Matching Patient Cases to Clinical Trials

Ana Beatriz Breia, João Funenga, and Mário Miranda

NOVA School of Science and Technology - Universidade NOVA de Lisboa

Abstract. No mundo da medicina e da saúde é fundamental fazer uma boa correspondência entre os casos dos pacientes e os ensaios clínicos, porém estima-se que não se consegue encontrar um bom ensaio clínico em 80% dos pacientes, no tempo previsto. Assim, este projeto tem como principal objetivo utilizar técnidas de recuperação e análise de informação por forma a conseguir resolver problemas de correspondência e relevância entre documentos.

Keywords: Ensaios Clínicos · Recuperação de Informação· Relevância entre Dcocumentos.

1 Introdução

Por forma a encontrar ensaios clínicos onde os pacientes possam participar consideramos duas grandes fontes de dados: as *queries*, que representam os registos dos pacientes, e os ensaios clínicos, que contêm uma panóplia de informações relativas a quem pode ou não ser recrutado para esse determinado ensaio. Uma vez que estas informações são bastante extensas, apenas consideremos uma secção destes documentos, os *brief_summaries*, que constituirão o nosso *corpus.*
As principais estratégias aplicadas e analisadas neste projeto, para realizar boas correspondências entre estes os dados, são modelos como o *Vector Space Model* ou modelos de linguagem.

2 Métodos Implementados

Este capítulo é dedicado a um breve resumo teórico dos métodos implementados no projeto. Para mais detalhes veja [**slides**].

2.1 Vector Space Model e LMJM smoothing

Neste projecto foram testados dois *retrieval models*. O primeiro modelo utilizado foi um modelo de *"geometric/linear spaces"*, o *Vector Space Model*.
O *Vector Space Model* (VSM) é um modelo algébrico para representar documentos de texto. Uma forma de analisar a importância de um documento dado uma certa *query* é usar o método TF-IDF, isto é, vamos dar "pesos" diferentes aos termos desse documento. Estes pesos são determinados segundo a frequência desse termo no documento ($tf_i(d)$), e no grau de raridade desse termo em todo o *corpus* (idf_i). Temos então,

$$w_{i,j} = tf_i(d_j) \cdot idf_i$$

Também é possível comparar o quão similares são documentos diferentes, calculando o cosseno do ângulo x entre ambos (função *pairwise_distance*).

O segundo modelo utilizado foi o *"Language Model with Jelineck-Mercer smoothing"*. A ideia básica deste método é estimar um modelo de linguagem, por forma a determinar a probabilidade de determinada *query* pertencer ao nosso *corpus*.

SEGUNDA PARTE

2.2 Standardização

Para que um modelo de Machine Learning (ML) não tenha que lidar com atributos com diferentes escalas/ unidades (instabilidade numérica), deve-se reescalar, neste caso standardizar, os atributos para que fiquem todos na mesma unidade. Contudo, é importante notar que não faz sentido standardizar a variável resposta visto que esta é uma variável binária.

A standardização faz-se da seguinte maneira

$$z = \frac{x - \mu}{\sigma},$$

onde μ representa a média e σ o desvio padrão, ambos do conjunto de treino.

2.3 Regressão Logística

A Regressão Logística é um método de regressão, contudo também pode ser utilizado como classificador se considerarmos

$$P(C_0|x) = P(C_1|x),$$

que funciona como valor de corte das duas classes.

2.4 Avaliação

Para avaliar os nossos resultados vamos utilizar as seguintes métricas:
Precision@10: que determina quantos documentos no top 10 são relevantes para uma dada *query;*
nDCG@5: que mede o ganho cumulativo dos vários níveis de relevância nos documentos.
Recall@100: que dá o nº de documentos relevantes que não foram disponibilizados ao paciente;
Mean Reciprocal Rank: que calcula a fração $\frac{1}{rank}$, onde *rank* designa a posição do primeiro documento relevante;
Mean Average Precision: para avaliar a precisão média dos documentos;
Precision-recall curves: que mostra graficamente a relação entre a precisão e o recall.

TERCEIRA PARTE

2.5 Modelo Bert e Tokenizer

O Modelo Bert (*Bidirectional Encoder Representations from Transformers*) é um *language model* baseado em transformações, sendo que o primeiro passo para a sua utilização é o *Tokenizer*.

Por *Tokenizer* entende-se a fragmentação de palavras, seja na transformação da palavra nela própria (uma palavra transforma-se num *token*) ou em vários segmentos (uma palavra é dividida em diversos *tokens*).

Voltando ao nosso modelo, este tem duas capacidades principais: *language modelling* (prever *tokens* em falta a partir do contexto frásico) ou *next sentence prediction* (prever se uma frase é provável dada a frase anterior). [**bert**]

O *core* deste modelo baseia-se no método *transformer* que apresentaremos na próxima subsecção.

2.6 Word Embedding

Word embedding é uma forma de representar palavras através de um vetor real, de modo que palavras semelhantes tenham representações semelhantes. Para tal, a cada palavra vai estar associado um determinado vetor que *"codifica"* o significado dessa palavra num determinado contexto frásico.

Para determinar o *embedding* final, recorre-se ao método *transformer* que utiliza várias *layers*. Cada *layer* é responsável por dar mais peso a certos aspetos de cada palavra (através de *heads*). Assim, cada *layer*, sendo no total 12, vai ter um *input embedding* e um *output embedding*.

3 Configuração Experimental

3.1 Ler os Ensaios Clínicos e os Casos dos Pacientes

A primeira fase do nosso projeto é ler e extrair a informação pertinente dos documentos dos ensaios clínicos. Para tal, apenas analisamos os ensaios presentes no ficheiro *qrels-clinical_trials.txt*. Este ficheiro é composto por várias linhas em que cada uma contém 3 informações que nos interessam, o "Patient Id" (primeira coluna), "Clinical Trial Id" começado por NCT (terceira coluna) e o valor dado à qualidade da correspondência entre o respetivo paciente e ensaio, podendo este ser 0 (mau *match*), 1 (bom *match*), 2 (excelente *match*). Os documentos que não estão presentes neste ficheiro não foram avaliados por profissionais, ou seja, não conseguimos saber se são relevantes ou não. Assim, começamos por percorrer o ficheiro *qrels-clinical_trials.txt* e guardar todos os ID's nele presente para depois podermos filtrá-los do ficheiro ".zip", que contém todas as entradas de ensaios clínicos de dezembro de 2015. Este ficheiro de texto corresponde à nossa *ground-truth*, isto é, temos de facto a qualidade do *match* entre os casos dos pacientes e os ensaios clínicos.
Os ensaios clínicos são o *corpus* do nosso projeto. As *queries* são os casos dos pacientes.

3.2 Utilização do Vector Space Model

Vamos agora analisar a importância de um documento dada uma certa *query*, isto é, vamos utilizar o método TF-IDF (*"TfidfVectorizer"* da biblioteca sklearn), bem como a função *pairwise_distance*. De modo a testarmos várias abordagens, iremos experimentar este modelo tanto com unigramas como com bigramas, bem como com e sem "stop words", por forma a comparar a performance de todos estes cenários.

3.3 Utilização do LMJM com smoothing

Com o intuito de descobrir o melhor lambda que integra a fórmula do LMJM, fizemos a separação das *queries* num grupo de treino (80% das *queries*) e num grupo de teste. A razão pela qual são separadas as *queries* e não os ensaios é simples: do ponto de vista de um paciente é desejável saber quais os ensaios mais relevantes segundo o seu caso, por isso a análise para cada paciente (*query*) tem de ser feita com todos os ensaios disponíveis. Caso fossem os ensaios separados num grupo de treino e outro de teste, estaríamos a comparar para cada ensaio os pacientes que seriam atribuídos, o que faria sentido da perspetiva de um médico responsável por um *trial*, porém não é esse o objetivo do trabalho.

Relativamente à escolha do melhor lambda, este foi escolhido de acordo com o P@10 e, de facto, vamos poder confirmar no próximo capítulo que o lambda escolhido correponde à curva com uma área superior.

Para isto, em cada iteração sobre os dados de treino, calculamos o valor do P@10 e caso este valor seja melhor que o anterior, guardamos o lambda correspondente para usarmos posteriormente sobre o grupo de teste.
Também utilizaremos este modelo tanto com unigramas como com bigramas.

SEGUNDA PARTE

3.4 Generalizar a implementação do LMJM e do VSM

Uma vez que o nosso objetivo agora é determinar uma combinação linear de *scores*, começamos por generalizar a nossa implementação do LMJM e do VSM para todas as secções dos documentos dos ensaios clínicos, isto é, para o *brief_title*, para o *brief_summary*, para a *detailed_description* e para o *criteria*.

Para isso, cada uma das funções relativas aos modelos recebe alguns argumentos, como a secção e o número de n-gramas, e retorna uma lista de docscores. Depois de implementadas as funções, chamamo-las para todas as secções dos documentos, para unigramas.

3.5 Criar triplos e dicionário

Para treinar o nosso modelo, precisamos de criar uma lista de tuplos com 3 elementos, a partir do ficheiro *qrels-clinical_trials.txt*, sendo eles o id do paciente, o id do documento e o *match* entre eles.

De seguida, criamos um dicionário para o método LMJM, em que as chaves são um par (*query*, documento), e os valores são os 4 *docscores* do modelo. Depois, adicionamos a este dicionário os 4 *docscores* do VSM, obtendo o nosso dicionário 'final'.

Por forma a treinar corretamente o nosso modelo, fazemos a separação das queries num conjunto de treino e num conjunto de teste. O conjunto de treino é constituído por *queries* e documentos para os quais conhecemos a *groundtruth* (*qrels-clinical_trials.txt*), visto que precisamos de calcular o *score* para determinar os melhores coeficientes. Já o conjunto de teste, são as *queries* restantes, mas com todos os documentos presentes do corpus, e não só aqueles para o qual temos *groundtruth*.

Depois de standardizar os valores dos *docscores* correspondentes ao conjunto de treino, como foi referido na secção 2.3, criámos uma matriz com as *features* (*docscores*), mas com as linhas do *match* igual a 2 substituídas por um *match* igual a 1, e triplicamos estas mesmas linhas, de maneira que o número de observações com estes *matches* (1 e 2) seja mais semelhante ao número de observações com *match* 0, de modo a equilibrar a proporção de documentos relevantes *vs* não relevantes, isto porque, no conjunto de dados inicial, existem mais exemplos com *match* 0 do que os restantes.

3.6 Treinar o modelo

O passo seguinte é treinar o nosso modelo de Regressão Logística. Note-se que, apesar deste ser um problema de *ranking*, iremos utilizar este método como classificador de modo a obter os coeficientes que maximizam o número de documentos bem classificados.

Uma vez que os polinómios que vamos obter têm grau 8, é importante utilizar um paramêtro de regularização para que o modelo não sofra *overfitting*. Assim, implementamos uma função de *cross-validation*, *calc_fold_LR*, que recebe um determinado valor c (paramêtro de regularização), treina o modelo com esse mesmo valor e devolve os respetivos erro de treino e erro de validação. Posteriomente estes erros são utilizados para obter uma média de erros (de treino e validação). Fazemos este processo para cada c no intervalo [0.1,1] e no final guarda-se o melhor valor de erro de validação e o respetivo c, que será utilizado daqui em diante.

De seguida, fazemos *fit* do modelo através das features e das classes de treino, e por fim, obtemos os coeficientes (utilizando a função *.coef_*).

Uma vez que se trata de um problema de *ranking*, aplicamos estes coeficientes aos 8 atributos dos dados de teste, que resultará num *score* final por par (*query*, documento) que será organizado de forma decrescente (do mais relevante para o menos relevante)

Note-se que antes de utilizar os dados reservados para teste é necessário standardizá-los (utilizando a média e o desvio padrão do conjunto de treino). De seguida, fazemos a filtragem das nossas *queries* por sexo e intervalo de idade, segundo os critérios de cada ensaio clínico. Por fim, vamos calcular as métricas p10, recall, ap, ndcg5, mrr para os respetivos pares de *queries* e documentos filtrados.

TERCEIRA PARTE

Nesta terceira parte, o objetivo é semelhante ao anterior, isto é, calcular o *ranking* dos documentos para cada *query*, utilizando uma combinação linear cujos coeficientes são calculados através de uma regressão logística, por forma a dar pesos às diferentes *features* que maximizam a performance do modelo. Assim, nesta terceira parte, teremos 768 *features* extraídas, para cada par *query*-documento, através do modelo pré-treinado num corpus de índole médico ("Bio-ClinicalBERT") . Para a concretização deste método utilizaram-se, por razões computacionais, apenas os primeiros 200 caracteres do *detailed-description* de cada documento.

3.7 Extração de embeddings e Treino

Para a conceção deste modelo, como já foi explicado, é necessário extrair os *embeddings* das palavras. Isto é, para cada palavra vamos obter um vector que representa essa mesma palavra numa frase composta por *query*-documento. No entanto, como o objectivo para este projeto é perceber se uma frase pode vir a seguir a outra ou não, não é muito relevante saber os *embeddings* de cada *token* individualmente. Assim, como medida para percebermos a compatibilidade de uma *query* com um documento utilizar-se-á o *token* 'CLS'. O CLS é um *token* que foi treinado para conseguir dizer se as duas frases estão relacionadas ou não.

Depois de se extrair os *embeddings* de cada *token* CLS para cada par *query*-documento, procedeu-se ao treino. Nesta fase de treino, utilizaram-se as mesmas *queries* que na fase anterior do projeto e, achou-se o valor ótimo de C, através de *cross-validation*, como sendo de 0.1.

Para concluir, procedeu-se à ordenação dos documentos para cada uma das *queries* (pacientes). Para isto, e de forma a dar um peso diferente às 768 *features* que caracterizam cada *token* CLS, multiplicaram-se os coeficientes anteriormente calculados por cada uma das *features*.

4 Análise dos Resultados

4.1 Resultados com Vector Space Model

Ao aplicarmos este modelo sem "stop words" (veja-se a **figura 11b** da curva *precision-recall*), podemos observar que inicialmente a precisão é elevada, tanto para unigramas como para bigramas, e vai decrescendo à medida que vamos classificando os documentos, isto significa que quanto mais documentos classificarmos mais documentos não relevantes vamos descobrir, mas por outro lado, vamos conseguir aproximarmo-nos do número total de documentos relevantes (o recall aumenta). Assim, o melhor cenário seria a curva ser o mais próxima de uma reta na forma $y = 1$, isto significaria que o nosso modelo conseguiu classificar corretamente todos os documentos relevantes como relevantes, de todos os que existem.

Para testar o modelo com "stop words" utilizou-se:

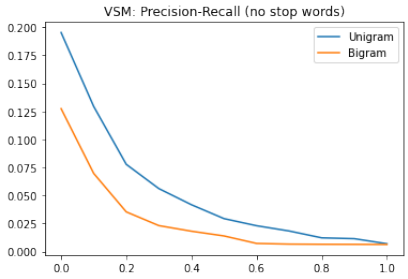
common_stop_words = "is", "the", "an", "a", "to", "and", "be", "been", "that", "this", "i", "than", "patient", "am", "health", "sick", "clinical",

tendo-se obtido o gráfico da **figura 1b**:

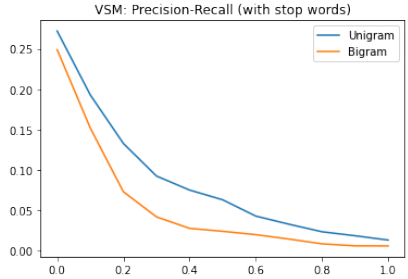
Conclui-se, pelos gráficos, que o método com "stop words" apresenta um pequeno aumento na melhoria de performance, tanto em unigramas como em bigramas. Já relacionando os n-gramas, conclui-se que os unigramas, em ambos os casos, apresentam resultados mais satisfatórios, uma vez que a área abaixo das curvas é superior à dos bigramas, traduzindo-se num valor mais alto de precisão e de recall.

4.2 Resultados com LMJM

O mesmo raciocínio utilizado na curva precision-recall no caso do *Vector Space Model*, também é válido na curva no caso do LMJM. Veja-se a **figura 2** (a precisão decresce e o recall aumenta).



(a) VSM sem Stop Words

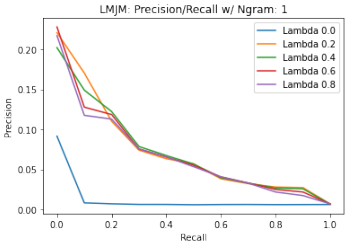


(b) VSM com Stop Words

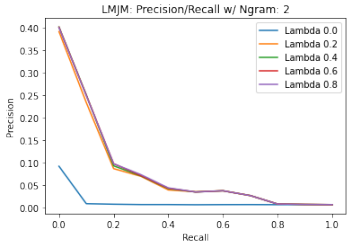
Fig. 1: Comparação da curva Precision-Recall no método VSM sem e com Stop Words

Calculando o melhor lambda para unigramas e bigramas, a partir da métrica P@10, e atendendo a figura 2, o melhor lambda para ambos os casos, como já foi referido, é 0.4, o que na teoria faz sentido, uma vez que baixos valores de lambda são adequados para *queries* longas.

Para além disto, neste caso é nítido que a utilização de bigramas aumenta bastante a perfomance do método em termos de precisão, porém em termos de recall os unigramas são superiores.



(a) Unigramas



(b) Bigramas

Fig. 2: Comparação dos diferentes valores para lambda na curva Precision-Recall

Tendo calculado o melhor lambda, analisemos agora a curva de *precision-recall* relativa ao conjunto de dados reservados para o teste do modelo.

Através do gráfico da **figura 3** percebemos que a performance neste *set* de dados desconhecido não corresponde ao esperado, o que nos leva a crer que possivelmente o modelo sofreu *overfitting*, por exemplo, e portanto não consegue generalizar corretamente as ocorrências dos dados.

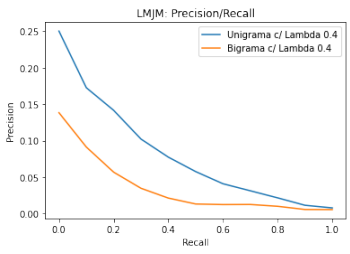
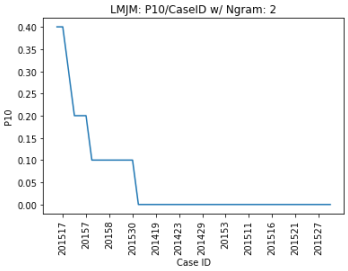
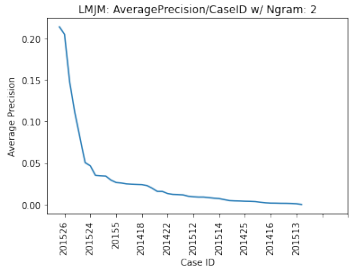


Fig. 3: Comparação da curva Precision-Recall para o modelo LMJM



(a) P@10 por paciente



(b) AP por paciente

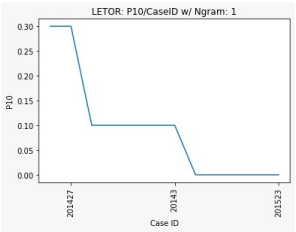
Fig. 4: Comparação de P@10 e AP por paciente.

O gráfico da **figura 4a)**, relaciona o valor do P@10 com os 60 pacientes que compõem as nossas *queries*. Fazendo a sua análise, podemos verificar que existem pacientes onde nenhum documento relevante apareceu nos 10 primeiros resultados. Caso não nos queiramos limitar aos 10 primeiros documentos, podemos fazer a mesma análise no gráfico com a precisão média (AP) pelos pacientes (gráfico da **figura 4b**).

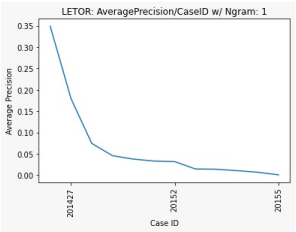
4.3 Resultados com LETOR Model

Analisando os resultados do LETOR Model (figura 5), o *P@10* obtido não é tão satisfatório comparativamente aos métodos anteriores, o que significa que este modelo tem mais dificuldade em encontrar um documento relevante, dada uma determinada *query*, nos 10 primeiros documentos apresentados. Já a *average precision*, apresenta uma melhoria. Este aumento pode ser explicado porque utilizando o LETOR Model, tiramos partido do melhor dos dois modelos, dando um "peso" específico a cada um deles através dos coeficientes calculados. Outro aspecto que contribui para este aumento, está relacionado com o facto de restringirmos o número de documentos a comparar com o paciente, filtrando estes consoante o seu género e idade, isto é, excluímos *Clinical Trials* em que a idade e o género do paciente não são compatíveis com a *query*.

Comparando a curva *Precision-Recall* do LETOR model com as anteriores, conseguimos perceber que esta não é tão satisfatória, uma vez que a área abaixo da curva é ligeiramente inferior às curvas dos outros modelos.



(a) P@10 por paciente



(b) AP por paciente

Fig. 5: Comparação de P@10 e AP por paciente.

4.4 Formatação de Input e Tokenizationl

Como explicado na secção anterior a utilização de fragmentos de palavras permite ao BERT identificar facilmente palavras relacionadas. Tomemos como exemplo a palavra *'stiffness'*, esta é constituída pela

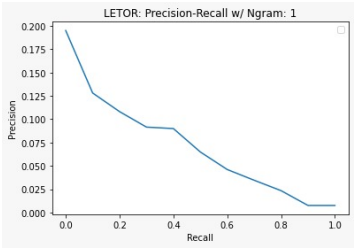


Fig. 6: Precision-Recall para o LETOR model

palavra raíz *'stiff'* e pelo sufixo *'##ness'*. Como o método consegue fazer esta separação, é possível comparar a palavra *stiff* nas várias formas, por exemplo no singular e no plural. Outro exemplo seria a palavra *Hypertension*, que é separada em *hyper##* e *##tension*, assim se houvesse uma palavra do tipo *Hypotension* o modelo relacionaria-as através do *token '##tension'*.

4.5 Entendimento e Visualização de atenção-Bert Model

Antes de procedermos à avaliação dos resultados obtidos com este modelo parece-nos interessante analisar a forma como este modelo atua e o porquê de ser tão conhecido e utilizado.

A partir do que foi dito sobre o *transformer*, é interessante perceber como o *embedding* de cada palavra se vai alterando ao longo das diferentes *layers*. Para podermos observar estas alterações através de um gráfico, foi necessário extrair os *embeddings* das palavras em diferentes *layers*. Neste projeto, apenas extraímos os *embeddings* da primeira *layer* e da última, sendo que a última tem em conta todas as anteriores. Como cada palavra é representada por um vetor com 768 *features*, foi necessário reduzir este número de forma a podermos representá-las num gráfico. Para isto, utilizou-se a função TSNE da biblioteca Sklearn, reduzindo o número de *features* para duas.

No gráfico abaixo, podemos observar a forma como cada palavra é caracterizada em cada *layer* e como diferentes palavras se vão afastando ou aproximando mediante aquilo que se esteja a treinar.

É fácil de entender que, para diferentes *layers* a mesma palavra tem coordenadas diferentes no gráfico, isto porque o vetor que a caracteriza (o seu *embedding*) é diferente. Também podemos constatar que, por exemplo, as palavras *'half'* e *'by'* na primeira *layer* se encontram próximas, no entanto, na última *layer* estas afastam-se. Isto demonstra, que na primeira *layer* estas duas palavras são semelhantes, e que à medida que o contexto frásico é analisado elas se vão afastando.

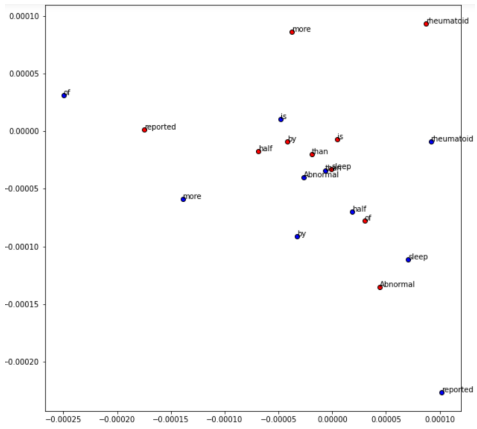
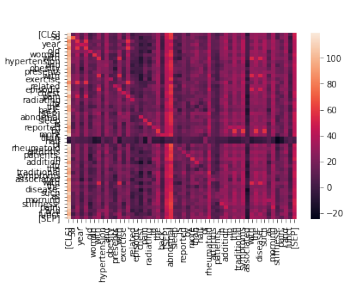
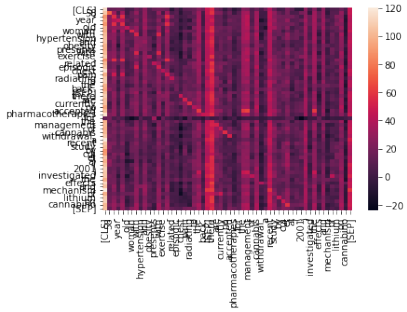


Fig. 7: Visual Embeddings da 1ª e última layer

Agora apresentaremos dois *HeatMaps* de *self-attention*. Estes gráficos representam a relação entre o *embedding* de entrada e de saída de uma *layer*. As palavras para as quais vamos calcular a *self-attention* são *Query*+Documento Relevante/*Query*+Documento Não Relevante. Por este ser um processo computacionamente exigente apenas o fizemos para a primeira *layer*.



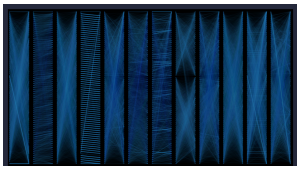
(a) Query+Documento Relevante



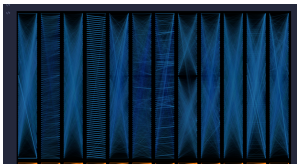
(b) Query+Documento Não Relevant

Fig. 8: HeatMap de Self Attention para a 1ª layer

Pelos gráficos, não se consegue perceber grandes diferenças entre os diferentes pares Query-Documento. Apenas se pode verificar que a relação entre os tokens CLS são maiores no gráfico Query+Documento Relevante.



(a) Query+Documento Não Relevante



(b) Query+Documento Relevante

Fig. 9: Self Attention

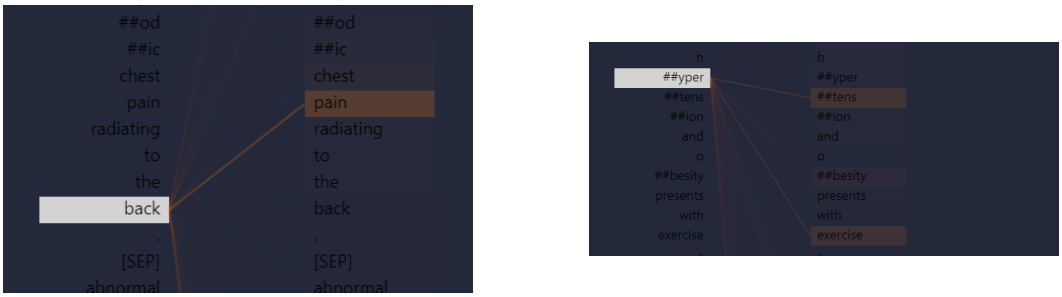


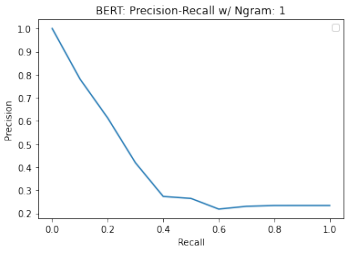
Fig. 10: Relação entre tokens

Os gráficos acima apresentados, apesar de também corresponderem ao *self-attention*, permitem ver a diferença de relações entre *tokens* nas várias *heads* da mesma *layer*. Assim como nas outras análises, analisou-se a mesma *query* para um documento relevante e outro não relevante. Esta análise é feita na primeira *layer*, entre a primeira e a última *head*.

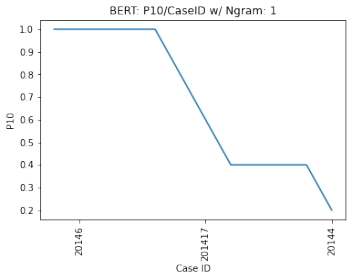
Nos documentos relevantes, para a primeira *head*, em geral todas as palavras se relacionavam com todas mas com baixa correlação. Quando analisamos a última *head*, as relações entre os *tokens* já são mais específicas (ligações entre prefixos e radicais). Por exemplo, o *token* 'hyper' relaciona-se fortemente com 'tension', e o 'pain' com 'back' (**Figura 10**).

Nos documentos não relevantes as relações entre *tokens* sao bastante semelhantes ao primeiro por estar a comparar *token* a *token*. No entanto, o CLS não se relaciona com tantos *tokens* do documento.

4.6 Resultados com Bert Model



(a) Curva Precision-Recall



(b) @P10 por paciente

Fig. 11: Análise de Resultados

Visualizando tanto o gráfico que compara a *Precision-Recall* com aquele que compara o P@10 por paciente, percebemos que neste modelo os resultados parecem ser bastante satisfatórios. Com este modelo atingiu-se algo que até agora nunca foi possível atingir. Agora, é possivel encontrar pacientes em que os primeiros 10 documentos recuperados são de facto relevantes para o paciente, e que apenas para um número reduzido de pacientes não é possível apresentar o P@10 acima de 40%. Estes aumentos apesar de consideráveis, para certos conjuntos de Query-Documento, poderiam ser melhorados visto que só estamos a considerar os primeiros 200 caracteres do *detailed-Description*.

	VSM (w/o Stop Words)					VSM (w/ Stop Words)				
	P@10	*Recall@100*	*AP*	*NDCG@5*	*MRR*	*P@10*	*Recall@100*	*AP*	*NDCG@5*	*MRR*
Unigramas	0.095	0.2580	0.0718	0.0968	0.0051	0.0967	0.2630	0.0715	0.0940	0.0051
Bigramas	0.0583	0.1200	0.0377	0.0675	0.0051	0.0566	0.1289	0.0408	0.0723	0.0051
	LMJM									
	P@10	*Recall@100*	*AP*	*NDCG@5*	*MRR*					
Unigramas	0.0958	0.2272	0.0658	0.0880	0.0050					
Bigramas	0.0999	0.2310	0.0662	0.1049	0.0058					
	LETOR									
	P@10	*Recall@100*	*AP*	*NDCG@5*	*MRR*					
Unigramas	0.0917	0.8324	0.0665	0.0624	0.0070					
	Bert									
	P@10	*Recall@100*	*AP*	*NDCG@5*	*MRR*					
Unigramas	0.6833	0.6836	0.8426	0.6467	0.3602					

4.7 Resultados com Métricas de Avaliação

Analisemos primeiro o método VSM. Conseguimos verificar que, na generalidade, os valores das métricas entre o método com ou sem "stop words" são muito semelhantes, porém ligeiramente mais elevados em algumas métricas com a presença destas. Isto porque, quando usadas no método, estas palavras são ignoradas, não enviesando a probabilidade do *match*. Em relação ao uso de unigramas e bigramas, podemos verificar que com unigramas a perfomance é significativamente melhor, tanto no *P@10* como no *recall@100*.

Note-se que neste domínio o recall, comparativamente à precisão, é uma métrica mais importante, visto ter em conta falsos negativos (documentos considerados não relevantes, mas relevantes), enquanto que na precisão, os falsos negativos não entram no cálculo desta. Não ter em conta os falsos negativos resulta, neste caso, em ignorarmos ensaios clínicos (analisando-os como um mau *match*) e estes serem possívelmente positivos (relevantes). Visto ser do interesse do paciente saber todos os ensaios em que este pode participar, faz sentido analisá-los a todos.

Em jeito de conclusão da análise do VSM, se tivéssemos de utilizar este método , aplicá-lo-íamos com "stop words" e recorrendo a unigramas.

Relativamente ao LMJM acontece o contrário, sendo a performance com bigramas melhor que com unigramas. Por estarmos a lidar com termos médicos que por vezes são constituídos por mais que uma palavra, faz sentido que o uso de bigramas seja mais relevante, embora não tenha sido o concluído com o VSM. Isto pode ser devido ao modelo VSM ser mais antigo e não se comportar tão bem para o tipo de dados que estamos a analisar.

Relativamente ao MRR, este não é muito relevante no nosso caso de estudo uma vez que apenas tem em conta a posição do primeiro documento relevante. Ora, um paciente não tem apenas em conta o primeiro ensaio relevante que lhe é apresentado, este pode, e deve, analisar outros ensaios em posições "maiores" das ordenadas.

Posto isto, entre ambos os modelos, o que seria mais indicado escolhermos seria o método LMJM com bigramas, visto que apresenta os melhores resultados.

Analisando os valores das métricas para o LETOR Model, podemos observar que alguns são inferiores e outros superiores, comparativamente aos modelos estudados anteriormente.

Os valores que se apresentam agora inferiores podem-se dever ao facto de utilizarmos as *queries* e todos os documentos do corpus no conjunto de teste, que pode ser constituído por um elevado número de documentos não relevantes, o que influencia algumas das métricas calculadas. Caso tivéssemos

utilizado apenas as *queries* para as quais conhecemos *groundtruth*, os resultados seriam possivelmente mais satisfatórios.

Apesar dos outros resultados o *Recall@100* aumentou, isto significa que nos 100 primeiros documentos obtiveram-se mais documentos relevantes face aos outros dois modelos.

Para o último modelo aplicado podemos ver que os resultados foram, sem dúvida, os mais satisfatórios. A razão é simples. Enquanto os modelos anteriores devolvem o mesmo vetor para uma palavra, independentemente da forma como é utilizada, o modelo Bert devolve vetores diferentes para a mesma palavra, dependendo das palavras à sua volta. Apesar dos resultados em termos globais terem aumentado, o Recall não se comportou da mesma forma. Ora esta métrica, no contexto em que se insere o projeto, é de facto importante uma vez que podemos estar a privar pacientes de correspondências (ensaios) relevantes.

4.8 Conclusão

Face ao exposto anteriormente, é-nos possível concluir que em termos de resultados os modelos VSM e LMJM apresentam valores semelhantes, mesmo sabendo que o VSM é um modelo mais antigo e sendo o LMJM, segundo a literatura, mais eficaz para *queries* longas. Isto acontece porque o modelo VSM utiliza a classe TF-IDF, que dá "pesos" às palavras consoante a sua raridade. Com o nosso dataset de índole médica, por ter um vocabulário especifico, esta vantagem do VSM face ao modelo LMJM é evidente. (Esta "vantagem" poderia ser reduzida e talvez eliminada com a utilização de N-gramas mais altos no modelo LMJM).

Comparando estes dois modelos face ao Modelo LETOR, é fácil de concluir que este último obteve alguns resultados melhores e outros piores, pelas razões mencionadas acima. Apesar disso, o esperado seria uma melhoria nos gráficos/resultados em geral, uma vez que o LETOR Model, como já foi referido anteriormente, funciona como uma evolução resultante da junção dos métodos anteriores.

Assim, perante estes resultados, é necessário definir bem o nosso objetivo aquando a procura de *Clinical Trials* relevantes para uma determinada *query*. Se o objetivo for encontrar o máximo de documentos relevantes entre os primeiros analisados, então o LETOR Model é a escolha mais apropriada.

Como conclusão para todo o projeto, ao longo dos modelos apresentados, podemos constatar um aumento progressivo de desempenho. Ficou claro que quanto mais informação e detalhe tivermos em conta, melhor é o desempenho. Comparando os três primeiros modelos com o Bert Model, este apresenta maior detalhe. Como já falado, este caracteriza as palavras consoante o contexto em que se insere, permitindo-lhe olhar para o significado efetivo das palavras, em cada par *query*-Documento, sendo por isto, bastante melhor que os restantes.

References

1. Chaudhary, A. (2020, 8 de outubro). *Evaluation Metrics For Information Retrieval.* Acedido a 02/11/2021, em `https://amitness.com/2020/08/information-retrieval-evaluation/`
2. BERT (language model). Wikipedia, the free encyclopedia. Acedido a 09/01/2022, em `https://en.wikipedia.org/wiki/BERT$\_$(language$\_$model)`
3. Horev, R. (2018, 10 de novembro). *BERT Explained: State of the art language model for NLP* Acedido a 09/01/2022, em `https://towardsdatascience.com/bert-explained-state-of-the-art-language-model-for-nlp-f8b21a9b6270`
4. Yeung, A. (2020, 19 de junho). *BERT - Tokenization and Encoding* Acedido a 09/01/2022, em `https://albertauyeung.github.io/2020/06/19/bert-tokenization.html/`