendizaje, como los clasificadores a partir de regresión logística y Naive Bayes, modelos más avanzados como redes neuronales recurrentes con diferentes arquitecturas y capas, hasta el estado del arte que hacen uso de modelos pre-entrenados a partir de Transformers. En el capítulo 2 hablaremos de trabajos relacionados con la detección de paráfrasis, en el 3 describiremos a nivel matemático los modelos implementados, mientras que en el 4 desarrollaremos la implementación puntual de dichos modelos; concluimos en el 5 presentando los resultados, comparándolos entre ellos, y planteamos posibles trabajos a futuro.

2. Trabajos relacionados

Autores como Meshram [13] recopilan de manera superficial diversos métodos y artículos correspondientes a sistemas de detección automática. De una manera similar Mohamed [16] recopila y presenta modelos de aprendizaje automático divididos entre

los sistemas clásicos y los de aprendizaje profundo donde nos muestra los resultados de algunos modelos particulares. Por otra parte existen una plétora de artículos donde se presentan distintos modelos con arquitecturas y representaciones matemáticas del texto evaluados bajo el corpus de paráfrasis de Microsoft (MRPC). Al inicio de la década pasada Ji [9] presenta un modelo que logra optimizar el uso de sistemas estadísticos usando variaciones de tf-idf, evaluándose con el corpus MRPC y obteniendo Accuracy de 80 % y una F1 de 85 %.

Mientras que trabajos como el de Vrbanec [23] comparan modelos de embeddings como Word2Vec, GLoVE, o ELMO, obteniendo una F1 máxima de 81 % en el mismo corpus. Por otra parte el inicio de ésta década presenta modelos como Arase [1], un fine-tuning que permite la convergencia rápida de un modelo pre-entrenado con BERT [4].

Debido a la creación de distintos modelos pre-entrenados tenemos artículos que implementan y evalúan modelos como RoBERTa [12], XLM-R [2] y ALBERT [11], como lo hace Corbeil [3] haciendo ajustes al MRPC por medio del aumento de datos y obteniendo un F1 tan alto como 91 %.

La cercanía aparente entre los modelos estadísticos y los modelos neuronales, al igual que la constante creación de modelos pre-entrenados especializados, nos motiva a presentar implementaciones de distintos modelos al igual que distintas caracterizaciones matemáticas del texto, para ser entrenados y evaluados usando el MRPC.

3. Modelos

Al hablar de implementaciones de modelos de aprendizaje automático dentro del contexto del PLN tenemos que hacer una diferencia importante entre los algoritmos de clasificación y las representaciones matemáticas del texto. El aprendizaje automático se enfoca en encontrar tendencias numéricas a partir de múltiples algoritmos de enseñanza que hacen uso de optimizadores y funciones de pérdida para eficientizar su rendimiento, como contraste en el caso del PLN no se trabaja, inicialmente, con datos numéricos por lo que es necesario encontrar métodos de representación matemática del texto tales que las propiedades léxicas y morfológicas se preserven, así permitiendo que los resultados de estos modelos de aprendizaje se presten a una interpretación adecuada a nivel lingüístico. Presentaremos primero las representaciones matemáticas del texto.

3.1. Representación matemática del texto

Las representaciones que veremos obtienen desde valores numéricos hasta espacios vectoriales a partir de un texto, podemos notar que algunas representaciones sencillas necesitan solo un par de textos para implementarse mientras que algunas más complejas requieren un conjunto de textos (corpus) completo. Estos objetos matemáticos obtenidos son las características que vamos a usar a la hora de entrenar nuestro modelo.

Es importante notar que no todas las representaciones son compatibles con todos los distintos modelos de clasificación, como veremos, existen casos de representaciones que se crean a partir de un modelo particular y cuya implementación va de la mano con el modelo base.

Coeficientes y métricas. Los coeficientes de Dice y Jaccard, desarrollados originalmente en el contexto de la estadística, se prestan a una fácil aplicación al contexto del PLN, mientas que las distancias de edición (edit distances) son una familia de métricas sobre hilos que nos permiten obtener un valor de similitud a partir de un par de oraciones.

Para los primeros coeficientes consideremos A y B dos oraciones expresadas como conjuntos donde cada palabra de cada oración corresponde a un elemento del conjunto (existen implementaciones que consideran las letras y signos de puntuación en vez de las palabras como los elementos de los conjuntos, pero la definición es análoga), entonces:

– Definimos el **Coeficiente de Dice** como:

$$D(A,B) = \frac{2|A \cap B|}{|A| + |B|}. \tag{1}$$

– Definimos el **Coeficiente de Jaccard:** como:

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|}. \tag{2}$$

El concepto de distancia de edición ya es propio del PLN, busca determinar la similitud de dos hilos u oraciones en función de la cantidad de modificaciones que se deben de aplicar sobre el primero para obtener el segundo, estos cambios son inserción, eliminación y substitución.

A cada cambio se le puede otorgar un peso dependiendo de la complejidad lingüística, aunque normalmente se maneja con el mismo peso para todos los cambios. Si bien existen distintas variaciones que dependen de las modificaciones permitidas, normalmente se usa la distancia de Levenshtein [10] (mínima distanca de edición) al ser la más permisiva con pesos iguales para cada modificación.

Frecuencia de términos. Los siguientes modelos presentan una complejidad mayor debido a la influencia de la hipótesis distribucional del lenguaje, resumida elegantemente por la frase del lingüísta J. R. Firth: “conocerás una palabra por la compañía que tenga” [5]. Estos modelos generan un espacio vectorial en función de la frecuencia de las palabras dentro de un corpus para describir similitudes entre palabras y textos.

La representación vectorial base corresponde a la frecuencia de términos (tf). Dado un corpus C y un conjunto de palabras P, correspondiente al vocabulario del corpus, creamos una matriz $A \in M_{|P|x|C|}$ donde la entrada $x_{p_i c_j}$ corresponde a la cantidad de veces que se usa la palabra p_i en el texto c_j. Esta matriz nos permite crear dos vectores, considerando columnas obtenemos un vector característico del texto c_j en función del vocabulario P, mientras que considerando filas obtenemos un vector característico de la palabra p_i en función de los textos C [10].

Un problema con este modelo radica en que la frecuencia de las palabras es un valor sesgado que no encapsula el significado completo de una palabra. Además, debido a que los idiomas siguen la distribución de Zipf [24] con respecto al uso de palabras, existen casos con una frecuencia extremadamente alta como son los artículos, mientras otras que pueden aparecer una o dos veces en todo el corpus, esto genera representaciones vectoriales sesgadas a favor de aquellas palabras con alta frecuencia.

Para arreglar este problema se recalculan las entradas de cada vector en función de la cantidad de documentos en los que aparece dicha palabra con la finalidad de darle mayor peso a palabras exóticas y minimizar el impacto de palabras demasiado comunes, a este valor se le conoce como frecuencia inversa de documento o idf por sus siglas en inglés.

A este proceso se le llama peso tf-idf (tf-idf weighing) [19]. Una vez que obtenemos una representación vectorial de nuestro vocabulario calculamos similitudes entre palabras a partir de técnicas matemáticas, generalmente usando la distancia coseno entre vectores para determinar similitudes.

Formalmente consideremos una palabra p, un documento d, N la cantidad de documentos en el corpus, y df_t la cantidad de documentos en donde se encuentra la palabra t, entonces definimos la frecuencia de términos tf, y la frecuencia inversa de documentos idf como:

$$\mathrm{tf}_{p,d} = \log_{10}(\text{cantidad}(p, d) + 1), \tag{3}$$

$$\mathrm{idf}_t = \log_{10}\left(\frac{N}{df_t}\right). \tag{4}$$

Por lo que la entrada de cada vector en un modelo tf-idf está dada por:

$$c = tf_{p,d} \cdot idf_t. \tag{5}$$

Naturalmente, solo podemos comparar vectores del mismo tipo (texto con texto, y palabra con palabra) ya que no hay garantía de que las matrices $M_{|P|x|C|}$ sean cuadradas. Estas representaciones poseen ventajas sobre su contraparte estadística ya que permiten contextualizar las palabras y minimizan sesgos presentes en la variada frecuencia del uso de palabras, además con implementaciones computacionales podemos analizar textos de mayor longitud y complejidad.

No obstante estos modelos tienen limitaciones ya que un vocabulario puede generar vectores cuyas dimensiones sean demasiado altas, por ejemplo el corpus Brown [6] tiene aproximadamente un millón de palabras. Esto genera problemas computacionales y vectores con escasa información, una palabra que aparece una única vez va a ser representada por un vector de un millón de dimensiones que tiene una sola entrada no cero.

Word Embeddings. La idea subyacente de esta representación es similar a la de los modelos previos: crear un espacio vectorial que capture las similitudes lingüísticas de las palabras. Las diferencias radican en los algoritmos usados para crear los espacios, las dimensiones, y la eficiencia de los mismos. El primer modelo de word embeddings fue Word2Vec (w2v) [15] que en vez de usar frecuencia y pesos para ajustar los vectores, considera una palabra y una ventana de contexto sobre la palabra donde aplica

regresión logística, entrenándose con contexto real y contexto falso, este último creado de manera aleatoria. Esto hace que w2v no analice la cercanía de las palabras si no las probabilidades de que dada una palabra base a y una palabra objetivo b, b sí pertenezca al contexto de a.

Consideremos la oración .ª mí me gusta el pan con café y frutas", sea p = pan la palabra base, con ventana de contexto 2, y c_j las palabras dentro de la ventana del contexto de p. Obtenemos 4 ejemplos positivos de pares de palabras, mientras que creamos 4 ejemplos negativos, n_j, de manera aleatoria:

- $c_1 = (\text{pan}, \text{gusta})$,
- $c_2 = (\text{pan}, \text{el})$,
- $c_3 = (\text{pan}, \text{con})$,
- $c_4 = (\text{pan}, \text{cafe})$,
- $n_1 = (\text{pan}, \text{automovil})$,
- $n_2 = (\text{pan}, \text{musica})$,
- $n_3 = (\text{pan}, \text{ellos})$,
- $n_4 = (\text{pan}, \text{goloso})$.

A partir de conjuntos similares w2v logra maximizar la similitud de la palabra base y objetivo (p, c_j), y minimizar la similitud entre la palabra base y los ejemplos negativos (p, n_j). Los word embeddings logran encapsular el contexto de uso de las palabras, además de agruparlas en función de su similitud individual y de contexto global, por ejemplo palabras como pan, café, o desayuno, se encuentran en una vecindad cercana aunque su significado individual difiera.

Esto nos permite encontrar analogías de manera matemática, es decir operaciones como $'\text{Paris}' - '\text{Francia}' + '\text{Italia}' = '\text{Roma}'$, o $'\text{Rey}' - '\text{Hombre}' + '\text{Mujer}' = '\text{Reina}'$ tienen una consistencia y veracidad alta [14]. Existen múltiples modelos de embeddings que optimizan y ajustan distintos parámetros y que sobrepasan a w2v; sin embargo, están sujetos a los problemas inherentes del modelo.

Para empezar los vectores son estáticos es decir cada palabra está asignada a un valor único, generando un conflicto con la polisemia, así mismo la arquitectura los embeddings genera conflictos al analizar palabras fuera del vocabulario del corpus de entrenamiento, volviéndolo dependiente de las propiedades del corpus.

Además los embeddings son muy propensos a aprender sesgos dentro del corpus, sesgos sexistas o racistas se pueden observar en la inconsistencia de operaciones entre palabras al cambiar el género, o las cercanías que llegan a tener grupos sociales particulares con palabras negativas [25].

3.2. Modelos de aprendizaje automático

Hablaremos brevemente sobre los modelos de aprendizaje automático. Describiremos la forma en la que los modelos estadísticos analizan un valor de entrada (input) y lo clasifican, explicaremos distintas capas de redes neuronales recurrentes y acabaremos analizando el modelo Transformer.

La descripción que damos de estos modelos hace énfasis en las formas de clasificar un valor, por lo que evitamos hablar del proceso de entrenamiento y ajuste de pesos ya que se desvía del tema central del artículo y, si bien es sumamente importante para el aprendizaje automático, no es vital para el análisis de la paráfrasis.

Naive Bayes. Este es un clasificador probabilístico entonces dada una entrada i y un conjunto de clases objetivo C, Naive Bayes (NB) calculará la probabilidad de que dicho input pertenezca a cada una de las clases posibles y regresará la clase con mayor probabilidad. En términos matemáticos la clase $c \in C$ que va a regresar NB es:

$$c = \underset{c_j \in C}{\text{máx}}\ P(c_j|i), \tag{6}$$

Lo interesante de NB es el manejo que hace de la probabilidad (4), y una supusición bastante fuerte durante el manejo algebráico de la misma, primero expresa $P(c_j|i)$ en términos de la regla de Bayes para probabilidades condicionales, luego como i es fijo y los valores son positivos podemos ignorar el denominador, y considerando $i = \alpha_1, \alpha_2, ..., \alpha_k$ en función de sus características llegamos a la expresión:

$$P(c_j|i) = \frac{P(i|c_j)P(c_j)}{P(i)} = P(i|c_j)P(c_j) = P(\alpha_1, \alpha_2, ..., \alpha_k|c_j)P(c_j), \tag{7}$$

NB obtiene su nombre a partir de la suposición ingenua (naive) de que las probabilidades condicionales $P(\alpha_l|c_j)$ son todas independientes. Por último, para evitar problemas computacionales con valores numéricos muy grandes se ajusta aplicando el logaritmo:

$$c = \underset{c_j \in C}{\text{máx}}\ P(c_j) \prod_{\alpha_l \in \alpha} P(\alpha_l|c_j) = \underset{c_j \in C}{\text{máx}}\ \log P(c_j) \sum_{\alpha_l \in \alpha} \log P(\alpha_l|c_j), \tag{8}$$

Regresión logística. Si bien este modelo sigue siendo un clasificador probabiístico en contraste con NB la regresión logística (RL) no busca caracterizar las clases en función de las probabilidades de pertenencia, más bien busca encontrar elementos que diferencien a las clases sin necesariamente aprender las propiedades particulares de cada clase.

La RL no trabaja en un inicio con probabilidades, al principio del entrenamiento genera un vector de pesos a partir del conjunto de aprendizaje, el peso se asocia al input y determina la relevancia dentro del problema para cada característica. Se genera una combinación lineal del vector de pesos ($\mathbf{p}$), el input ($\mathbf{i}$) y el factor de sesgo (s) para posteriormente calcular la probabilidad de clasificación dado un input:

$$z = (\mathbf{p} \cdot \mathbf{i}) + s, \tag{9}$$

Como la RL es probabilística necesitamos asegurarnos que $z \in (0, 1)$, para esto se aplica la función logística sobre z y, dependiendo de la cantidad de clases sobre las que queramos clasificar, ajustamos haciendo softmax:

$$P(y = 1|i) = \sigma(z) = \frac{1}{1 + e^{(-z)}} = \frac{1}{1 + e^{(-((\mathbf{p} \cdot \mathbf{i}) + s)}}, \tag{10}$$

En el caso de la clasificación binaria no es necesario ajustar con softmax ya que podemos encontrar las dos expresiones necesarias para las dos clases recordando que $1 - \sigma(x) = \sigma(-x)$ y que $P(y = 1) + P(y = 0) = 1$, por lo que:

$$P(y = 0) = 1 - \sigma(z) = 1 - \frac{1}{1 + e^{-z}} = \frac{e^{(-((\mathbf{p}\cdot\mathbf{i})+s)}}{1 + e^{(-((\mathbf{p}\cdot\mathbf{i})+s)}}, \tag{11}$$

Ya que tenemos las probabilidades determinamos un umbral de clasificación y clasificamos en función de esto, si bien el umbral estándar es 0.5 se puede modificar si se logra mejorar el rendimiento del modelo.

Redes neuronales. Se plantearon a la par que NB y RL, pero no fue hasta la última década, con la inovación de los Transformers, que las redes neuronales han logrado superar a los modelos estadísticos. Una red neuronal está compuesta por unidades neuronales sencillas (neural units) conectadas donde, dado un valor de entrada, se realizan operaciones, se produce un valor de salida, y se conecta con otra unidad para ser modificada hasta obtener un valor final tras recorrer toda la red.

El modelo más sencillo es una red prealimentada (feedfoward network), es decir una red donde las capas no generan ciclos entre ellas, y donde todas las unidades entre capas adyacentes están conectadas. Sin embargo nosotros usaremos modelos más complejos como son los recurrentes que usan ciclos entre capas, permitiendo múltiplies ajustes de un valor en una misma capa y a la preservación de información a lo largo del procesamiento.

Otro modelo que usaremos surge a partir del concepto de atención, un mecanismo que logra optimizar el proceso recurrente en función de las características más relevantes de las entradas, este modelo es el Transformer y los modelos preentrenados a partir de esta arquitectura.

Las unidades sencillas que crean estas redes se comportan de una manera similar a la RL, cada unidad posee un vector de pesos y un sesgo, en combinación con el input se hace una combinación lineal como en (7), posteriormente se aplica una función no lineal al valor obtenido, como en (8), esta función no necesariamente tiene que ser la logística, ejemplos de dos funciones de activación distintas son:

$$\tanh(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}}, \tag{12}$$

$$\mathrm{ReLU}(z) = \mathrm{máx}(z, 0). \tag{13}$$

La agrupación de estas unidades en capas permite un mejor rendimiento, una red neuronal está formada por 3 tipos de capas, el primero es la capa de entrada donde se presentan los valores iniciales, el segundo son las capas ocultas donde se realiza la mayoría del procesamiento (una red puede tener múltiples capas ocultas), y finalmente la capa de salida donde se obtiene el resultado final que dependerá del problema en cuestión, para clasificación binaria obtendremos un valor probabilístico de pertenencia en una clase u otra. Las redes recurrente permiten revisitar una capa durante el procesamiento agregando un factor temporal a la forma de analizar un input, esto permite almacenar memoria sobre estados previos, es decir información previa afecta los cálculos inmediatos.

Estas redes son capaces de apilar múltiples capas para mejorar su rendimiento, si bien puede aumentar el tiempo de entrenamiento, este análisis tiende a entender mejor el input en distintos niveles de profundidad.

Además las redes recurrentes optimizan la apilación de capas gracias a la bidireccionalidad, es decir dado un input apilamos dos capas recurrentes donde la segunda capa procesa el input en sentido inverso, el valor de salida está dado por una operación sobre los outputs de cada entrada de la capa bidireccional.

[18] Aunque las redes recurrentes permiten un manejo creativo de la información, son susceptibles al problema del desvanecimiento del gradiente ya que las operaciones dentro del entrenamiento pueden llevar a que el gradiente tienda a cero, como ajuste se usan capas llamadas Long Short-Term Memory (LSTM) [8]. Una capa LSTM es capaz de discernir entre información que no será relevante en un futuro (y descartar la misma), además de preservar la información que sí se considera relevante al igual que adecuarla para usos futuros.

Se logra esto mediante el uso de compuertas que permiten el flujo de información entre capas, llamadas compuertas de olvido, suma, y salida (forget gate, add gate, output gate), estas compuertas clasifican la información en función de operaciones secuenciales y funciones de activación como $\tanh$, generando como salida un vector que se usará en futuras capas al igual que un vector de contexto (VC). LSTM no solo presenta beneficios conceptuales si no que logra resolver el problema del gradiente, por lo que estas capas normalmente se usan sobre la arquitectura base de una red recurrente.

Como buen modelo matemático las redes recurrentes presentan fallas, en particular a la hora de implementarse en modelos codificador-decodificador (encoder-decoder) [20], ya que el vector de contexto obtenido por el codificador no resume de manera óptima la información de todo un input, además la conexión entre estas dos secciones se ve ralentizada por la unicidad de este vector.

La solución a estos problemas son los mecanismos de atención que permiten más conexiones entre el encoder y decoder generando un acceso continuo al VC de cada capa oculta, y evitando el sesgo proveniente del VC final ya que en cada paso se logra priorizar el contexto más relevante para el problema.

Modelos Preentrenados. A partir de la atención se crea el modelo Transformer [21] y como consecuencia los modelos preentrenados. Los mecanismos de atención permiten comparar elementos y determinar sus niveles de relevancia dado un contexto, los Transformers son capaces de replicar esto sobre un solo input al tener acceso a esta relevancia de cada valor del input desde el inicio hasta el punto en curso, a esta variación se le conoce como auto-atención (self-attention).

Es importante notar que un Transformer no es una red recurrente con auto-atención, estos modelos están compuestos por bloques Transformer (Transformer Blocks) que hacen uso de capas con auto-atención, capas prealimentadas, y capas de ajuste para optimizar el entrenamiento.

Además al trabajar con múltiples bloques con distintos parámetros es posible que un modelo logre aprender diferentes relaciones sin la necesidad de aumentar las dimensiones de una red. Lo increíble de los Transformers radica en que entre más información esté disponible para su entrenamiento, mejores van a ser las implementaciones del modelo.

Tabla 1. Parámetros de regresión logística.

Métrica	C	Penalty	Solver
Dice	1	None	saga
Jaccard	1	None	LBFGS
Levenshtein	1	None	saga
tf	1	None	saga
tf-idf	100	None	saga
Multi	10	L1	saga

Esto introduce la idea de los modelos preentrenados, que radica en entrenar un Transformer con una alta cantidad de información, en el caso del PLN textual, y una vez obtenido el modelo final hacer un segundo entrenamiento con un corpus más específico al problema en cuestión.

Recordando los modelos codificador-decodificador existen distintas formas de preentrenar un modelo de PLN dependiendo del objetivo que se busque, si se busca dar una representación del texto, dígase un word embedding, se usa un modelo encoder, en caso de que se busque generar texto se usa un modelo decodificador, y en caso que se quiera hacer las dos tareas podemos usar los modelos codificador-decodificador. BERT es un codificador, GPT (en sus múltiples versiones) es un decodificador, y T5 es un modelo mixto.

Las ventajas del preentrenamiento es que se necesita entrenar una sola vez para obtener un modelo base que rinde de manera eficiente en una gran variedad de problemas de PLN, estos modelos son ajustados dependiendo del problema que se esté tratando de resolver. Nosotros haremos este ajuste para la detección de paráfrasis.

4. Implementación

Primero describiremos las especificaciones técnicas de la computadora en donde se realizaron los experimentos, mencionaremos las paqueterías usadas, el preprocesamiento aplicado al corpus, y finalmente describiremos todos los modelos con sus parámetros correspondientes dividiéndolos en 3 secciones.

Nuestros experimentos se realizaron usando Python 3.9 desde un entorno de Anaconda, en una computadora con Windows 10 Pro de 64 bits, 32GBs de RAM, procesador i7-12700K, y una tarjeta NVIDIA RTX 3060.

La paqutería usada para Regresión Logística y Naive Bayes fue SKLearn, mientras que para las redes neuronales se usó TensorFlow, los modelos preentrenados se obtuvieron desde HuggingFace. Para el preprocesamiento se eliminaron signos de puntuación, todas las letras mayúsculas se convirtieron en minúsculas, y se preservaron los números. Las representaciones usadas varían dependiendo del modelo en cuestión, veamos los detalles.

Naive Bayes y Regresión. Entrenamos estos dos modelos usando las métricas de Dice, Jaccard, vectores tf y vectores tf-idf, primero analizando cada característica de manera individual y por último combinando las 4 características en un solo modelo. Para obtener los parámetros adecuados de la regresión se realizó un GridSearch sobre el conjunto de entrenamiento, no hay parámetros de NB ya que el modelo GaussianNB

de SKLearn se implementa de manera directa. En la tabla siguiente se pueden observar los parámetros usados en cada modelo de regresión logística, dependiendo de las cada métrica usada.

Redes Neuronales. Implementamos 4 redes siamesas con diferentes arquitecturas base. Una red siamesa está compuesta por dos redes idéndticas que procesan las oraciones de manera independiente, posteriormente realizan una concatenación de los valores respectivos, usan una capa densa y clasifican en función esta última capa. Las arquitecturas bases que usaremos son:

– Red 1: LSTM ,
– Red 2: biLSTM,
– Red 3: Doble LSTM,
– Red 4: Doble biLSTM.

Como parte del preprocesamiento hicimos embeddings locales de dimensión $n = 25$, las funciones de activación de cada capa oculta fueron ReLU, mientras que para el output se consideró $\tanh$. Recordando que tenemos un problema de clasificación binaria usamos como función de pérdida Binary Cross-Entropy (BCE), mientras que el optimizador fue Adam para las 4.

Modelos pre-entrenados. Usamos 3 modelos preentrenados obtenidos de HuggingFace: BERT [4], DistilBERT [7], all-Mini [17]. A partir de estos hicimos un fine-tuning usando el MRPC, al ser modelos neuronales ajustamos nuevamente algunos hiperparámetros para igualar los modelos neuronales previos, es decir la pérdida fue BCE mientras que el optimizador fue Adam.

5. Resultados

Al trabajar con problemas de clasificación binaria se tienen métricas definidas que resumen el comportamiento de un modelo dedicado a este problema. Las 4 métricas que usaremos serán Accuracy, Precision, Recall, y F1 que se calculan en función de 4 clasificaciones de las predicciones de un modelo.

Recordemos que como nuestro problema es uno de aprendizaje supervisado, dado un input sabemos el valor real de dicho input; en el contexto de pares de oraciones significa que dadas dos oraciones sabemos a priori si las oraciones son o no son paráfrasis; diremos que un par de oraciones pertenecen a la clase positiva si sí son paráfrasis, en caso contrario diremos que pertenecen a la clase negativa. En función de esto definimos 4 categorías de predicciones de un modelo.

– **Verdadero Positivo (TP)**: Una predicción verdadera de un par de oraciones pertenecientes a la clase positiva.
– **Verdadero Negativo (TN)**: Una predicción falsa de un par de oraciones pertenecientes a la clase negativa.
– **Falso Positivo (FP)**: Una predicción verdadera de un par de oraciones pertenecientes a la clase negativa.
– **Falso Positivo (FN)**: Una predicción falsa de un par de oraciones pertenecientes a la clase positiva.

Tabla 2. Regresión logística.

Métrica	Acc	Recall	Precision	F1
Dice	0.69	0.69	0.67	0.62
Jaccard	0.69	0.69	0.68	0.65
Levenshtein	0.67	0.68	0.68	0.65
tf	0.72	0.72	0.7	0.69
tf-idf	0.72	0.72	0.71	0.7
Multi	**0.73**	**0.73**	**0.72**	**0.72**

Tabla 3. Naive Bayes.

Métrica	Acc	Recall	Precision	F1
Dice	0.67	0.67	0.67	0.55
Jaccard	0.69	0.69	0.67	0.63
Levenshtein	0.67	0.67	0.68	0.65
tf	**0.72**	**0.72**	0.7	0.69
tf-idf	**0.72**	**0.72**	0.71	0.7
Multi	0.71	0.71	**0.72**	**0.72**

Tabla 4. Redes neuronales.

Modelo	Acc	Recall	Precision	F1
LSTM	0.73	**0.88**	0.71	**0.79**
biLSTM	**0.74**	0.75	0.72	0.74
2xLSTM	**0.74**	0.74	0.73	0.73
2xbiLSTM	0.73	0.71	**0.74**	0.72

Tabla 5. Preentrados.

Modelo	Acc	Recall	Precision	F1
BERT	0.86	0.85	**0.94**	**0.89**
DistilBERT	**0.87**	**0.88**	0.9	**0.89**
all-Mini	0.77	0.82	0.81	0.81

Definimos:

$$\textbf{Accuracy} = \frac{\mathrm{TP} + \mathrm{TN}}{\mathrm{TP} + \mathrm{TN} + \mathrm{FP} + \mathrm{FN}}, \tag{14}$$

$$\textbf{Precision} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FP}}, \tag{15}$$

$$\textbf{Recall} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FN}}, \tag{16}$$

$$\textbf{F1} = 2 \cdot \frac{\mathrm{precision} \cdot \mathrm{recall}}{\mathrm{precision} + \mathrm{recall}}. \tag{17}$$

Ahora presentamos los resultados completos con las métricas de evaluación previamente mencionadas. Observamos que los modelos clásicos de clasificación funcionan mejor entre más características puedan analizar, aunque modelos como $\mathrm{tf} - \mathrm{idf}$ son suficientemente buenos de manera aislada. Sin embargo estos modelos se quedan considerablemente detrás de las implementaciones neuronales, incluso la peor red funciona a la par de los modelos clásicos.

Notamos que una red LSTM sencilla supera modelos más complejos y que una arquitectura demasiado rebuscada como la doble bidireccional puede ser contraproducente para problemas sencillos. Por otra parte los modelos preentrenados superan sustancialmente al resto, BERT y DistilBERT son 2 de los 10 modelos más usados en todo HuggingFace por lo que el alto rendimiento es esperado, aunque incluso un modelo no tan notorio como es all-Mini al estar diseñado para el análisis de oraciones supera a las redes neuronales tradicionales.

6. Conclusiones

Confirmamos que los modelos preentrenados son el estado del arte para la detección de paráfrasis automatizada, si bien los modelos tradicionales presentan resultados adecuados se ven opacados por modelos preentrenados. Por otra parte este estudio presenta posibilidades de crecimiento, futuros trabajos podrían buscar la implementación de estos modelos en corpus más especializados o no tan conocidos como lo es el MRPC. A niveles técnicos las implementaciones fueron simples, para expandir estos experimentos se pueden hacer búsquedas más extensas sobre parámetros particulares y distintas arquitecturas planteadas.

Agradecimientos. Este artículo fue financiado por los proyectos Detección automática de paráfrasis mediante métodos basados en la distribución de texto, A1-S-27780, y Grafos conceptuales para la construcción de diccionarios inversos en áreas de especialidad, CF-2023-G-64, del Consejo Nacional de Ciencia y Tecnología (CONACYT), al igual que el proyecto Desarrollo del sistema de gestión de corpus GECO, IT100822 del Programa de Apoyo a Proyectos de Investigación e Innovación Tecnológica (PAPIIT).

Referencias

1. Arase, Y., Tsujii, J.: Transfer fine-tuning of BERT with phrasal paraphrases. Computer Speech and Language, vol. 66, pp. 101164 (2021) doi: 10.1016/j.csl.2020.101164
2. Conneau, A., Khandelwal, K., Goyal, N., Chaudhary, V., Wenzek, G., Guzmán, F., Grave, E., Ott, M., Zettlemoyer, L., Stoyanov, V.: Unsupervised cross-lingual representation learning at scale (2019) doi: 10.48550/arXiv.1911.02116
3. Corbeil, J. P., Abdi Ghavidel, H.: BET: A backtranslation approach for easy data augmentation in transformer-based paraphrase identification context (2020) doi: 10.48550/arXiv.2009.12452
4. Devlin, J., Chang, M. W., Lee, K., Toutanova, K.: BERT: Pre-training of deep bidirectional transformers for language understanding. In: Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Association for Computational Linguistics, vol. 1, pp. 4171–4186 (2019) doi: 10.18653/v1/N19-1423
5. Firth, J. R.: A synopsis of linguistic theory 1930-1955. Studies in Linguistic Analysis, Special Volume of the Philological Society, vol. 1952-59, pp. 1–31 (1957)
6. Francis, W. N., Kucera, H.: Brown corpus manual. Technical report, Department of Linguistics, Brown University, Providence, Rhode Island, US (1979) icame.uib.no/brown/bcm.html
7. HF Canonical Model Maintainers: Distilbert-base-uncased-finetuned-sst-2-english (2022) doi: 10.57967/HF/0181
8. Hochreiter, S., Schmidhuber, J.: Long short-term memory. Neural Computation, vol. 9, no. 8, pp. 1735–1780 (1997) doi: 10.1162/neco.1997.9.8.1735
9. Ji, Y., Eisenstein, J.: Discriminative improvements to distributional sentence similarity. pp. 891–896 (2013)
10. Jurafsky, D., Martin, J. H.: Speech and language processing: An introduction to natural language processing, computational linguistics, and speech recognition. Prentice Hall, Pearson Education International (2009)

11. Lan, Z., Chen, M., Goodman, S., Gimpel, K., Sharma, P., Soricut, R.: ALBERT: A Lite BERT for self-supervised learning of language representations (2019) doi: 10.48550/ARXIV.1909.11942
12. Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., Levy, O., Lewis, M., Zettlemoyer, L., Stoyanov, V.: Roberta: A robustly optimized bert pretraining approach (2019) doi: 10.48550/ARXIV.1907.11692
13. Meshram, S.: Review on NLP paraphrase detection approaches. International Journal of Innovative Science and Research Technology, vol. 4, no. 2, pp. 351–354 (2019)
14. Mikolov, T., Chen, K., Corrado, G., Dean, J.: Efficient estimation of word representations in vector space. International Conference on Learning Representations (2013)
15. Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., Dean, J.: Distributed representations of words and phrases and their compositionality. In: Advances in Neural Information Processing Systems, vol. 26 (2013) doi: 10.48550/ARXIV.1310.4546
16. Mohamed, I., Wael, H.: Exploring the recent trends of paraphrase detection. International Journal of Computer Applications, vol. 182, no. 46, pp. 1–5 (2019) doi: 10.5120/ijca2019918317
17. Reimers, N., Gurevych, I.: Sentence-BERT: Sentence embeddings using siamese BERT-networks (2019) doi: 10.48550/ARXIV.1908.10084
18. Schuster, M., Paliwal, K.: Bidirectional recurrent neural networks. IEEE Transactions on Signal Processing, vol. 45, no. 11, pp. 2673–2681 (1997) doi: 10.1109/78.650093
19. Sparck-Jones, K.: A statistical interpretation of term specificity and its application in retrieval. Document Retrieval Systems, vol. 28, no. 1, pp. 11–21 (1972) doi: 10.1108/eb026526
20. Sutskever, I., Vinyals, O., Le, Q. V.: Sequence to sequence learning with neural networks. In: Advances in Neural Information Processing Systems, pp. 3104–3112 (2014) doi: 10.48550/arXiv.1409.3215
21. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., Polosukhin, I.: Attention is all you need (2017) doi: 10.48550/ARXIV.1706.03762
22. Vila, M., Martí, M. A., Rodríguez, H.: Paraphrase concept and typology. A linguistically based and computationally oriented approach. Sociedad Española para el Procesamiento del Lenguaje Natural, vol. 46 (2011)
23. Vrbanec, T., Meštrović, A.: Corpus-based paraphrase detection experiments and review. Information, vol. 11, no. 5, pp. 241 (2020) doi: 10.3390/info11050241
24. Yu, S., Xu, C., Liu, H.: Zipf's law in 50 languages: Its structural pattern, linguistic interpretation, and cognitive motivation (2018) doi: 10.48550/ARXIV.1807.01855
25. Zhao, J., Wang, T., Yatskar, M., Cotterell, R., Ordonez, V., Chang, K.-W.: Gender bias in contextualized word embeddings. In: Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Association for Computational Linguistics, vol. 1, pp. 629–634 (2019) doi: 10.18653/v1/N19-1064