UNIVERSIDADE FEDERAL DE GOIÁS
INSTITUTO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

DIOGO DE FREITAS RIBEIRO

Estudo comparativo de comitês de sub-redes neurais para o problema de aprender a ranquear

Goiânia
2023

UNIVERSIDADE FEDERAL DE GOIÁS
INSTITUTO DE INFORMÁTICA

TERMO DE CIÊNCIA E DE AUTORIZAÇÃO (TECA) PARA DISPONIBILIZAR VERSÕES ELETRÔNICAS DE TESES E DISSERTAÇÕES NA BIBLIOTECA DIGITAL DA UFG

Na qualidade de titular dos direitos de autor, autorizo a Universidade Federal de Goiás (UFG) a disponibilizar, gratuitamente, por meio da Biblioteca Digital de Teses e Dissertações (BDTD/UFG), regulamentada pela Resolução CEPEC nº 832/2007, sem ressarcimento dos direitos autorais, de acordo com a Lei 9.610/98, o documento conforme permissões assinaladas abaixo, para fins de leitura, impressão e/ou download, a título de divulgação da produção científica brasileira, a partir desta data.

O conteúdo das Teses e Dissertações disponibilizado na BDTD/UFG é de responsabilidade exclusiva do autor. Ao encaminhar o produto final, o autor(a) e o(a) orientador(a) firmam o compromisso de que o trabalho não contém nenhuma violação de quaisquer direitos autorais ou outro direito de terceiros.

1. Identificação do material bibliográfico

[x] Dissertação [ ] Tese [ ] Outro*:_____

*No caso de mestrado/doutorado profissional, indique o formato do Trabalho de Conclusão de Curso, permitido no documento de área, correspondente ao programa de pós-graduação, orientado pela legislação vigente da CAPES.

Exemplos: Estudo de caso ou Revisão sistemática ou outros formatos.

2. Nome completo do autor

Diogo de Freitas Ribeiro

3. Título do trabalho

Estudo comparativo de comitês de sub-redes neurais para o problema de aprender a ranquear

4. Informações de acesso ao documento (este campo deve ser preenchido pelo orientador)

Concorda com a liberação total do documento [X] SIM [ ] NÃO[1]

[1] Neste caso o documento será embargado por até um ano a partir da data de defesa. Após esse período, a possível disponibilização ocorrerá apenas mediante:
a) consulta ao(à) autor(a) e ao(à) orientador(a);
b) novo Termo de Ciência e de Autorização (TECA) assinado e inserido no arquivo da tese ou dissertação.
O documento não será disponibilizado durante o período de embargo.
Casos de embargo:
- Solicitação de registro de patente;
- Submissão de artigo em revista científica;
- Publicação como capítulo de livro;
- Publicação da dissertação/tese em livro.

Obs. Este termo deverá ser assinado no SEI pelo orientador e pelo autor.

Documento assinado eletronicamente por **Thierson Couto Rosa**, **Professor do Magistério Superior**, em 21/09/2023, às 13:14, conforme horário oficial de Brasília, com fundamento no § 3º do art. 4º do Decreto nº 10.543, de 13 de novembro de 2020.

Documento assinado eletronicamente por **Diogo De Freitas Ribeiro**, **Discente**, em 21/09/2023, às 16:33, conforme horário oficial de Brasília, com fundamento no § 3º do art. 4º do Decreto nº 10.543, de 13 de novembro de 2020.

A autenticidade deste documento pode ser conferida no site https://sei.ufg.br/sei/controlador_externo.php?acao=documento_conferir&id_orgao_acesso_externo=0, informando o código verificador **4061843** e o código CRC **ADB6DD0F**.

DIOGO DE FREITAS RIBEIRO

Estudo comparativo de comitês de sub-redes neurais para o problema de aprender a ranquear

Dissertação apresentada ao Programa de Pós–Graduação em Ciência da Computaçãodo Instituto de Informática da Universidade Federal de Goiás, como requisito para obtenção do título de Mestre em Ciência da Computação.

Área de concentração: Ciência da Computação.

Linha de pesquisa: Sistemas Inteligentes e Aplicações.

Orientador: Prof. Thierson Couto Rosa

Co-Orientador: Prof. Daniel Xavier de Sousa

Goiânia
2023

Ficha de identificação da obra elaborada pelo autor, através do
Programa de Geração Automática do Sistema de Bibliotecas da UFG.

Ribeiro, Diogo de Freitas
Estudo comparativo de comitês de sub-redes neurais para o problema de aprender a ranquear [manuscrito] / Diogo de Freitas Ribeiro. - 2023.
LXXX, 80 f.

Orientador: Prof. Dr. Thierson Couto Rosa ; co-orientador Dr. Daniel Xavier de Sousa .
Dissertação (Mestrado) - Universidade Federal de Goiás, Instituto de Informática (INF), Programa de Pós-Graduação em Ciência da Computação, Goiânia, 2023.
Bibliografia.
Inclui algoritmos, lista de figuras, lista de tabelas.

1. Recuperação de Informação. 2. Aprender a Ranquear. 3. Aprendizado por comitê. 4. Comitê de sub-redes neurais. I. , Thierson Couto Rosa, orient. II. Título.

CDU 004

UNIVERSIDADE FEDERAL DE GOIÁS
INSTITUTO DE INFORMÁTICA

ATA DE DEFESA DE DISSERTAÇÃO

Ata nº **11/2023** da sessão de Defesa de Dissertação de **Diogo de Freitas Ribeiro**, que confere o título de Mestre em Ciência da Computação, na área de concentração em Ciência da Computação.

Ao primeiro dia do mês de setembro de dois mil e vinte e três, a partir das catorze horas, na sala 151 do INF, realizou-se a sessão pública de Defesa de Dissertação intitulada **"Estudo comparativo de comitês de sub-redes neurais para o problema de aprender a ranquear"**. Os trabalhos foram instalados pelo Orientador, Professor Doutor Thierson Couto Rosa (INF/UFG) com a participação dos demais membros da Banca Examinadora: Professor Daniel Xavier de Sousa (IFG), coorientador; Professor Doutor Sérgio Daniel Carvalho Canuto (IFG), membro titular externo; Professor Doutor Wellington Santos Martins (INF/UFG), membro titular interno. Durante a arguição os membros da banca não fizeram sugestão de alteração do título do trabalho. A Banca Examinadora reuniu-se em sessão secreta a fim de concluir o julgamento da Dissertação, tendo sido o candidato **aprovado** pelos seus membros. Proclamados os resultados pelo Professor Doutor Thierson Couto Rosa, Presidente da Banca Examinadora, foram encerrados os trabalhos e, para constar, lavrou-se a presente ata que é assinada pelos Membros da Banca Examinadora, ao primeiro dia do mês de setembro de dois mil e vinte e três.
TÍTULO SUGERIDO PELA BANCA

Documento assinado eletronicamente por **Daniel Xavier de Sousa**, **Usuário Externo**, em 19/09/2023, às 15:59, conforme horário oficial de Brasília, com fundamento no § 3º do art. 4º do Decreto nº 10.543, de 13 de novembro de 2020.

Documento assinado eletronicamente por **Diogo De Freitas Ribeiro**, **Discente**, em 19/09/2023, às 16:00, conforme horário oficial de Brasília, com fundamento no § 3º do art. 4º do Decreto nº 10.543, de 13 de novembro de 2020.

Documento assinado eletronicamente por **Thierson Couto Rosa**, **Professor do Magistério Superior**, em 19/09/2023, às 18:59, conforme horário oficial de Brasília, com fundamento no § 3º do art. 4º do Decreto nº 10.543, de 13 de novembro de 2020.

Documento assinado eletronicamente por **Sergio Daniel Carvalho Canuto**, **Usuário Externo**, em 20/09/2023, às 16:23, conforme horário oficial de Brasília, com fundamento no § 3º do art. 4º do Decreto nº 10.543, de 13 de novembro de 2020.

Documento assinado eletronicamente por **Wellington Santos Martins**, **Professor do Magistério Superior**, em 21/09/2023, às 11:53, conforme horário oficial de Brasília, com fundamento no § 3º do art. 4º do Decreto nº 10.543, de 13 de novembro de 2020.

A autenticidade deste documento pode ser conferida no site https://sei.ufg.br/sei/controlador_externo.php?acao=documento_conferir&id_orgao_acesso_externo=0, informando o código verificador **4054442** e o código CRC **75A7FFA7**.

Referência: Processo nº 23070.037140/2023-18 SEI nº 4054442

Agradecimentos

Agradeço primeiramente a Deus, fonte de toda sabedoria e conhecimento, que me abençoou em cada etapa, proporcionando-me força e discernimento para superar os desafios inerentes à realização desta pesquisa.

Agradeço imensamente ao meu orientador, Prof. Dr. Thierson Couto Rosa, pela paciência, dedicação e sabedoria. Suas orientações foram essenciais para o desenvolvimento deste trabalho e marcaram profundamente minha trajetória acadêmica. Igualmente, sou grato ao meu co-orientador, Prof. Dr. Daniel Xavier de Sousa, cujo conhecimento e suporte foram fundamentais para o aprimoramento da dissertação.

Meu sincero agradecimento à Universidade Federal de Goiás (UFG) e aos recursos computacionais do LaMCAD/UFG que utilizei para realização dos experimentos.

Finalizo reiterando minha profunda gratidão a todos que contribuíram direta ou indiretamente para a realização desta pesquisa. Cada palavra de incentivo, cada momento de compartilhamento de conhecimentos e experiências, foram fundamentais para a concretização desta dissertação acadêmica. É com humildade e reconhecimento que celebro a conclusão deste trabalho e agradeço por todas as bênçãos recebidas ao longo desta jornada.

Sob as circunstâncias certas, grupos são notavelmente inteligentes e muitas vezes são mais inteligentes do que o indivíduo mais inteligente dentro deles.

James Surowiecki,
A Sabedoria das Multidões.

Resumo

Ribeiro, Diogo. **Estudo comparativo de comitês de sub-redes neurais para o problema de aprender a ranquear**. Goiânia, 2023. 80p. Dissertação de Mestrado. Programa de Pós-Graduação em Ciência da Computação, Instituto de Informática, Universidade Federal de Goiás.

Aprender a Ranquear (AR) é uma sub-área em Recuperação de Informação que tem como objetivo usar aprendizado automático para otimizar o posicionamento dos documentos mais relevantes no ranque de resposta a uma consulta específica. Até recentemente, o método *LambdaMART*, que corresponde a um comitê de árvores de regressão, era considerado o estado-da-arte em AR. Contudo, a introdução do *AllRank*, um método de aprendizado profundo que incorpora mecanismos de *self-attention*, superou o *LambdaMART* como uma abordagem mais efetiva para tarefas de AR. Este estudo, em questão, explorou a efetividade e a eficiência dos comitês de sub-redes neurais como um método complementar a uma ideia que já era de excelência, que é o *self-attention* utilizado no AllRank, estabelecendo assim um novo patamar de inovação e efetividade no domínio do ranqueamento. Diferentes métodos de formação de comitês de sub-redes, como Multi-Sample Dropout, Multi-Sample Dropout (Treino e teste), BatchEnsemble e Masksembles, foram implementados e testados em duas coleções de dados padrão: MSLR-WEB10K e YAHOO!. Os resultados dos experimentos indicaram que algumas dessas abordagens de comitês, especificamente Masksembles e BatchEnsemble, superaram o AllRank original em métricas como NDCG@1, NDCG@5 e NDCG@10, embora fossem mais custosas em termos de tempo de treinamento e teste.

Em conclusão, a pesquisa revela que a aplicação de comitês de sub-redes neurais em modelos de AR é uma estratégia promissora, especialmente em cenários onde o tempo de latência não é crítico. Desse modo, esse trabalho não apenas avança o estado da arte em AR, mas também abre novas possibilidades para melhorias de efetividade e eficiência, inspirando pesquisas futuras no uso de comitês de sub-redes neurais em AR.

Palavras–chave

Recuperação de Informação. Aprender a Ranquear. Aprendizado por comitê. Comitê de sub-redes neurais

Abstract

Ribeiro, Diogo. **A Comparative study of ensembles of neural sub-networks for the learning to ranking problem**. Goiânia, 2023. 80p. MSc. Dissertation. Programa de Pós-Graduação em Ciência da Computação, Instituto de Informática, Universidade Federal de Goiás.

Learning to Rank (L2R) is a sub-area of Information Retrieval that aims to use machine learning to optimize the positioning of the most relevant documents in the answer ranking to a specific query. Until recently, the LambdaMART method, which corresponds to an ensemble of regression trees, was considered state-of-the-art in L2R. However, the introduction of AllRank, a deep learning method that incorporates self-attention mechanisms, has overtaken LambdaMART as the most effective approach for L2R tasks. This study, at issued, explored the effectiveness and efficiency of sub-networks ensemble as a complementary method to an already excellent idea, which is the self-attention used in AllRank, thus establishing a new level of innovation and effectiveness in the field of ranking. Different methods for forming sub-networks ensemble, such as Multi-Sample Dropout, Multi-Sample Dropout (Training and Testing), BatchEnsemble and Masksembles, were implemented and tested on two standard data collections: MSLR-WEB10K and YAHOO!. The results of the experiments indicated that some of these ensemble approaches, specifically Masksembles and BatchEnsemble, outperformed the original AllRank in metrics such as NDCG@1, NDCG@5 and NDCG@10, although they were more costly in terms of training and testing time. In conclusion, the research reveals that the application of sub-networks ensemble in L2R models is a promising strategy, especially in scenarios where latency time is not critical. Thus, this work not only advances the state of the art in L2R, but also opens up new possibilities for improvements in effectiveness and efficiency, inspiring future research into the use of sub-networks ensemble in L2R.

Keywords

Information Retrieval, Learning to Rank, Ensemble Learning, Sub-networks ensemble

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Algoritmos

CAPÍTULO 1

Introdução

Em sistemas de Recuperação de Informação (*Information Retrieval*), o processo de busca de documentos é normalmente dividido em duas etapas: primeiro, os documentos potencialmente relevantes para uma determinada consulta de um usuário são identificados e, em seguida, os documentos encontrados são ordenados por uma função de ranqueamento que visa colocar os documentos mais relevantes para a consulta no topo da lista [83].

A função de ranqueamento é gerada fazendo uso de um método supervisionado. Este método utiliza uma coleção de dados de treinamento para gerar a função de ranqueamento. Assim, este aprendizado supervisionado é denominado *Aprender a Ranquear* (AR) (*Learning to Rank*). Deste modo, a função gerada utiliza os atributos do par consulta-documento para posicionar o documento em ordem de relevância em relação aos demais documentos relevantes a uma consulta [55]. Na verdade, espera-se de uma boa função de ranqueamento que ela seja capaz de posicionar os documentos mais relevantes mais próximo do topo do ranqueamento gerado.

Até recentemente, o algoritmo estado da arte em AR era o LambdaMART [15]. O LambdaMART corresponde a um comitê (*ensemble*) de árvores de decisão geradas pela técnica de *boosting*. A abordagem de comitê busca combinar múltiplos algoritmos de aprendizagem (que no caso do LambdaMART correspondem às múltiplas árvores de decisão) que aprendem aspectos distintos durante o treinamento. O resultado do comitê é obtido através de alguma função que combina os resultados dos seus componentes. Por exemplo, essa combinação pode ser o resultado da média dos valores de relevância atribuídos por cada árvore de decisão a um documento.

Por outro lado, avanços recentes na área de Aprendizado Profundo (*Deep Learning*) têm sido usados para melhorar as técnicas de AR. Pobrotyn *et al.* [68] propuseram um modelo chamado *AllRank*, que pode ser considerado atualmente o estado da arte para tarefas de AR. O AllRank é capaz de extrair informações das interações entre os documentos de uma determinada consulta utilizando o mecanismo de *self-attention* [86].

Considerando que comitês e aprendizado profundo são duas abordagens que têm sido exitosas isoladamente em AR, uma estratégia que pode ser promissora é a

utilização em conjunto das duas abordagens. Essa combinação já vem sendo utilizada para o problema de classificação [37, 60, 52], onde um pequeno número de redes neurais (entre cinco e dez redes) são treinadas individualmente e suas predições são combinadas para gerar uma predição final. Esta abordagem demonstrou ser bem-sucedida para melhorar a acurácia dos resultados [52]. As redes componentes do comitê possuem a mesma arquitetura, e os mesmos valores de hiper-parâmetros (número de épocas, tamanho do *batch*, etc), mas cada uma tem os seus pesos inicializados de forma distinta [4].

No entanto, o treinamento do comitê de redes neurais é custoso, tanto em termos de recursos computacionais quanto em tempo de execução, tornando esses comitês inviáveis em várias aplicações [90, 24, 93, 45]. No problema de ranqueamento, mesmo considerando o comitê já treinado, para cada consulta teríamos k ranques gerados por k redes neurais distintas. Para que cada rede neural possa ser eficaz, ela teria que ter vários parâmetros e várias camadas. Estes ranques gerados ainda teriam que ser combinados em um único ranque final a ser apresentado ao usuário. Mas o tempo para executar todo esse processamento faz com que o tempo de resposta a uma consulta em uma máquina de busca de produção seja proibitivo.

Visando tornar viável a combinação de comitê com rede neural, mesmo em aplicações em que o tempo de resposta é menos crítico, trabalhos recentes têm aplicado a abordagem de comitês a uma única rede neural [24, 93, 90, 45]. Esta abordagem consiste em obter sub-redes de uma única rede neural e combinar as saídas dessas sub-redes neurais. As duas principais técnicas utilizadas para a geração de sub-redes neurais são:

a) por *dropout* - nesse caso, uma sub-rede é obtida a partir da rede neural-base pelo desligamento de um conjunto de neurônios. Pode-se obter sub-redes distintas pelo desligamento de conjuntos distintos de neurônios da rede original;
b) por perturbações distintas dos pesos da rede neural - nesse caso é mantida uma matriz distinta para cada sub-rede a ser criada. Durante o treinamento da rede principal, redes "virtuais" são criadas pela multiplicação (perturbação) do pesos da rede-base pelas matrizes que são mantidas para cada sub-rede.

Ao contrário do que ocorre com os comitês de redes neurais comentados acima, em ambas as abordagens de comitês de sub-redes neurais as sub-redes não são completamente independentes entre si. Sempre há algum compartilhamento de informações entre as sub-redes. Ainda assim, comitês de sub-redes neurais têm apresentado resultados de efetividade superiores ao da rede neural única da qual esses comitês são derivados [24, 93, 90, 45].

Os comitês de sub-redes neurais foram todos experimentados apenas no problema de classificação de imagens. Talvez, por ainda ser recente, essa abordagem ainda não foi investigada em outras aplicações. Especialmente, sua aplicação no problema de

AR ainda não foi pesquisada. Este trabalho visou a preencher essa lacuna. Para tanto, adaptou-se abordagens de comitês de sub-redes neurais existentes na literatura para suas aplicações em AR. Especificamente, as seguintes abordagens de comitês de sub-redes neurais foram utilizadas: Multi-Sample Dropout [45], BatchEnsemble [90], Masksembles [24] e uma simples variante do Multi-Sample Dropout denominada Multi-Sample Dropout Treino e Teste, proposta nesta pesquisa. Estes métodos foram utilizados tomando-se como rede neural base para derivar as sub-redes, a rede neural do estado da arte em AR, o AllRank [68]. Os métodos foram comparados entre si e com a proposta original do AllRank sobre os seguintes aspectos: a) a efetividade do ranque gerado, b) O tempo de treino e teste. Especificamente, o trabalho visou responder às seguintes questões de pesquisa (QP):

QP1 Quais das técnicas de comitês de sub-redes neurais aplicadas ao AllRank conseguem superá-lo em efetividade?

QP2 Existe alguma técnica de comitê de sub-redes neurais superior em efetividade a outras?

QP3 Qual o custo em termos de tempo de treino e, principalmente, tempo de teste (tempo para gerar o ranque) dos comitês de sub-redes neurais em relação ao AllRank original (sem comitês)?

Os experimentos relatados nesse trabalho mostram que dois métodos de comitês de sub-redes neurais, o Masksembles e e BathcEnsemble foram superiores em efetividade ao AllRank nas duas coleções avaliadas (Web10K e YAHOO!). Os ganhos sobre o AllRank foram de até 5,5% de NDCG@1, 5,0% de NDCG@5 e 4,0% de NDCG@10.

Em relação ao tempo de execução, os métodos de comitês são mais custosos do que o AllRank, tanto no tempo de treinamento do modelo quanto no tempo de teste do modelo. Dos dois métodos que se mostraram superiores ao AllRank em efetividade, o Masksembles é o que apresentou menor aumento de tempo tanto no treino quanto no teste, nas duas coleções, sendo que o aumento máximo no teste foi de 51% em relação ao AllRank.

O trabalho mostra que algumas abordagens de comitês de sub-redes neurais são promissoras em AR. Especialmente, essas abordagens podem ser promissoras em tarefas de ranqueamento em que o tempo de latência não é crítico. Exemplos de tais aplicações são os sistemas de recomendação *offline* e mesmo ranqueamento online em que o número de documentos associados à consulta não é muito grande.

Em resumo, as contribuições deste trabalho foram as seguintes:

- a apresentação de uma nova aplicação promissora para os métodos de comitês de sub-redes neurais, no caso, AR;

- a descrição de um estudo comparativo de quatro métodos de comitês aplicados a um modelo estado da arte em AR, analisando três medidas de efetividade (NDCG@1, NDCG@5 e NDCG@10) e tempo de treinamento e teste;
- a constatação de que alguns métodos de comitês conseguem superar o estado da arte em AR, em termos de efetividade. Esta constatação é relevante não apenas pela superação em si, mas porque o método superado, o AllRank, introduziu novidade importante que é o mecanismo de *self-attention* aplicado ao ranqueamento. Este trabalho mostra que alguns métodos de comitês de sub-redes não interferem na melhoria introduzida pelo mecanismo de *self-attention*, pelo contrário, trazem benefício, em termos de efetividade. Espera-se que essa observação inspire novos trabalhos que visem tornar mais efetivos e mais eficientes os comitês de sub-redes aplicados ao problema de AR.

Este trabalho está organizado da seguinte forma: o capítulo 2 apresenta a fundamentação teórica. O capítulo 3 trata dos trabalhos relacionados. No capítulo 4 é detalhada a metodologia utilizada nos experimentos. No capítulo 5 são apresentados e discutidos os resultados dos experimentos, em termos de efetividade e tempo de execução dos métodos. Por último, o capítulo 6 apresenta as conclusões finais do trabalho e possíveis trabalhos futuros.

CAPÍTULO 2

Fundamentação Teórica

A aplicação dos métodos de comitê de sub-redes neurais em Aprender a ranquear envolve vários conceitos que até então foram inexplorados para a tarefa de ranqueamento. Portanto, antes de procedermos à exposição da metodologia proposta, é pertinente revisitarmos alguns conceitos fundamentais para garantir uma melhor compreensão da explicação subsequente. Primeiramente, mostramos uma visão geral sobre o problema do ranqueamento e Aprender a ranquear (AR), a classificação das abordagens de AR, o uso de AR em máquinas de busca e a métrica de avaliação do modelo adotado neste estudo. Logo em seguida, explicamos sobre as técnicas de Aprendizado de Máquina para AR, tanto as tradicionais, seção 2.4, quanto a mais moderna, que é em aprendizado profundo, seção 2.5, além das suas formas utilizando a estratégia de aprendizado por comitê que são explicadas nas seções 2.4.1 e 2.5.1, respectivamente. Por fim, detalhamos algumas métricas utilizadas, como o viés e variância na seção 2.6 e a sua adaptação para AR 2.6.1.

2.1 O Problema do Ranqueamento

A Recuperação de Informação (RI)[1] é uma área abrangente da Ciência da Computação em que o foco principal é oferecer aos usuários o acesso fácil às informações do seu interesse [5]. Existem diversas tarefas em RI, mas a mais popular delas é a recuperação de documentos que são relevantes a uma consulta textual fornecida pelo usuário em máquinas de busca (Google, Bing, Yahoo, etc). Entretanto, há várias outras tarefas em RI, tais como recomendação de conteúdo, atribuir respostas a perguntas, sumarizar textos e associar propagandas a conteúdos de páginas na Web [55].

Em todas as referidas tarefas, as soluções computacionais precisam decidir quais documentos são relevantes e mostrar ao usuário apenas eles. É uma estratégia necessária, mas não suficiente na maioria das aplicações práticas em que envolvem coleções gigantescas de dados como páginas coletadas da Web. Nesses casos, o número

[1] *Information Retrieval* - em inglês.

de documentos relevantes ainda pode ser gigantesco, logo apresentar todos eles ao usuário não o ajudaria em nada.

Portanto, é necessário que as soluções de RI não apenas identifiquem se um documento é relevante, mas que também sejam capazes de atribuir um grau de relevância a esse documento. Se isso for feito, então, a solução pode apresentar ao usuário um **ranque** de documentos considerados relevantes para ele. O ranque consiste em uma lista de documentos ordenados em ordem decrescente por grau de relevância. Assim, o usuário tem maior chance de satisfazer sua necessidade de informação consultando apenas os documentos mais próximos ao topo do ranque.

Por questão didática, desse ponto em diante no texto, será considerada apenas a tarefa de recuperação de documentos em resposta a consultas feitas pelos usuários para descrever o processo de ranqueamento. O problema principal no processamento de consultas é o de se encontrar uma forma eficaz de atribuir valores de relevância a documentos em relação a uma consulta. É um problema difícil de resolver automaticamente, pois a definição do que é relevante não é explicitada pelo usuário, além disso, o grau de relevância de um documento para uma mesma consulta pode variar entre usuários. Logo, não é possível desenvolver um algoritmo que dê sempre uma resposta exata sobre o valor de um documento, pois não existe um consenso do que é resposta exata nesse problema de ranqueamento. Na verdade, o que as máquinas de busca fazem é tentar inferir de modo automatizado se um documento é relevante a uma dada consulta.

Com o objetivo de automatizar a valoração de documentos relativa a consultas, os trabalhos de pesquisa em RI propuseram utilizar funções matemáticas para essa tarefa. Ou seja, procura-se encontrar uma função $f(q, d)$ capaz de associar um valor no conjunto dos números reais ($\mathbb{R}$) ao par (q, d), onde q representa a consulta e d representa um documento.

Neste trabalho, denominaremos qualquer alternativa para a função f por *função de ranqueamento* ou ainda, *modelo de ranqueamento*, mas para gerar modelos de ranqueamento, os pesquisadores procuraram modelar matematicamente consultas e documentos. Desse modo, vários modelos surgiram na literatura, tais como o Modelo Lógico (*Boolean Model*), o Modelo do Espaço Vetorial (*Vector Space Model*) e o Modelo Probabilístico (*Probabilistic Model*) [5, 56]. Em todos esses modelos, os atributos utilizados para representar a consulta e os documentos são apenas os termos que os compõem e em alguns deles, as quantidades de termos que formam a consulta e os documentos.

No modelo vetorial, cada termo possível em um documento ou consulta é tratado como uma dimensão distinta no espaço vetorial de termos. Assim, a consulta e cada documento são representados como vetores (ou pontos) nesse espaço. O modelo vetorial atribui valores de relevância a documentos com base na proximidade deles em relação à consulta. A medida de proximidade mais utilizada é o cálculo do cosseno do ângulo

formado entre o vetor-consulta e o vetor-documento. O valor do cosseno é associado ao documento. Logo, um valor igual a um significa máximo grau de relacionamento (mesma direção) entre a consulta e o documento. Se o cosseno é zero os dois vetores são ortogonais entre si (não há termos em comum entre a consulta e o documento).

Os modelos probabilísticos consideram que a relevância r_d de um documento d em relação a uma consulta q pode existir (i.e., $r_d = 1$) ou não existir ($r_d = 0$). Na verdade, o que se deseja encontrar é um modelo de ranqueamento que estime a probabilidade de um documento d ser relevante ($r_d = 1$) para a consulta q, ou seja, estimar $P(r_d = 1|q, d)$. Diversos modelos de ranqueamento foram propostos seguindo a modelagem probabilística, mas mais utilizada na literatura e em máquinas de busca é a função Okapi BM25 [77].

Com o surgimento do hipertexto e principalmente da Web, outros atributos, isto é, não apenas os termos, surgiram e podem auxiliar na identificação de documentos relevantes. Por exemplo, o local onde o termo ocorre (título ou url), se o termo aparece em negrito ou em uma *tag* em HTML, indicando uma cor de texto diferente. Há ainda valores de importância do documento medido pelo número de *links* que o documento possui ou pelo número de *clicks* que o documento recebeu no passado em consultas que continham termos em comum com a consulta a ser processada.

Para melhorar a qualidade do ranque apresentado ao usuário, é necessário combinar o resultado de um modelo tradicional de ranqueamento (BM25 ou cosseno) com todos esses novos atributos de valoração da relevância comentados no parágrafo anterior. Entretanto, esses atributos são numerosos, podendo chegar a centenas. Portanto, encontrar uma fórmula matemática capaz de combinar todos eles e gerar um ranque ótimo é uma tarefa complicada e suscetível a erros, mesmo para uma pessoa bem treinada em RI. Além disso, em um ambiente de dados dinâmico como a Web, a importância de cada um deles na composição da função final muda frequentemente [56].

As soluções que têm sido utilizadas para combinar os diversos atributos empregam técnicas de aprendizado de máquina que visam aprender a construir uma função ou modelo de ranqueamento, usando um conjunto de treinamento. A aplicação de técnicas de aprendizado de máquina para gerar modelos de ranqueamento levou ao surgimento de uma subárea de pesquisa em RI denominada *Aprender a Ranquear* (AR) - do inglês, *Learning to Rank*. A próxima seção discute sobre essa subárea.

2.2 Aprender a Ranquear

A figura 2.1 mostra um cenário típico em AR. Logo, para aprender a ranquear, é necessário um conjunto de treino $\mathcal{T} = \{(q_i, \{p_{i,1}, p_{i,2} \dots p_{i,n_i}\})\}_{i=1}^{m}$, que é formado por m pares, em que cada par contém uma consulta q_i, $1 \leq i \leq m$, e um conjunto de com

n_i pares $p_{i,j}$, $1 \leq j \leq n_i$. Assim cada consulta q_i tem o seu próprio número n_i de pares. Além disso, cada par $p_{i,j} = (x_{i,j}, r_{i,j})$ de uma consulta q_i é composto pelo vetor de atributos $x_{i,j}$ correspondentes a um documento j que é relevante a q_i e pelo **valor verdadeiro**[2] de relevância $r_{i,j}$ do documento j em relação a q_i. Já o vetor de atributos corresponde ao conjunto de atributos de valoração comentados no penúltimo parágrafo da seção anterior. Exemplos desses atributos são: o valor do BM25 entre a consulta q_i e o documento j, a soma das frequências com que os termos da consulta q_i ocorrem no documento j, dentre vários outros. O valor de relevância $r_{i,j}$ é geralmente um número positivo (i.e., $r_{i,j} \in \mathbb{R}^+$).

Um algoritmo de aprendizado de máquina é utilizado para gerar ("aprender") automaticamente um modelo de ranqueamento f, que é capaz de combinar apropriadamente os atributos do vetor de atributos $x_{i,j}$ tal que o valor de saída de f possa predizer o valor de relevância $r_{i,j}$ de um documento de treino. Ou seja, dado uma consulta q_i e vetor de atributos $x_{i,j}$, $f(x_{i,j}) = y_{i,j}$ tal que $y_{i,j}$ seja o mais próximo possível do valor verdadeiro de relevância de $x_{i,j}$, isto é $r_{i,j}$. O algoritmo é denominado algoritmo de aprendizado porque o conjunto de treino é utilizado para "aprender" (gerar) o modelo f usando os vários exemplos do conjunto de treino. Deste modo, uma vez gerado o modelo f, ele deve ser aplicado apenas aos vetores de atributos de cada documento associado a uma consulta de teste. Por isso, na Figura 2.1, um exemplo de teste é formado pelo par $(q, \{x_1, x_2, \cdots, x_n\})$, pois o modelo f não pode considerar os rótulos dos documentos de teste para avaliá-los (ou seja, não pode considerar os pares $p_i = (x_i, r_i)$ das consultas de teste, apenas os vetores x_i de atributos que representam um documento d_i).

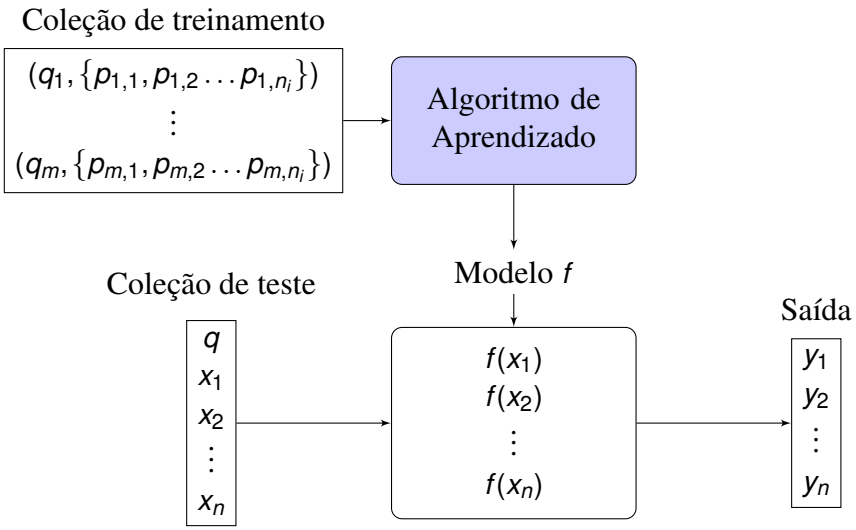


Figura 2.1: Aprendizado da função de ranqueamento [55]

[2]Denomina-se valor verdadeiro de relevância o valor de relevância atribuído por um ser humano especialista. O termo correspondente em inglês é *ground truth*.

Um grande conjunto de algoritmos de aprendizado de máquina utiliza os erros da função f nos exemplos do conjunto de treino para aprimorar a função f durante o processo de treinamento. Esse processo é denominado é *otimização do modelo* f. Mas para que a otimização possa ocorrer é necessária uma função auxiliar l denominada *função de perda* (do inglês, *loss function*). Portanto, a função de perda mede o quanto a predição $y_{i,j}$ gerada pela função f para o par $(q_i, x_{i,j})$ está próximo do valor verdadeiro de relevância r_i. Exemplos de funções de perda largamente utilizadas em AR incluem a função exponencial, a função dobradiça (do inglês, *hinge loss* e a função logística (do inglês, *logistic loss*). A função de perda exerce um papel central no processo de treinamento de um aprendizado e máquina, pois ela é utilizada não apenas para medir o erro, mas também para atualizar o modo como o modelo faz as combinações dos atributos para gerar a predição. Ou seja, é a função de perda que permite o modelo ir aprendendo a predizer os exemplos da coleção de treino.

Uma vez que um modelo de ranqueamento é obtido ao final do processo de treinamento, ele pode ser avaliado utilizando-se uma coleção de teste e medidas de avaliação de ranqueamento. Uma coleção de teste é semelhante a uma coleção de treino, isto é, os vetores de atributos que representam os documentos devem estar rotulados com o valor verdadeiro de relevância a uma dada consulta. O que difere a coleção de treino da coleção de teste é que eles devem ser conjuntos disjuntos entre si, para que a avaliação seja isenta. A avaliação da efetividade do modelo gerado quando aplicado à coleção de teste é aferida por alguma medida de efetividade. As medidas de efetividade de um modelo utilizado neste trabalho são apresentadas na seção 2.3. Entretanto, uma vez gerado o modelo de ranqueamento, ele pode ser aplicado em uma máquina de busca, como um módulo do sistema de ranqueamento.

2.2.1 Classificação das Abordagens de AR

Para simplificar a explicação sobre como um modelo é treinado (ou "aprendido"), explanamos sobre a abordagem mais comum e simples em AR na secção anterior. Entretanto, na literatura, existem distintas perspectivas propostas para as técnicas de aprender a ranquear. Elas podem ser agrupadas em três categorias, conforme a entrada, a saída do modelo e o funcionamento da função de perda durante o processo de treinamento. As categorias de abordagem são as seguintes: a) *point-wise*, b) *pair-wise* e c) *list-wise*. Tie-Yan Liu [55] destaca as seguintes abordagens:

Abordagem *point-wise*

Nessa abordagem, também conhecida como abordagem pontual, a entrada do modelo a ser treinado corresponde ao vetor de atributos e a saída corresponde a um valor

em $\mathbb{R}$ a ser predito pelo modelo. Ou seja, o modelo $f(x_{i,j}) = y_j$, onde $x_{i,j}$ é o vetor de atributos do documento j associado a uma consulta q_i e $y_{i,j}$ é uma estimativa do valor verdadeiro de $r_{i,j}$. Esse tipo de modelo é denominado *função de pontuação* (do inglês, *score function*) porque atribui uma pontuação $y_{i,j}$ ao vetor de atributos $x_{i,j}$.

Dependendo como os valores verdadeiros r dos vetores documentos são considerados, o modelo f obtido pode ser apontado como um modelo de regressão ou como um modelo de classificação. Se os valores verdadeiros são encarados como números reais, o modelo é de regressão, porém, se os valores verdadeiros são categóricos (por exemplo, não relevante, pouco relevante, relevante, muito relevante), o modelo é um classificador. No primeiro caso, algoritmos de regressão podem ser utilizados para gerar o modelo, já no segundo caso, algoritmos de classificação são utilizados. Nessas abordagens, a função de perda mede o quanto $y_{i,j}$ se difere de $r_{i,j}$ e o valor dessa função é utilizado para corrigir o modelo durante o treinamento.

Abordagem *pair-wise*

Nesta abordagem, também conhecida como abordagem em pares, o objetivo é aprender uma função de ranqueamento que minimize o número de pares de documentos que são ranqueados de maneira incorreta. Ou seja, se um documento A é considerado mais relevante do que outro B em um par (A, B), o algoritmo de aprendizado deve ranquear A mais alto do que B.

A ideia é que, se o algoritmo puder ranquear corretamente a maioria dos pares de documentos, ele será capaz de produzir uma boa lista classificada de documentos.

Deste modo, a abordagem em pares começa criando pares de documentos a partir do conjunto de treinamento. Cada par consiste em um documento relevante para a consulta do usuário e outro que é menos relevante que o primeiro. Em seguida, um algoritmo de aprendizado é usado para aprender uma função de ranqueamento que tenta ranquear um documento mais relevante acima do menos relevante.

Considere que uma consulta q é composta pelos vetores de características x_a e x_b correspondentes aos documentos A e B, respectivamente. O algoritmo de aprendizagem tenta obter uma função bivariada f que recebe um par de documentos A e B como entrada $f(x_a, x_b)$ que possa prever um valor da ordem relativa dos dois itens. Em geral, é um valor em $\{+1, -1\}$ que indica se um documento é mais relevante que o outro:

$$f(x_a, x_b) = \begin{cases} +1, \text{se } x_a \text{ é mais relevante que } x_b \\ -1, \text{se for o contrário} \end{cases} \tag{2-1}$$

A função de perda compara a ordem do par de documentos calculados pela função $f(x_a, x_b)$ em relação à ordem correta, indicada pelos valores de relevância real

dos elementos que formam o par.

Um exemplo de um método que utiliza a abordagem em pares é o *RankBoost*. Este algoritmo baseado em uma técnica de comitês denominada *boosting* (ver seção 2.4.1) aprende a ranquear através de várias iterações, tentando minimizar os erros de ranqueamento de pares em cada iteração.

A principal vantagem da abordagem em pares, conforme apresentado por Tie-Yan Liu [55], é que ela foca na tarefa de ranqueamento, em vez de se concentrar em prever uma pontuação precisa para cada documento. Isso pode torná-la mais robusta a ruídos nos dados de treinamento.

No entanto, a abordagem em pares tem a desvantagem de que ela não leva em consideração a estrutura de todo o ranqueamento. Isso significa que ela pode ignorar as interações entre os documentos que não estão no mesmo par, o que pode levar a classificações sub-ótimas.

Abordagem *list-wise*

A abordagem Listwise, também conhecida como abordagem em listas, é uma categoria de métodos de aprendizado de máquina em que o foco está em aprender a ranquear um conjunto completo de itens. Isso difere das abordagens pontuais e em pares que consideram cada documento individualmente ou pares de documentos, respectivamente.

Na abordagem em listas, a permutação de documentos é o foco principal. Ou seja, a abordagem busca aprender a permutação ideal que ordena melhor os documentos. A principal motivação para essa abordagem é que ela se alinha melhor às métricas de avaliação de Recuperação de Informação, como *Normalized Discounted Cumulative Gain* (NDCG) e *Precision at K*, que são baseadas em listas completas.

A abordagem em listas envolve um processo de aprendizado que considera toda a lista de documentos. O objetivo é minimizar uma perda baseada em uma permutação dos documentos.

Por exemplo, um método que utiliza a abordagem em listas proposto por Tie-Yan Liu [55] é o *ListNet*. O *ListNet* usa uma função de perda baseada na probabilidade de permutações. Ele modela a probabilidade de uma permutação de documentos e busca minimizar a divergência entre a distribuição de probabilidade prevista e a verdadeira distribuição de probabilidade das permutações.

Segundo Tie-Yan Liu [55], a principal vantagem da abordagem em listas é que ela é diretamente otimizada para métricas de avaliação que consideram listas completas. Assim, a abordagem em listas pode produzir ranqueamentos mais precisos em termos dessas métricas do que as abordagens pontuais e em pares.

2.2.2 O uso de AR em Máquinas de Busca

Na figura 2.2 é descrita uma máquina de busca que possui duas fases consecutivas no processo de ranqueamento. Para a primeira fase, considere a coleção $D = \{d_1, d_2, \cdots, d_n\}$ como sendo uma base de dados com um grande volume de documentos. A partir de uma consulta qualquer q de um usuário, deve ser obtido um sub-conjunto D' de D de documentos potencialmente relevantes à consulta q. O sub-conjunto D' é geralmente obtido aplicando-se um modelo de recuperação de informação tradicional como, por exemplo, o algoritmo BM25. Esse modelo gera um ranque dos documentos de D em relação à consulta q com base apenas nas palavras em q. Uma seleção dos $|D'|$ documentos mais próximos do topo do ranque é feita para formar o sub-conjunto D'. Ao final da primeira fase, os atributos dos documentos são computados, gerando para cada documento o vetor de atributos que é utilizado na segunda fase do ranqueamento.

A segunda fase do ranqueamento consiste em aplicar em D' um modelo supervisionado $f(q, D^{'})$ previamente treinado por algum método de AR. O ranque gerado por esse modelo é então apresentado ao usuário [55].

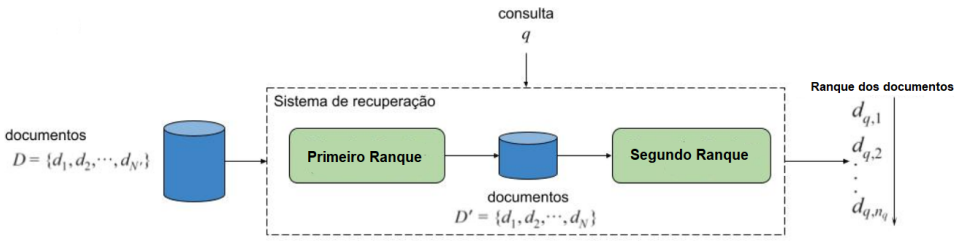


Figura 2.2: Processo de uma máquina de busca [55]

2.3 Medida de avaliação do modelo

Em problemas tradicionais de aprendizado de máquina como, por exemplo, classificação de imagens, a avaliação normalmente é realizada medindo a probabilidade da predição em relação ao valor de relevância real.

No entanto, quando se trata de problemas de Aprender a Ranquear, a tarefa é classificar uma lista de itens com base em suas relevâncias para uma consulta específica de usuário. Nesses casos, a avaliação é mais complexa, pois é importante levar em consideração a ordem dos itens na lista, não apenas suas relevâncias individuais [55]. Isto é, uma função de ranqueamento utiliza os atributos do par consulta-documento para computar um valor de relevância do documento em relação à consulta. Após calcular todas as pontuações do conjunto de documentos, é gerada uma lista L de pontuações ordenadas em ordem decrescente.

Existem várias medidas disponíveis para avaliar a qualidade desta lista L gerada por um modelo no contexto de Aprender a Ranquear. Algumas dessas medidas incluem a Média da Precisão Média (MAP - Mean Average Precision), a Precição em k (P@k - Precision at k), o Ranqueamento recíproco médio (MRR - Mean Reciprocal Rank) e o Ganho Cumulativo Descontado Normalizado (NDCG - Normalized Discounted Cumulative Gain). Cada uma dessas medidas aborda aspectos diferentes do problema de ranqueamento e ajuda a avaliar a efetividade do modelo em ordenar os itens relevantes no topo da lista [55]. A seguir, focaremos na descrição da medida NDCG que é a mais utilizada na literatura e que também foi, pois foi utilizada nos experimentos apresentados neste trabalho.

Antes de entender o NDCG precisamos compreender o conceito de Discounted Cumulative Gain (DCG). O DCG é calculado com base na relevância dos itens recuperados, levando em conta a posição do item na lista. Em geral, os itens mais relevantes aparecem no topo da lista e são considerados os melhores resultados.

Se considerarmos q_i como uma consulta e D_i como o conjunto de documentos correspondentes a essa consulta, vamos definir y_i como os valores de relevância verdadeiros e π_i como uma ordem de ranqueamento sobre D_i. O Ganho Cumulativo Descontado (DCG) para uma posição k nessa ordem π_i pode ser calculado por meio de uma função específica, em que j representa o índice de um documento dentro do conjunto D_i, conforme descrito na equação 2-2:

$$DCG(k) = \sum_{j:\pi_i(j)\leq k}^{j} \frac{2^{y_{i,j}} - 1}{log_2(1 + \pi_i(j))} \tag{2-2}$$

O NDCG é uma versão normalizada do DCG. Desse modo, como o DCG depende do tamanho da lista, diferentes listas podem ter valores de DCG que não são comparáveis. Para tornar os valores comparáveis entre as listas de classificação, o DCG é normalizado pelo DCG Ideal (IDCG), que é o DCG quando todos os itens estão ranqueados perfeitamente conforme a sua relevância.

Portanto, o NDCG é calculado como:

$$NDCG(k) = \frac{DCG_k}{IDCG_k} \tag{2-3}$$

O valor de NDCG pode variar de 0 a 1, onde 1 indica que os itens foram classificados perfeitamente conforme a sua relevância. Portanto, quanto maior o valor do NDCG, melhor é a qualidade da classificação.

2.4 Técnicas Tradicionais de Aprendizado de Máquina para AR

Com o surgimento das técnicas denominadas *Aprendizado Profundo* (em inglês *Deep Learning*), as técnicas de aprendizagem de máquina têm sido classificadas em dois tipos: a) *técnicas tradicionais de aprendizado por máquina* e b) *técnicas de aprendizado profundo* [32].

As técnicas "tradicionais" de aprendizado de máquina geralmente requerem que os dados sejam preparados e transformados por um processo conhecido como "engenharia de recursos". Além disso, as técnicas tradicionais se concentram na aprendizagem de uma função não composta aplicada aos dados de entrada.

Em contraste, o aprendizado profundo é caracterizado por aprender hierarquias de características por meio de funções compostas geradas durante o processo de treinamento. De modo geral, a aprendizagem pode ser realizada a partir de dados brutos ou não estruturados sem a necessidade de engenharia de recursos extensivos. Por exemplo, em uma tarefa de classificação de imagens, um modelo de aprendizado profundo pode aprender a identificar características relevantes, tais como: bordas, texturas e formas, diretamente dos dados brutos da imagem, enquanto um algoritmo de aprendizado de máquina tradicional normalmente exigiria que essas características fossem identificadas e extraídas manualmente antes do treinamento.

De modo simplificado, a aprendizagem de máquina (em inglês *Machine Learning*) supervisionada utilizada para resolver um determinado problema, consiste em "aprender" (ou gerar automaticamente) uma função $f(x)$ em que a entrada corresponde aos valores dos atributos de x, sendo $x \in \mathbb{R}$, e cuja saída desta função é a solução do problema para aquela instância.

Na aprendizagem de máquina, a solução $f(x)$ é gerada automaticamente (por isso o termo aprendizado de máquina), utilizando-se para esta aprendizagem um conjunto de amostras para as quais se conhece a solução problema. Esse conjunto é denominado *coleção de treinamento*.

Inicialmente, técnicas como as árvores de decisão e as máquinas de vetores de suporte (SVM) foram bastante usadas em AR. Porém, essas técnicas utilizam de abordagens pontuais (Pointwise) no processo de aprendizado, tratando cada instância de maneira independente, sem considerar sua relação com outras instâncias [55].

Contudo, visando aprimorar as técnicas de ranqueamento, foram propostos métodos como, por exemplo, Processos Gaussianos (GP - Gaussian Precesses) [20] em que é utilizada a regressão ordinal com o intuito de usar a abordagem em listas (listwise) no processo de aprendizado em Aprender a ranquear. Em [21], é proposta uma adaptação do SVM para gerar funções de ranqueamento com abordagem em listas. Em [31] a Regres-

são Logística é usada para inferir relevância de documentos na tarefa de ranqueamento. Já em [50], uma árvore de decisão é utilizada para predizer classes ordinais, o que é um problema semelhante ao de Aprender a ranquear.

Entretanto, devido à complexidade do problema de aprender funções de ranqueamento percebeu-se que uma única técnica tradicional não era suficiente para treinar boas funções de ranqueamento. Assim, uma abordagem para melhorar a qualidade da função de ranque foi utilizar um comitê de funções geradas por uma técnica clássica de aprendizado. Esses comitês são denominados *Ensembles* na literatura.

O *Random Forests*, um dos primeiros algoritmos de comitê aplicados a AR, combina várias árvores de decisão treinadas em subconjuntos aleatórios dos dados. Esta abordagem melhorou o desempenho em relação aos métodos pontuais, oferecendo robustez e reduzindo o sobreajuste [12].

Posteriormente, os métodos de boosting, que combinam vários modelos de aprendizado fraco para formar um modelo forte, se tornaram populares em AR. Como exemplo, temos o RankBoost [27] que otimiza o ranqueamento em pares de documentos. Logo depois, surgiu o LambdaMART [16], que é um algoritmo de AR que combina MART (*Multiple Additive Regression Trees*) [28] e LambdaRank [16]. O LambdaMART é uma extensão do algoritmo MART, ambos pertencentes à família de *Gradient Boosting Decision Trees* (GBDT). Porém, o LambdaMART implementa uma função de custo derivada do LambdaRank que permite ajustar os pesos das árvores de decisão durante o treinamento para melhorar a precisão do ranqueamento.

Visando melhorar a eficiência e a eficácia dos métodos de boosting, foram desenvolvidas implementações otimizadas, como XGBoost [19] e LightGBM [47]. O XGBoost usa a função de perda logarítmica de classificação para treinar seus modelos e além de ser reconhecido por sua velocidade e desempenho. O LightGBM, por sua vez, utiliza um método de crescimento de árvores de decisão baseado em folhas que resulta em uma maior eficiência e precisão.

Uma tendência mais recente de pesquisa em AR corresponde ao uso de técnicas de aprendizado profundo para superar em efetividade as técnicas clássicas, em vez de agrupá-las em comitês.

2.4.1 Aprendizado por Comitê com Técnicas Tradicionais

As estratégias de comitê, que envolvem a combinação de várias hipóteses de aprendizado para formar uma hipótese agregada, têm se mostrado eficazes para melhorar a efetividade dos modelos de aprendizado de máquina. A razão subjacente a essa melhoria pode ser atribuída à diversidade entre os modelos constituintes do comitê. Quando cada modelo individual é treinado, ele captura diferentes características do espaço de entrada,

levando a uma especialização em diferentes partes dos dados. Com a combinação desses modelos diversos, o comitê pode proporcionar uma cobertura mais abrangente dos dados, resultando em previsões mais precisas [25].

Além disso, a combinação de múltiplos modelos em um comitê tem o potencial de reduzir o sobreajuste (*overfitting*), que é uma das principais fontes de incerteza na previsão. Logo, cada modelo individual no comitê pode ter uma tendência para ajustar-se demais em diferentes partes dos dados. No entanto, ao combinar esses modelos, a tendência de sobreajuste de um único modelo pode ser mitigada, levando a redução do viés e da variância do modelo. Portanto, esta redução visa diminuir o erro total, melhorando a qualidade do modelo em fases de treino e teste. [55, 39].

Medir a incerteza em um modelo de aprendizado de máquina é outra importante contribuição das estratégias de comitê. A incerteza é medida avaliando a variabilidade nas previsões provenientes dos vários modelos que formam o comitê. Então, um modelo com baixa incerteza possui uma maior confiabilidade de suas previsões. Em setores críticos, como diagnóstico médico, finanças e sistemas autônomos de condução, é importante entender o grau de certeza associado a uma previsão para tomar decisões informadas e minimizar riscos. Portanto, ao combinar as previsões de múltiplos modelos, a incerteza pode ser significativamente reduzida, já que o comitê tira proveito da sabedoria coletiva de vários modelos em vez de depender de um único modelo. Esta redução na incerteza pode levar a previsões mais confiáveis e melhorar a capacidade do modelo de generalizar para dados não vistos [30].

A ideia central no método comitê é treinar vários modelos para resolver o mesmo problema. Ao contrário de tentar criar somente um modelo para gerar as predições, os métodos de comitê são a combinação de diversos algoritmos de aprendizado de máquina chamados de modelos fracos para criação de um modelo mais forte. Cada modelo fraco produz um resultado preditivo fraco a partir de uma determinada projeção a partir das instâncias e atributos extraídos do conjunto de dados, aumentando a diversidade dos resultados gerados [23].

São descritos quatro aspectos principais que podem ser utilizados individualmente ou em conjunto para caracterizar a diversidade de um *comitê* [95]:

- Diversidade de Dados: A diversidade também pode ser introduzida por meio dos dados usados para treinar cada modelo individual. Logo, técnicas como *bagging* e *boosting* geram conjuntos de dados de treinamento ligeiramente diferentes para cada modelo, o que pode levar a modelos que extraiam contextos distintos para as diferentes partes do espaço de entrada.
- Diversidade de Algoritmos: Utilizar diferentes algoritmos de aprendizado de máquina para construir cada modelo individual pode aumentar a diversidade do comitê. Neste sentido, cada algoritmo tem suas próprias características e pode se destacar

em diferentes aspectos do problema, permitindo que o comitê, na totalidade, seja mais robusto.

- Diversidade de Parâmetros: Os modelos individuais em um comitê também podem ser diversificados ajustando os parâmetros de cada modelo de maneira diferente. Por exemplo, considere o algoritmo de aprendizagem de árvores de decisão, cada árvore pode ser treinada com profundidades diferentes, critérios de divisão diferentes, etc.
- Diversidade de Atributos: Outra maneira de introduzir a diversidade é treinar cada modelo em um subconjunto diferente dos atributos. Isso pode ser particularmente útil ao quando os elementos de entrada possuem vários atributos e cada modelo individual só podendo lidar efetivamente com um subconjunto deles.

Existem três estratégias tradicionalmente utilizadas em comitê que são: *bagging* [11], *stacking* [91] e *boosting* [26].

Bagging

Bagging é uma técnica para minimizar o erro de generalização combinando vários modelos fracos em que cada modelo fraco é treinado com uma coleção de treinamento diferente. Assim, cada coleção de treinamento é gerada aleatoriamente com reposição a partir do conjunto de treinamento original. Em outras palavras, cada conjunto de treinamento tem o mesmo tamanho que o conjunto de treinamento original, mas algumas amostras podem ser repetidas enquanto outras podem ser omitidas devido à seleção aleatória. Dessa forma, um modelo de aprendizado de máquina é treinado independentemente em cada um desses conjuntos de dados. Portanto, o resultado do *bagging* é obtido agregando as previsões dos modelos individuais, enquanto a diferença entre as amostras da coleção de treinamento resulta em diferenças nos modelos treinados, aumentando a diversidade das predições. Já as amostras que não foram selecionadas na coleção de dados de treinamento são denominadas de *out-of-bagging* e são geralmente utilizadas na etapa de validação do modelo [11].

As predições dos modelos fracos são combinados para gerar a predição final. Neste sentido, a combinação pode ser feita por um esquema de votação ou pela média das predições. A figura 2.3 ilustra a estrutura de uma estratégia *ensemble bagging*.

Stacking

O Comitê por Empilhamento (*Stacking*) é uma técnica de aprendizado de máquina que visa melhorar a efetividade e robustez dos modelos preditivos, combinando as previsões de diferentes algoritmos de aprendizado de máquina. Ao contrário do método mais tradicional de ensembles, como o *Bagging* e o *Boosting*, que geralmente combinam

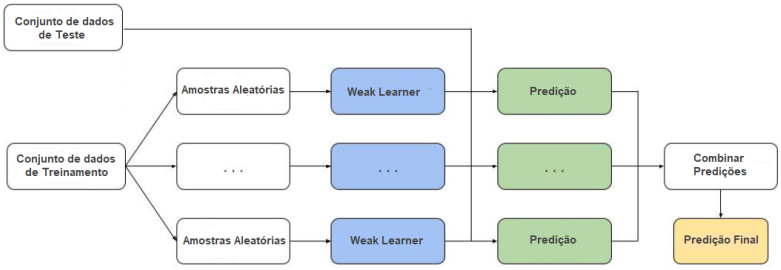


Figura 2.3: Estrutura da estratégia *bagging* [23]

várias instâncias do mesmo algoritmo, o Comitê por Empilhamento utiliza diferentes tipos de modelos para gerar as previsões, mais um modelo adicional para combinar essas previsões geradas, conforme detalhado a seguir [23].

Segundo Worlpert. [91], o processo do comitê por empilhamento geralmente envolve duas ou mais camadas de modelos:

- **Camada base** (*Level 0 space)*: Nesta etapa, um conjunto de diferentes algoritmos de aprendizado são treinados independentemente usando o mesmo conjunto de dados de treinamento. Cada modelo aprende padrões diferentes nos dados e faz previsões individualmente.
  Os modelos de base são treinados utilizando coleção de treinamento completo. Eles podem ser uma mistura de diferentes tipos de modelos de aprendizado de máquina, como árvores de decisão, SVMs, regressão logística, redes neurais, etc. Isso é feito para aproveitar a diversidade dos diferentes modelos.
- **Camada de Combinação** (*Level 1 space)*: Nesta etapa, o intuito é de combinar as previsões geradas pelos modelos na camada base, por isso é adicionado outro modelo na camada de combinação, chamado de "meta-modelo".
  Uma vez que os modelos de base são treinados (no conjunto de treinamento), eles são usados para fazer previsões utilizando a coleção de validação. As previsões desses modelos de base são então usados como atributos de entrada para o meta-modelo que é treinado usando os rótulos reais do conjunto de validação, conforme apresentado na Figura 2.4. Após o treinamento, o meta-modelo aprende a melhor forma de ponderar as previsões dos modelos da camada de base, a fim de alcançar uma previsão final mais efetiva.

Boosting

A estratégia *boosting* consiste em treinar modelos fracos de forma sequencial, utilizando os erros dos primeiros modelos treinados para fazer ajustes nos pesos dos próximos modelos [26]. Assim, o comitê aprende a ponderar as predições de cada modelo

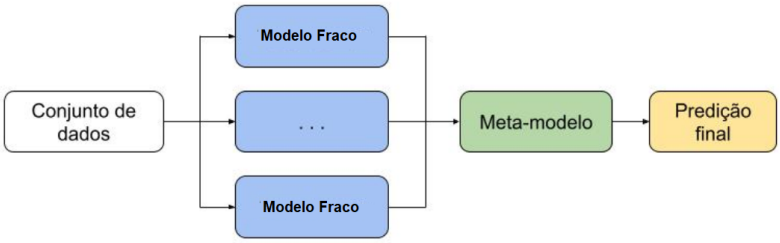


Figura 2.4: Estrutura da estratégia *stacking* [23]

fraco de forma que seja selecionado um subconjunto específico de instâncias para o treinamento do modelo fraco subsequente.

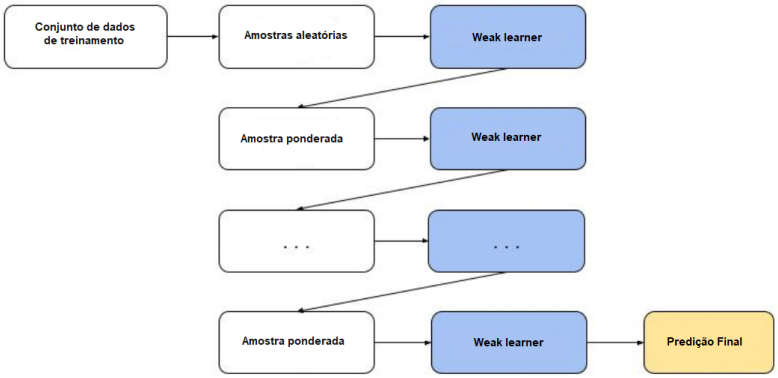


Figura 2.5: Estrutura da estratégia *boosting* [23]

Esse tipo de estratégia permite que os modelos fracos privilegiem instâncias em seu treinamento. Portanto, esse privilégio sobre instâncias que não foram bem avaliadas nos modelos anteriores levam a modelos subsequentes a treinar sobre regiões mais difíceis do problema.

2.5 Aprendizado profundo

Aprendizado profundo (*Deep learning*) é um subcampo do aprendizado de máquina que utiliza redes neurais artificiais com várias camadas, pois “Deep” refere-se à quantidade de camadas em uma rede neural. Por isso, as redes neurais artificiais também são conhecidas como redes neurais *feedforward* ou *multilayer perceptrons* (MLPs).

Nesta seção, abordaremos os temas sobre a arquitetura, o estudo focado em gradiente, a função de perda e a retropropagação (*backpropagation*) aplicadas no aprendizado profundo. Ian Goodfellow [32] descreve o aprendizado profundo da seguinte forma:

Arquitetura

O objetivo do aprendizado profundo é de gerar uma função f. Por exemplo, no caso de um classificador $f(x) = y$, onde uma entrada x atribui como saída a função f, uma categoria y.

No aprendizado profundo, $f(x)$ corresponde a uma função composta:

$$f(x) = f^{(d)}(\cdots(f^{(2)}(f^{(1)}(x))))$$

onde d é o grau de "profundidade" do aprendizado profundo, correspondendo ao número de níveis de composição da função composta obtida. Cada nível também é denominado *camada*, sendo que $f^{(1)}$ corresponde à primeira camada de aprendizagem e $f^{(d)}$ corresponde à última camada de aprendizagem. A primeira camada processa exemplos de entrada da coleção de treinamento representados por um vetor de atributos de características e gera dados de saída que correspondem a um nível de aprendizado em relação aos dados de entrada.

As informações de saída da primeira camada servem de entrada para a segunda camada que, por sua vez, geram informações adicionais de saída que correspondem a um segundo nível de aprendizado sobre os dados de entrada, e assim por diante. Portanto, a cada camada do aprendizado profundo, a técnica introduz uma representação (saída da camada) expressa em termos de outras representações mais simples (aquelas que correspondem à entrada da camada).

Logo, o aprendizado profundo permite a geração de um modelo que é capaz de obter conceitos complexos a partir de conceitos mais simples, formando, assim, uma hierarquia de conceitos. Tal hierarquia não pode ser obtida a partir das técnicas tradicionais, pois geram uma função não composta.

Na verdade, a arquitetura de uma rede neural é inspirada na estrutura do cérebro humano. Ela é formada por unidades interligadas, denominadas neurônios ou unidades, os quais são organizadas em diferentes camadas sequenciais.

Geralmente, uma rede neural consiste em uma camada de entrada, uma ou mais camadas ocultas e uma camada de saída. Cada neurônio em uma camada é conectado a todos os neurônios da camada seguinte através de "conexões sinápticas", cada uma com um peso associado.

A camada de entrada é responsável por receber os dados de entrada, enquanto as camadas ocultas processam essas informações, aplicando várias transformações nos dados, à medida que eles passam de uma camada para a próxima. Finalmente, a camada de saída produz a previsão ou classificação final do modelo, conforme descrito na Figura 2.6.

Desse modo, os neurônios são unidades computacionais que recebem uma ou mais entradas, multiplicam cada entrada por um peso, somam os resultados (às vezes com

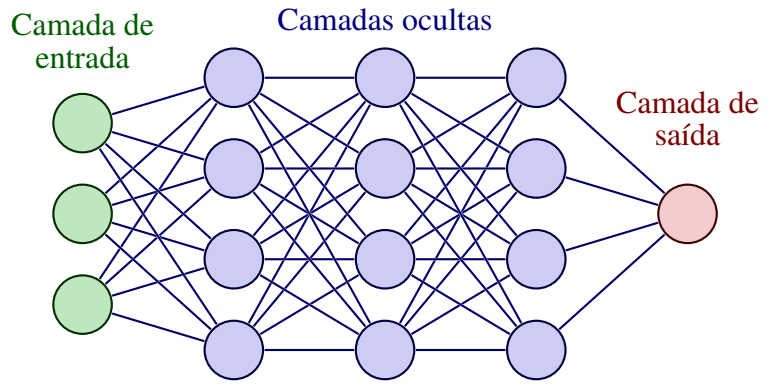


Figura 2.6: Estrutura de uma rede neural

a adição de um termo de viés), e depois passam essa soma através de uma função de ativação para produzir uma saída.

O cálculo realizado por um neurônio pode ser expresso pela seguinte equação:

$$f(w.x + b) = y$$

onde temos um produto escalar entre x (vetor de entrada) com o w (vetor de pesos). O resultado é somado com o b (viés). f é a função de ativação e y é a saída do neurônio.

Estudo focado em gradiente

Entre os modelos lineares tradicionais e as redes neurais tem um aspecto fundamental que diferencia as duas abordagens. Trata-se da não linearidade da rede neural. Isso torna muitas funções de perda significativas não convexas, contrastando com a natureza usualmente convexa dos problemas de otimização linear. Como resultado, as redes neurais são frequentemente treinadas com otimizadores iterativos baseados em gradientes que buscam minimizar a função de custo, em vez de solucionar equações lineares como ocorre na regressão linear, ou usar algoritmos de otimização convexa com garantias globais de convergência como na regressão logística ou nas SVMs.

A convergência de uma otimização convexa é garantida a partir de qualquer conjunto inicial de parâmetros. No entanto, a descida do gradiente estocástico, quando aplicada a funções de perda não convexas (como é o caso com redes neurais), não oferece essa garantia e é influenciada pelos valores dos parâmetros iniciais. Portanto, para as redes neurais do tipo *feedforward*, é essencial que todos os pesos sejam iniciados com valores aleatórios pequenos, enquanto os vieses podem ser iniciados em zero ou em pequenos valores positivos.

Na prática, a descida do gradiente é aplicada de forma iterativa durante o treinamento. Em cada passo, o algoritmo calcula o gradiente da função de perda com

relação aos parâmetros da rede, então ajusta os parâmetros na direção oposta ao gradiente.

Para redes neurais, este processo é mais complexo, pois a função de perda não é convexa devido a não linearidade da rede. Isso significa que existem muitos mínimos locais onde o modelo pode ficar preso em vez de encontrar o mínimo global. Portanto, as redes neurais geralmente são treinadas usando variações da descida do gradiente, como o gradiente descendente estocástico, que adiciona um elemento de aleatoriedade para evitar esses mínimos locais.

No pseudocódigo 1, é descrita a atualização do gradiente estocástico descente no treinamento de uma rede neural, onde x são os dados de entrada, y são os rótulos reais e L é a função de perda definida para o processo de aprendizagem.

Algorithm 1 Atualização do gradiente estocástico descente

Entrada: Taxa de aprendizagem $\epsilon_1, \epsilon_2, \cdots$
Entrada: Parâmetro inicial θ
$k \leftarrow 1$
while critério de parada não atendido **do**
  Seleciona um minibatch de n amostras da coleção de treinamento $\{b_1, \cdots, b_n\}\}$.
  Calcular estimativa de gradiente: $\hat{g} \leftarrow +\frac{1}{n}\nabla_\theta \sum_{i=1}^{n} L(f(b_i);\theta), y_i)$
  Aplicar atualição nos pesos da rede: $\theta \leftarrow \theta - \epsilon_k \hat{g}$
  $k \leftarrow k+1$

Função de perda

As funções de perda para redes neurais são muito parecidas com as que são usadas para outras técnicas de aprendizagem de máquina, como os modelos lineares. Na maioria das vezes, nosso modelo cria uma espécie de "mapa" de como os resultados (y) estão relacionados aos dados de entrada (x), dado um conjunto de parâmetros (θ). Esse "mapa" é, na verdade, uma distribuição de probabilidade, e nossa meta é torná-lo o mais preciso possível. Para fazer isso, usamos uma abordagem chamada de máxima verossimilhança, que, na prática, significa que tentamos minimizar a diferença (ou entropia cruzada) entre nossas previsões e os dados reais de treinamento.

No entanto, em vez de prever a distribuição de probabilidade completa, tentamos prever apenas algum resumo ou característica particular dos resultados, dado os dados de entrada. Nesses casos, usamos funções de perda especializadas para treinar nosso modelo.

Retropropagação

O algoritmo de retropropagação (backpropagation), basicamente, permite que a informação da perda volte pela rede neural para calcular o gradiente, que é uma medida de como a função de perda muda quando alteramos os parâmetros da rede.

De forma simplificada, o gradiente é como a inclinação do terreno em um ponto específico e a retropropagação é como um "mapa" que nos mostra para onde ir para alcançar o pico (minimizar a perda) o mais rápido possível.

Porém, calcular a expressão analítica para o gradiente tem um custo computacionalmente alto. Felizmente, o algoritmo de retropropagação minimiza o custo computacional utilizando um procedimento simples e economicamente eficaz.

No pseudo-código 2, é demostrada a etapa de *backpropagation*, onde em um número M de épocas, uma coleção de dados de treinamento D é dividida em um número N de mini-batchs, sendo $D = \{b_1, \cdots, b_n\}$, onde em cada etapa do treinamento do modelo é utilizada uma amostra de mini-batch b_i para calcular o gradiente e atualizar os pesos da rede neural.

Algorithm 2 Pseudo-código do backpropagation

Entrada: Coleção de dados de treinamento $D = \{b_1, \cdots, b_n\}$, Número de épocas M
for k = 1 : M **do**
    for b : D **do**
        Calcular o gradiente para cada peso e viés
        Usar gradientes para atualizar cada peso e viés

2.5.1 Aprendizado por Comitê de Redes Neurais Profundas

Allen-Zhu e LI [3] afirmam que utilizar simplesmente a média das saídas de poucas redes neurais (3 ou 10), treinadas de modo independente, mas que utilizam a mesma arquitetura, o mesmo conjunto de dados, é suficiente para obter obter uma melhora considerável na predição no conjunto de teste, quando comparado a apenas uma dessas redes neurais. Os autores consideram que um comitê de modelos com as características citadas acima, possui a capacidade de reconhecer e considerar diferentes características ou perspectivas nos dados. Eles denominam essa capacidade de *multi-visão* dos dados de entrada.

Uma única rede neural tem uma limitação ao lidar com a diversidade em um vetor de características. Por exemplo, considere um vetor de características $f \in \{f_1, f_2, f_3, f_4\}$. Durante o treinamento, uma única rede neural tende a se concentrar em um subconjunto específico de características (uma "visão única") para fazer previsões. Neste caso, por exemplo, $f' \in \{f_1, f_2\}$ seria selecionado para classificar o primeiro rótulo e $f'' \in \{f_3, f_4\}$ para o segundo rótulo. Isso leva à correta classificação de 90% dos exemplos de treinamento, incluindo todos os dados com multi-visão e dados com visão única (as características f' ou f''). Entretanto, após estes exemplos serem classificados corretamente, eles contribuem minimamente para futuras atualizações da rede, já que a perda de entropia cruzada é pequena para esses exemplos.

Em contraste, um comitê de redes neurais pode ser capaz de considerar todas as características $\{f_1, f_2, f_3, f_4\}$, pois diferentes redes dentro do comitê podem focar em diferentes subconjuntos dessas características e quando combinadas essas múltiplas visões se completam.

Outros diferentes métodos por comitê foram explorados [89, 33, 9, 52, 44], contudo, o padrão de fato continua sendo modelos de comitê por aprendizado profundo [52]. Porém, o seu alto custo computacional, tanto em processamento quanto em memória, cresce diretamente proporcional ao número de componentes do comitê, durante o treinamento e também durante a inferência. Por essa razão, o comitê por aprendizado profundo é inviável computacionalmente em várias aplicações [84].

2.6 Viés e Variância

Em aprendizado de máquina supervisionado, o erro de predição pode ser decomposto sob a perspectiva de viés e variância [39]. Ao se buscar um menor valor para ambos, na verdade, procura-se encontrar a melhor complexidade para um modelo. Porém, a relação entre esses dois componentes do erro de predição é uma relação de *tradeoff*. Geralmente, uma técnica de aprendizado diminui um desses componentes em detrimento do aumento do outro erro componente.

Nos problemas de classificação e regressão[3], o viés é considerado a diferença média entre o valor predito pelo modelo e o valor correto. Um alto viés indica um modelo simples, com poucos parâmetros de ajuste ou poucas regras aprendidas. Isto é, o modelo apresenta complexidade insuficiente para se ajustar aos dados de treinamento e, portanto, não aprende adequadamente. Para resolver esse problema, aumentamos a complexidade do modelo, reduzindo assim o viés. Isto leva a um maior aprendizado durante o treinamento e possivelmente a um menor erro de treinamento.

A variância, por sua vez, é a dispersão dos valores preditos para instâncias individuais dos dados [39]. Modelos com alta variância apresentam diferentes predições para os dados em diferentes treinamentos. Isto é, o modelo é muito complexo e se ajusta excessivamente aos dados de treinamento, um fenômeno conhecido como sobreajuste, perdendo inclusive, a capacidade de generalização sobre dados de teste. Ou seja, a redução da variância é alcançada reduzindo a complexidade do modelo.

O erro total do modelo, então, pode ser minimizado através do balanceamento entre diminuir o viés e a variância. No entanto, esse equilíbrio é conflitante, pois a redução do viés ocorre com o aumento da complexidade do modelo, enquanto a redução da

[3] e também no caso da categoria *pointwise* de AR, pois nesse caso, o problema AR pode ser visto como um caso de regressão (inferir o valor de relevância de um documento).

variância acontece com a simplificação da complexidade do modelo. Este *tradeoff* reflete o impacto da complexidade do modelo sobre o erro de treinamento e de teste.

Ao avaliar diferentes configurações de um mesmo modelo (vários treinamentos), a análise de viés e variância fornece uma melhor compreensão sobre as alterações de comportamento e de complexidade que cada configuração impõe ao modelo. Ferramentas como a validação cruzada e técnicas de regularização podem ser usadas para encontrar um ponto ótimo de complexidade do modelo que resulta na melhor capacidade de generalização, minimizando simultaneamente o viés e a variância.

A Figura 2.7-A mostra um problema que ocorre no treinamento e teste, conforme a complexidade do modelo aumenta, mas o erro de treinamento tende a diminuir à medida que o modelo se ajusta cada vez mais aos dados de treinamento. No entanto, com muito ajuste, o modelo se adapta muito aos dados de treinamento e apresenta dificuldades em generalizar para novos dados, ou seja, possui um alto erro no teste.

No entanto, Belkin et al. [7] discute que em modelos modernos (como as redes neurais) sobre-parametrizados podem ter outra interpretação da métrica tradicionalmente utilizada de viés e variância descrita na Figura 2.7-A.

A métrica "tradicional" de viés e variância busca um equilíbrio para minimizar o erro total. No entanto, em [7] sugere-se que os modelos sobre-parametrizados, que deveriam sofrer de alta variância (e, portanto, sobreajuste), ainda podem ter uma boa efetividade nas coleções de teste. Este fenômeno é uma contradição com o pensamento tradicional.

A curva de risco de descida dupla, descrita na Figura 2.7-B, é uma caracterização da efetividade de um modelo de aprendizado de máquina à proporção que sua complexidade aumenta. No início, à medida que o modelo se torna mais complexo, a efetividade do modelo melhora até chegar a um ponto chamado de limite de interpolação, que é um lugar onde um modelo tem capacidade suficiente para se ajustar perfeitamente aos dados de treinamento. No entanto, se a complexidade do modelo aumentar além do ponto ideal, então os modelos se tornam sobre-parametrizados.

Quando falamos em modelos "sobre-parametrizados" em aprendizado de máquina, estamos nos referindo a modelos que têm mais parâmetros do que o número de exemplos de treinamento. Teoricamente, tais modelos têm capacidade suficiente para memorizar completamente os dados de treinamento, incluindo qualquer ruído que possa estar presente. Essa situação é frequentemente associada ao fenômeno de "sobreajuste", no qual um modelo se ajusta tão bem aos dados de treinamento que se torna menos eficaz para generalizar a novos dados.

Porém, à medida que a complexidade do modelo continua a aumentar além do limite de interpolação, o erro de teste começa a diminuir novamente, criando a segunda descida da curva de descida dupla. O motivo para isso é que conforme o modelo se

torna mais complexo é que, além dele ser capaz de encontrar funções que se ajustam perfeitamente aos dados de treinamento, essas funções são “mais simples” de acordo com alguma medida de complexidade.

O “mais simples” está relacionado com o princípio da *navalha de Occam* [88], em que sugere que uma função com uma norma menor, geralmente, se ajusta melhor aos dados.

Isso significa que os modelos mais complexos são capazes de encontrar funções que representam bem os dados de treinamento, mas também possuem alguma “simplicidade” inerente que lhes permite generalizar bem para dados de teste. Este fenômeno é responsável pela segunda descida na curva de descida dupla.

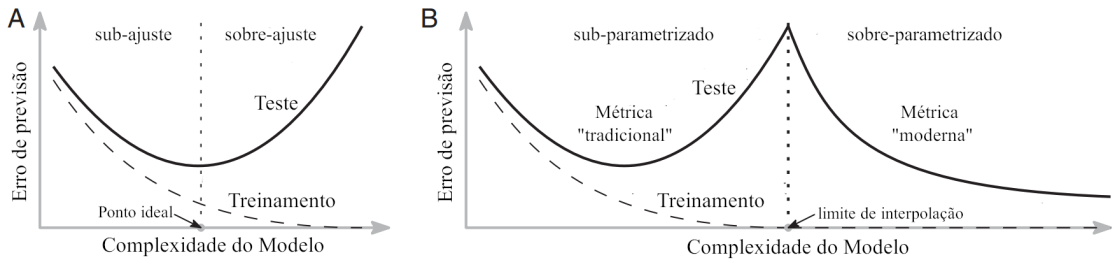


Figura 2.7: Erro no treino e teste em função da complexidade do modelo [7]

2.6.1 Viés e Variância em AR

Em AR, a decomposição do erro total em termos de viés e variância deve ser específica para tarefas de ranqueamento. Ou seja, ao decompor o erro em viés e variância no contexto do Aprender a ranquear, é necessário levar em consideração a natureza da função de perda utilizada no aprendizado do ranqueamento. Portanto, precisamos adaptar a maneira como medimos o viés e a variância que sejam pertinentes à tarefa de ranqueamento, em vez de simplesmente usar as definições padrão de viés e variância que são aplicáveis a outras tarefas de aprendizado de máquina (regressão ou classificação).

Shivaswamy e Chandrashekar [75] destacam que as métricas padrão utilizadas em outras áreas do aprendizado de máquina podem não ser totalmente aplicáveis ou revelar todas as nuances relevantes para o problema de AR. Foi observado que a tarefa de ranqueamento difere de muitas outras tarefas de aprendizado de máquina, visto que não se trata apenas de fazer previsões precisas, mas também de ordenar corretamente um conjunto de documentos. Isso implica que a maneira como o viés e a variância se manifestam e afetam o desempenho do modelo pode ser diferente. Portanto, os autores propõem uma definição de viés e variância, específica para AR, utilizando a abordagem em pares (pairwise) dos documentos, demonstrando que as discrepâncias de

ranqueamento entre as ordenações verdadeiras e uma função de ranqueamento podem ser decompostas em componentes de viés e variância da seguinte forma:

- **Viés**: medida da distância entre o ranqueamento ideal e o ranqueamento produzido pelo modelo, isso é, mensura premissas incorretas durante a produção do modelo em relação ao relacionamento entre rotulação e predições.
- **Variância**: inconsistência nas ordens de ranqueamento que um modelo produz para o mesmo par de documentos, quando treinado em diferentes coleções de dados. Ou seja, mensura a sensibilidade do modelo a pequenas variações na coleção de treino. Além disso, trata-se da frequência com que um modelo produz ranqueamentos diferentes em treinamentos diferentes.

O erro de ranqueamento de um modelo f sobre um conjunto de documentos D relacionados a uma mesma consulta e os valores de referência y, pode ser definido como:

$$error_D = rbias_D(f, y)^2 + var_D(f)$$

,
onde $rbias_D(f, y)$ é o viés, dado pelas probabilidades sobre os pares de documentos $(d_i, d_j) \in D$:

$$rbias_D(f, y) = \sqrt{P[f(d_i) > f(d_j)] - P[y_i > y_j]}$$

E o termo $var_D(f)$ representa a variância que é definida por:

$$var_D(f) = P[f(d_i) > f(d_j)]P[f(d_i) \leq f(d_j)]$$

Para todos os pares de documentos associados $(d_i, d_j) \in D$, podemos definir o viés como a diferença entre a probabilidade de d_i preceder d_j nas ordenações produzidas pelos modelos e a ordenação ideal indicada pelos valores de referência. Já a variância é determinada pela inconsistência nas ordenações geradas pelos modelos, refletindo a chance de d_i ser ranqueado antes de d_j em comparação com a probabilidade do evento oposto ocorrer. Por sua vez, a variância é a probabilidade de d_i preceder d_j em relação à probabilidade do oposto nos ranqueamentos produzidos pelos modelos treinados em conjuntos de treinamentos distintos.

Durante a avaliação de um algoritmo, essas probabilidades são quantificadas por meio da contagem de ocorrências do respectivo evento, em relação a todas as possíveis ordenações entre o par de documentos em análise, em diferentes treinamentos.

Portanto, é interessante calcular o viés e a variância em algoritmos de AR, pois possibilita uma melhor compreensão da superioridade de determinados algoritmos em relação a outros, embasando-se por uma métrica de avaliação.

CAPÍTULO 3

Trabalhos Relacionados

Conforme comentado na seção 2.5.1, Os comitês de redes neurais profuntas (*Deep ensemble*) apresentam efetividade superior a uma única rede neural. Porém possuem alto custo computacional, já que requer treinar e armazenar várias redes neurais separadas [30]. Para abordar esses problemas de eficiência, os pesquisadores têm buscado outras abordagens. Uma dessas técnicas é o Monte-Carlo Dropout (MC-Dropout), proposto por Gal e Ghahramani (2016) [30]. O MC-Dropout trabalha com a técnica de Dropout, bastante utilizada durante o treinamento de redes neurais para evitar o sobreajuste. Assim, ao invés de desligar os neurônios apenas durante o treinamento, o MC-Dropout também os desativa durante a inferência, realizando várias passagens pelos dados e, em seguida, agrega as saídas para formar uma distribuição preditiva. Isso permite uma quantificação mais precisa da incerteza do modelo, sem a necessidade de treinar várias redes separadas.

O Multi-Sample Dropout (MSD) é um método que também utiliza o Dropout para melhorar a efetividade do modelo, mas em vez de realizar várias passagens completas pelos dados durante a inferência, o Multi-Sample Dropout realiza várias passagens paralelas por um único minibatch de dados. Isso permite obter uma aproximação razoável da distribuição preditiva com menos recursos computacionais [45].

Embora o MC Dropout e o Multi-Sample Dropout tenham suas vantagens, pesquisas recentes introduziram abordagens ainda mais eficazes, como o BatchEnsemble [90] e o Masksembles [24]. Essas técnicas buscam combinar a robustez e a precisão das técnicas ensemble com a eficiência computacional das técnicas de dropout.

O BatchEnsemble [90] utiliza uma matriz de pesos de um “Posto de uma Matriz” (em inglês, rank-one) para criar várias “cópias” do modelo, cada uma com seus próprios pesos. Essa técnica permite a quantificação da incerteza, como em um comitê, mas sem o custo de armazenamento de vários modelos separados.

O Masksembles [24] estende a ideia do BatchEnsemble ao usar várias máscaras para alterar a conexão entre as camadas das redes neurais, criando, assim, múltiplas variantes de um único modelo (Huang et al., 2022). Essa técnica permite maior diversidade nos modelos, sem aumentar o custo computacional.

Alguns dos trabalhos relacionados como Multi-sample Dropout, Multi-sample Dropout (Treino e Teste), Masksemble e BatchEnsemble, apresentados nas seções 3.4, 3.6 e 3.7 propuseram estruturas semelhantes ou mais sofisticadas que ao Monte Carlo Dropout [30] para tornar viável o uso de comitê em aprendizado profundo. De agora em diante neste trabalho, chamaremos essas estruturas apresentadas nas seções 3.4, 3.6 e 3.7 de **comitê de sub-redes neurais**. Esse nome se deve ao fato de que sub-redes são criadas a partir de uma rede neural e essas sub-redes são tratadas como componentes de um comitê.

Até recentemente, as estratégias baseadas em comitê, como o LambdaMART, eram o estado da arte em aprender a ranquear. Porém, os avanços recentes na área de Aprendizado Profundo têm sido usados para melhorar as técnicas de AR. Pobrotyn et al. [68] propuseram um um modelo chamado AllRank. Este modelo não utiliza a estratégia de ensembles, mas uma rede neural com o mecanismo de atenção, chamado *self-attention*. Atualmente o AllRank é o estado da arte na área de L2R.

A motivação deste trabalho é a de experimentar a combinação do AllRank [68] com diversas propostas de comitês de sub-redes. Então, descrevemos os métodos de comitê de sub-redes neurais e explicamos como eles foram implementados no modelo base AllRank.

Como um ponto de observação, os métodos de comitê de sub-redes neurais são agnósticos ao modelo AllRank, isto é, eles podem ser utilizados em qualquer arquitetura de Multi-layer perceptron. Porém, nossa intenção é o de comprovar se há uma melhora nos resultados, medindo a efetividade e a eficiência dos modelos dos métodos de comitê de sub-redes neurais aplicados ao AllRank. A seções subsequentes tratam de trabalhos que estão relacionados a essa dissertação.

3.1 Dropout

A técnica de aprendizado profundo, normalmente, possui redes com muitas camadas e neurônios. Por um lado, isso torna o processo de aprendizado do modelo muito poderoso, mas, por outro lado, esse tipo de arquitetura acaba tendo um sério problema com o sobreajuste no conjunto de dados de treinamento. Para minimizar este problema foi proposto um método de regularização chamado *Dropout*. Durante o treinamento, o Dropout "desliga" aleatoriamente parte dos neurônios na camada de uma da rede neural. Isso evita que os neurônios se adaptem demais ao treinamento, o que reduz drasticamente o sobreajuste causado no conjunto de dados e melhora a eficácia do aprendizado do modelo [79].

A técnica de Dropout oferece um método de regularização computacionalmente barato e muito eficaz para melhorar o erro de generalização e reduzir o sobreajuste em

redes neurais profundas. As amostras de Dropout aplicadas nas camadas da rede neural formam uma sub-rede de neurônios ativos. Durante o treinamento, a cada passagem pelo *feed-forward* uma nova sub-rede é gerada e treinada.

Portanto, o Dropout consegue gerar diferentes arquiteturas durante o treinamento, o que consiste em todas sub-redes neurais que podem ser formadas pela desativação dos neurônios a partir de uma rede neural [32], conforme Figura 3.1.

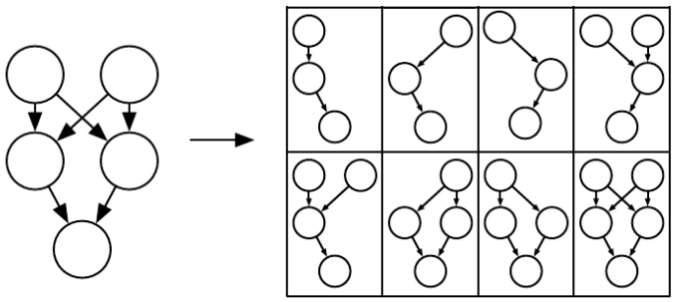


Figura 3.1: Sub-redes formadas pelo Dropout durante o treinamento [32]

Para entender o seu funcionamento, considere uma rede neural com H camadas ocultas, onde $h \in \{1 \cdots H\}$. Considere também o Dropout como um vetor m gerado por uma distribuição de *Bernoulli* que possui variáveis randômicas no espaço amostral $\{0, 1\}$, em que 0 define se o neurônio está desativado e 1 se está ativo. Logo, para cada amostra de Dropout que for aplicada em uma camada h é gerado um vetor m_h. Portanto, se x_h é o vetor de saída de uma camada h, o produto entre $m_h * x_h$ gera um subconjunto $\overline{x_h}$ formado de unidades ativas de x_h. Esse subconjunto $\overline{x_h}$ é então usado como entrada para a próxima camada $(h + 1)$ [79]. O Dropout em uma arquitetura de *feed-forward* pode ser expresso pela seguinte equação a seguir:

$$\begin{aligned} m_h &\sim Bernoulli(p) \\ \overline{x_h} &= m_h * x_h \\ o_{h+1} &= w_{h+1}.\overline{x_h} + b_{h+1} \\ y_{h+1} &= f(o_{h+1}) \end{aligned} \tag{3-1}$$

No final, aplicando o Dropout em todas as camadas ocultas h, é gerada uma sub-rede neural formada a partir de uma rede neural base das unidades que estão ativas. Para o treinamento, as derivadas da função de perda são retro-propagadas através da sub-rede [79].

As redes neurais que utilizam Dropout podem ser treinadas usando gradiente descendente estocástico de maneira semelhante às redes neurais padrões. A única diferença é que, para cada caso de treinamento em um mini-batch, é formada uma sub-rede

neural estruturada por eliminar um subconjunto de neurônios desta mesma rede neural. O *feed-forward* e o *backpropagation* para essa etapa do treinamento é feita apenas nesta sub-rede neural, enquanto os gradientes para cada parâmetro são calculados na etapa de treinamento para cada mini-batch [79].

3.2 AllRank

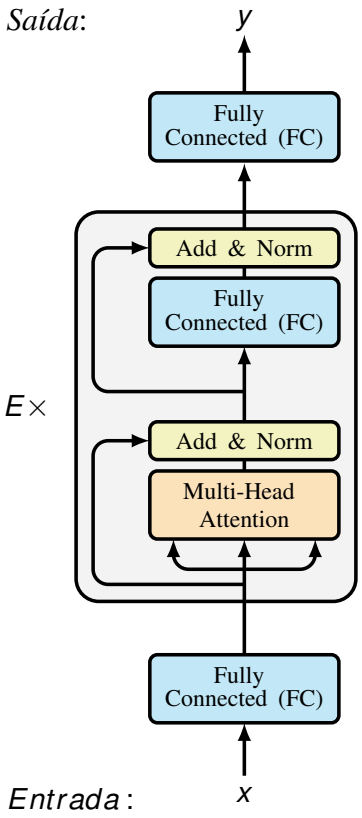
Até recentemente, os métodos de AR *pair-wise* e *list-wise* consideravam a relação inter-itens dos itens a serem ranqueados apenas na avaliação da função de perda ou de erro do modelo. Nessas classes de métodos, as funções de perdas (*loss functions*) medem o erro de posicionamento no ranque gerado pela função de pontuação que atribui um valor a cada documento. Esse erro é então utilizado para corrigir os parâmetros do modelo. Porém, a relação inter-documentos (às vezes referida como contexto) não era considerada pela função de pontuação dos documentos. Isto é, a função de pontuação, interna aos métodos de AR, computava o valor de relevância de um documento apenas com base nos atributos do par documento-consulta.

Após o surgimento do mecanismo de atenção [86], inicialmente proposto para o problema de tradução de textos, propostas começaram a surgir no sentido de adaptar esse mecanismo para o problema de ranque de documentos. [66, 63, 68]. Em especial, Pobrotyn et al. [68] propuseram o AllRank, um método de AR, no qual a função de pontuação dos documentos é treinada de modo a considerar não apenas os atributos de um único documento, mas também os valores dos atributos dos demais documentos que concorrem com ele na lista de documentos associados a uma consulta específica.

O AllRank utiliza o mecanismo de autoatenção (self-attention) para capturar as relações complexas entre os atributos dos diferentes documentos de uma determinada consulta. Isto permite que o modelo aprenda a atribuir pesos ou importâncias no ranqueamento de um documento considerando também as características dos outros documentos desta mesma consulta. Com isso, o AllRank conseguiu superar o método de AR estado da arte na época de sua publicação, que era o LambdaMart [15].

O AllRank [68] utiliza a arquitetura *self-attention* [86] na sua configuração de ranqueamento da seguinte maneira. A lista de documentos de uma determinada consulta é tratada como *tokens*[1] e dada como entrada na rede neural. Considere o tamanho da lista de entrada como l e o número de atributos de cada documento como d_f. Primeiramente, cada documento é passado por uma camada de entrada totalmente conectada (FC) de tamanho d_{input}. Em seguida, a saída da camada de entrada é passada para um codificador *self-attention* com um número de E blocos. Um bloco consiste em um número de H heads

[1] *token* - é a representação de uma unidade de informação por um vetor em um espaço n-dimensional



$$f(x) = FC(\underbrace{Encoder(Encoder(\cdots Encoder}_{E\times}(FC(x)))))$$

onde

$$Encoder(x) = LayerNorm(z + Dropout(FC(z))),$$

$$z = LayerNorm(x + Dropout(MultiHead(x)))$$

Figura 3.2: Arquitetura do modelo AllRank

com camada oculta de dimensão d_{hidden}. Conforme explicado no *self-attention* [86], um bloco codificador consiste em uma camada de atenção de vários *heads*, uma conexão residual seguida por uma camada de normalização, uma camada totalmente conectada (FC) e outra conexão residual seguida de uma camada de normalização. É aplicado o Dropout na camada de atenção e na camada FC. Finalmente, após E blocos do codificador, uma camada totalmente conectada (FC) de tamanho d_{output} é usada como saída da rede para calcular uma pontuação para cada documento. A figura 3.2 mostra de forma detalhada o esquema da arquitetura.

Portanto, ao usar o *self-attention* no codificador, é garantido que, no cálculo da pontuação de um determinado documento, a representação de todos os outros desta mesma consulta é considerada também. As representações obtidas pela rede neural, juntamente com os rótulos verdade, fornecem informações do contexto específico em que a consulta é realizada para a *Loss function*, levando a um cálculo de perda mais preciso. Mas depois do treinamento do modelo, o método consegue obter uma pontuação com maior exatidão de cada documento. Com isso, após a ordenação pela pontuação dos documentos, é esperado obter uma lista com maior acurácia neste ranqueamento.

3.3 Monte-Carlo Dropout

Yarin Gal e Zoubin Ghahramani [30] fornecem uma interpretação matemática do uso do dropout em redes neurais como uma aproximação de um processo Bayesiano. Mas a interpretação chave é baseada em conceitos probabilísticos e de inferência estatística.

No contexto de redes neurais, o dropout é uma técnica que envolve "desligar" aleatoriamente uma proporção dos neurônios durante o treinamento para evitar o sobreajuste. Matematicamente, isso pode ser representado pela multiplicação do vetor de saída de uma camada da rede por um vetor binário aleatório (denominado máscara de dropout), onde a probabilidade de um elemento ser zero é o parâmetro de dropout p.

A contribuição central do Monte-Carlo Dropout (MC-Dropout) é a proposta de que o dropout pode ser interpretado como uma aproximação para a inferência Bayesiana em redes neurais. Ao usar dropout em uma rede neural, estamos, essencialmente, simulando o comportamento de muitas redes neurais possíveis e diferentes entre si. Quando olhamos para essa coleção de redes em um contexto probabilístico, cada uma delas tem uma certa probabilidade (ou "peso") baseada em sua complexidade. Deste modo, no momento da inferência, a incerteza é calculada usando a variabilidade nas previsões e é feita a média das previsões geradas para calcular uma previsão final.

Em um aproximador bayesiano, a incerteza nos parâmetros é modelada como uma distribuição de probabilidade, que é atualizada à medida que novos dados são observados. Essa distribuição pode ser usada para calcular a probabilidade de diferentes valores dos parâmetros da função, o que permite estimar a incerteza na previsão do modelo.

Um exemplo de aproximador bayesiano é a regressão linear bayesiana, em que os parâmetros do modelo são tratados como variáveis aleatórias e uma distribuição a priori é atribuída a eles. À medida que novos dados são observados, a distribuição posterior dos parâmetros é atualizada, permitindo estimar a incerteza na previsão do modelo [89].

Os aproximadores bayesianos têm a vantagem de fornecer uma estimativa da incerteza associada à previsão do modelo, o que é útil em muitas aplicações, como diagnósticos médicos, detecção de fraudes e reconhecimento de fala. No entanto, eles também têm a desvantagem de serem computacionalmente caros, já que envolvem o cálculo de distribuições de probabilidade em tempo real. Por isso, técnicas como o Dropout em redes neurais artificiais foram propostas como aproximações mais eficientes para modelar a incerteza em grandes conjuntos de dados.

3.4 Multi-Sample Dropout

O conceito de Multi-sample Dropout (MSD) [45] estende a ideia básica do dropout. Em vez de criar apenas um subconjunto aleatório de neurônios desativados (por uma máscara de dropout), o multi-sample dropout cria várias dessas máscaras em cada iteração de treinamento. Isto é, em uma única iteração, são gerados múltiplos subconjuntos aleatórios de neurônios desativados, e para cada um desses subconjuntos, a perda (ou erro) é calculada. Após calcular a perda para todas essas amostras de dropout, é feita a média das perdas para obter a perda final dessa iteração.

Uma máscara é a representação de uma matriz que possui valores binários. Ao ser aplicada nas camadas ocultas de uma rede neural, desativa parte dos neurônios dessa rede. Os neurônios que permaneceram ativos, após a aplicação da máscara, formam um subconjunto desta mesma rede, ou seja, é gerada uma sub-rede neural.

Inoue [45], em sua abordagem com MSD, utilizou o mesmo *batch* de dados como entrada para as várias sub-redes neurais em cada iteração de treinamento. Porém, em nosso estudo, inspirados pela estratégia *bagging* (seção 2.4.1), adaptamos essa estratégia no MSD. Então, em uma única iteração de treinamento, dividimos um *mini-batch* de dados entre as diferentes sub-redes neurais, ou seja, cada sub-rede neural possui uma entrada de dados diferente, promovendo ainda mais a diversidade entre elas.

Considere que máscara m seja uma matriz randômica de probabilidade p que foi gerada por uma distribuição $m \sim Bernoulli(p)$. Cada variável de m tem a probabilidade p de ser 1 que significa que o neurônio deve continuar ativo e se for 0 deve ser desativado [79]. Este processo pode ser repetido M vezes, gerando um conjunto de máscaras $\{m^1, m^2, \cdots, m^n\}$. Esta máscara m_i é multiplicada com as saídas das camadas ocultas de uma rede neural, desativando um subconjunto de neurônios e gerando a sub-rede neural. No final, cada máscara m_i aplicada nas camadas ocultas da rede neural, gera um número M de sub-redes neurais, conforme descrito na figura 3.3. Os nós da rede em vermelho são os neurônios desativados pela máscara e os azuis são os neurônios ativos.

Durante o treinamento, a cada iteração, um *batch* de dados x é divido em um número n de subconjunto de dados $\{x_1, x_2, \cdots, x_n\}$, sendo $1 \leq i \leq n$. Cada sub-rede neural recebe x_i e gera um valor de predição p_i. Então, é utilizada uma função de perda para calcular o valor de perda l_i cometido na predição p_i de cada sub-rede neural. Após calcular a perda para todas as sub-redes neurais, é feita a média para obter a perda final l_{final} dessa iteração, conforme descrito na Figura 3.4.

Ao calcular a média das perdas de todas essas máscaras de dropout, a rede obtém uma estimativa mais robusta e suavizada do gradiente, o que pode levar a atualizações de pesos mais estáveis e consistentes durante o treinamento. Esse processo amplifica a regularização proporcionada pelo dropout tradicional, ajudando a mitigar o sobreajuste,

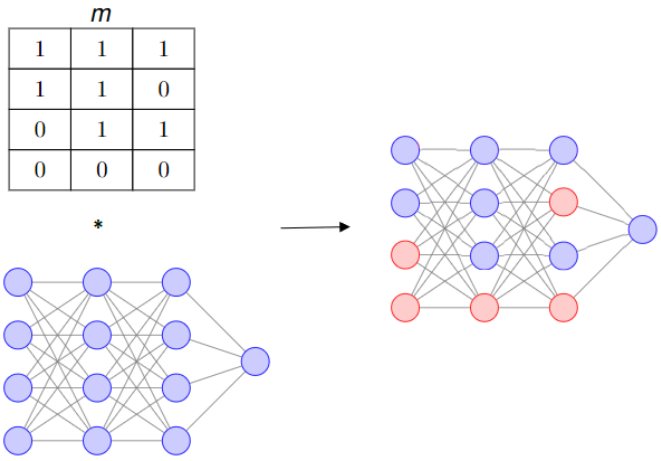


Figura 3.3: Sub-rede neural gerada a partir da aplicação da máscara

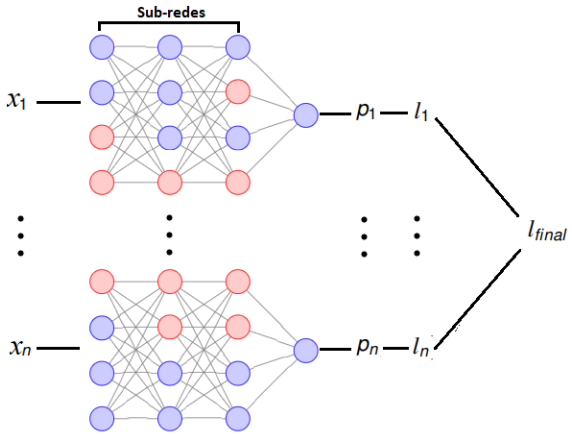


Figura 3.4: Sub-redes geradas na fase de treinamento

tornando as atualizações dos pesos menos oscilantes e, consequentemente, promovendo uma melhor generalização do modelo.

A implementação do MSD no modelo base AllRank é semelhante ao diagrama mostrado na Figura 3.2, em que temos um Dropout aplicado na camada totalmente conectada (FC) e outro aplicado na camada de *self-attention* (MultiHead). O que diferencia as duas implementações é que, no AllRank temos somente uma máscara de Dropout aplicada a cada iteração do treinamento, enquanto no MSD são aplicadas um número n de máscaras de Dropout a cada iteração, gerando um número n de sub-redes, conforme mostrado na Figura 3.4. Além disso, conforme já mencionado sobre o MSD, a cada iteração durante o treinamento, dividimos um *mini-batch* em n subconjuntos e cada sub-rede recebe um subconjunto de dados diferente, enquanto no AllRank temos somente uma entrada de dados para o modelo.

3.5 Multi-Sample Dropout (Treino e Teste)

O multi-sample dropout (MSD) [45] representa uma técnica na qual várias máscaras de dropout são criadas em cada iteração de treinamento. Em busca de refinamentos nessa abordagem, neste estudo, introduzimos uma variante do MSD denominada Multi-sample Dropout (Treino e Teste) (MSD-TT). Diferentemente do MSD que se concentra apenas no treinamento, o MSD-TT estende a geração de múltiplas máscaras de dropout para a fase de teste. Durante o teste, vários subconjuntos aleatórios de neurônios são desativados por máscaras de dropout, e para cada um desses subconjuntos (ou sub-redes neurais), uma previsão é elaborada. Ao final do ciclo, a previsão final é obtida através da média de todas as previsões individuais.

A abordagem do MSD-TT em utilizar as máscaras de dropout também durante o teste (ou inferência) foi inspirada no trabalho do Monte Carlo Dropout [30]. Assim como no Monte Carlo, a média das previsões é utilizada. A ideia subjacente é que ao fazer a média de várias previsões, se reduz a variância e se captura melhor a incerteza do modelo, fornecendo uma estimativa mais robusta e confiável.

3.6 Masksembles

O MC-Dropout [30], ao gerar múltiplas previsões para uma dada entrada, utiliza medidas como a variância para estabelecer estimadores de incerteza. No entanto, apresenta lacunas inerentes à sua abordagem. As máscaras, dentro da metodologia MC-Dropout, tendem a se sobrepor significativamente, o que pode resultar em previsões altamente correlacionadas e uma subestimação potencial da incerteza. Além disso, a ressistematização constante das máscaras em cada iteração de treinamento obriga cada unidade da rede a formar uma resposta coerente com qualquer outra unidade. Isso cria um efeito de mistura e conduz à uniformidade nas previsões de diferentes máscaras [24, 93].

O Masksembles [24] busca aproveitar as vantagens tanto do MC-Dropout [30] quanto do Deep Ensemble [52], enquanto procura mitigar as desvantagens associadas a cada abordagem. Ele não é um puro comitê como o Deep Ensemble, que treina vários modelos independentes, nem como o MC-Dropout, que usa um único modelo com dropout estocástico. Deste modo, o Masksembles busca se tornar uma solução subótima entre essas duas abordagens.

Em vez de depender de máscaras geradas aleatoriamente, a estratégia Masksembles utiliza um conjunto finito de máscaras binárias pré-definidas. Elas são concebidas para garantir um controle mais apurado sobre a sobreposição, possibilitando a exclusão de ativações relevantes da rede. O resultado é uma gama de sub-redes neurais que não

sofrem dos efeitos de mistura excessiva de neurônios, como observadas em abordagens anteriores, como o MC-Dropout.

A abordagem Masksembles visa garantir que um número N de máscaras pré-definidas se alinhem adequadamente com as ativações da rede. Para alcançar esse objetivo, a camada que recebe o Masksembles é frequentemente ajustada em tamanho. Um exemplo é fornecido pela Figura 3.5, onde uma rede Multi-layer perceptron (MLP), com uma camada oculta, é retratada. A camada Masksembles é integrada após esta camada oculta ter seu número de neurônios para que o tamanho da camada se iguale ao tamanho da máscara. Durante o processamento, as máscaras são aplicadas e os componentes que carregam um valor zero são efetivamente "desligados".

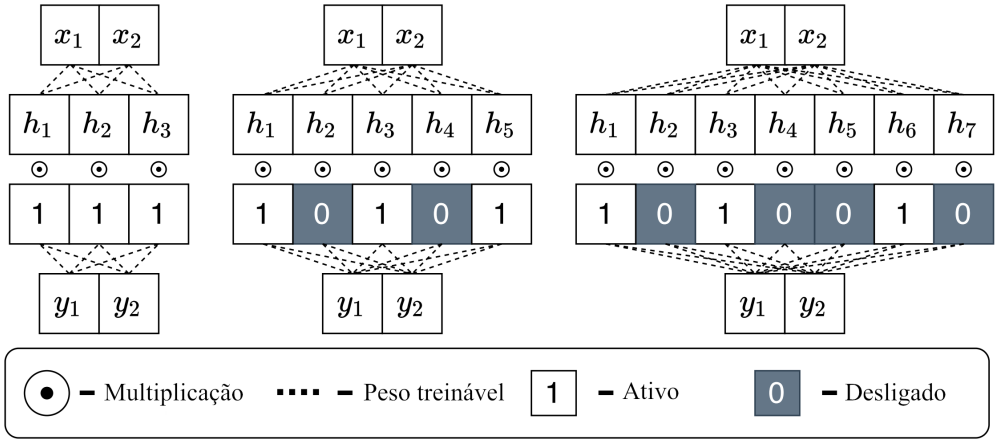


Figura 3.5: Exemplo do Masksembles em uma MLP com diferentes valores de **S** [24]

Na Figura 3.5, mencionada, ao inserir a camada Masksembles logo após uma camada oculta, o tamanho dessa camada é aumentado para $M \times S - D$, onde M é o número de uns em cada máscara, S é uma escala que controla a sobreposição, e D representa o número de neurônios desligados.

Esse aumento de tamanho é fundamental para permitir a aplicação eficaz das máscaras, garantindo que diferentes máscaras possam ser aplicadas a diferentes segmentos da camada. Em outras palavras, ao expandir a camada, é possível que cada máscara opere sobre uma "versão" diferente dela, permitindo múltiplas e relativamente descorrelacionadas previsões da mesma entrada.

As máscaras são geradas durante a inicialização do modelo e, em seguida, essas mesmas máscaras são utilizadas tanto na fase de treinamento quanto na inferência. Durante o processo de treinamento do modelo, para cada amostra, uma dessas máscaras é selecionada aleatoriamente. O modelo então desconsidera as partes especificadas por essa máscara, em uma técnica semelhante ao dropout tradicional. Já na fase de inferência, ou seja, quando o modelo faz previsões, ele é executado diversas vezes, uma para cada

máscara, resultando em várias previsões. Essas previsões múltiplas são, então, utilizadas para estimar a incerteza do modelo e combinadas para gerar uma previsão final.

3.6.1 Detalhes da implementação do Masksembles no modelo base AllRank

A partir da arquitetura original do modelo base AllRank, descrito na Figura 3.2, aplicamos o método Masksembles na camada totalmente conectada (FC) e na camada de *self-attention (MultiHead)*, conforme descrito na Figura 3.6.

$$f(x) = FC(\underbrace{Encoder(Encoder(\cdots Encoder}_{E\times}(FC(x)))))$$

onde

$$Encoder(x) = LayerNorm(z + Masksembles(FC(z), N, S)),$$

$$z = LayerNorm(x + Masksembles(MultiHead(x), N, S))$$

Figura 3.6: Implementação do método Masksembles nas camadas do modelo base AllRank

nos parâmetros do método Masksembles, N é o número de máscaras e S é a escala que controla a sobreposição.

3.7 BatchEnsemble

Em [90] destaca o BatchEnsemble como um método que permite implementar um comitê de sub-redes neurais em um único modelo de aprendizado profundo, explorando a estratégia de treinamento por mini-batches.

Na aprendizagem profunda, um mini-batch é um subconjunto da coleção de dados de treinamento usado para atualizar os parâmetros do modelo em uma única iteração. O BatchEnsemble, por sua vez, aproveita essa estrutura para associar a cada mini-batch um conjunto específico de pesos, efetivamente definindo uma "sub-rede neural" distinta.

Essa estratégia de arquitetura do BatchEnsemble permite a criação de um comitê de sub-redes neurais sem a necessidade de treinar várias redes neurais individualmente, o que teria um custo proibitivo, tanto em termos de tempo quanto em recursos computacionais.

Para esclarecer o funcionamento do BatchEnsemble, precisamos entender primeiro como os pesos são tratados. Em uma rede neural padrão, temos uma matriz de pesos compartilhados W. Já no BatchEnsemble, é criada uma estrutura modificando a

matriz W para formar um comitê de sub-redes neurais de tamanho M, em que cada sub-rede neural tenha uma matriz de peso W_i, $1 \leq i \leq M$. Cada matriz de peso W_i é gerada por um "Posto de uma Matriz"[2] F_i. Logo, F_i é uma matriz que pode ser representada como o produto de um vetor coluna r_i por um vetor linha s_i. Mas estes vetores r e s são distintos para cada sub-rede neural e, portanto, temos um conjunto de vetores r e s para cada sub-rede do comitê. A equação pode ser entendida da seguinte forma:

$$\overline{W_i} = W \circ F_i, \text{ onde } F_i = r_i s_i^T \tag{3-2}$$

O vetor r é responsável por aprender os recursos compartilhados entre as diferentes sub-redes do comitê, enquanto o vetor s aprende os recursos específicos de cada membro do comitê. Durante o treinamento, um *mini-batch* é usado para atualizar os vetores r_i e s_i associados à sub-rede i do comitê para um *mini-batch* em particular, enquanto a média dos gradientes de todas as sub-redes do comitê é usada para atualizar a matriz de pesos compartilhada W.

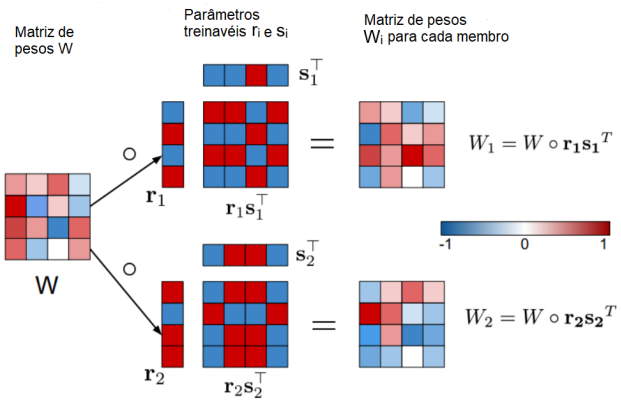


Figura 3.7: Geração dos pesos do conjunto $\overline{W_i}$ para dois membros do ensemble

Como exemplo, na figura 3.8, considere ϕ como o produto entre a matriz de pesos compartilhados W e os parâmetros $\{F_1, F_2 \text{ e } F_3\}$. O resultado desse produto gera matrizes com diferentes conjuntos de pesos $\{\overline{W_1}, \overline{W_2} \text{ e } \overline{W_3}\}$, sendo que cada matriz $\overline{W_i}$ representa uma sub-rede neural do comitê.

Durante a inferência, é gerada uma média das previsões de cada sub-rede do comitê. Considere que o tamanho do lote de teste seja B e que haja M sub-redes neurais no comitê. Para implementação, a entrada do mini-batch é repetida M vezes. Isso resulta em um tamanho efetivo do lote de $B * M$. A "replicação" permite que todas as sub-redes do comitê calculem a saída dos mesmos B pontos de dados de entrada em uma

[2] O Posto de uma matriz (em inglês, rank-one) é uma representação muito mais compacta do que uma matriz completa, pois é representada apenas pelos vetores coluna e linha que a compõem. O objetivo ao utilizar no BatchEnsemble é permitir economizar espaço de memória e facilitar cálculos mais rápidos.

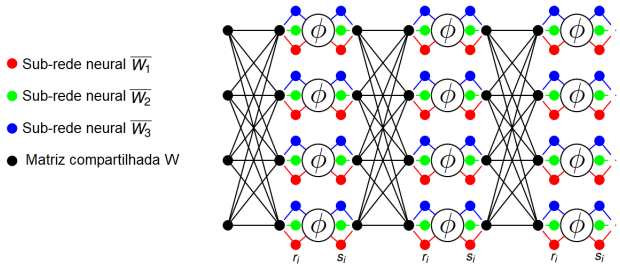


Figura 3.8: Todos os membros do BatchEnsemble compartilham uma mesma matriz de pesos W (cor preta) e cada sub-rede neural tem seus respectivos parâmetros (cor vermelha, verde e azul) [87]

única passagem direta (ou "forward pass"). Na verdade, o que significa que todas as sub-redes neurais estão processando os mesmos dados de entrada em paralelo. O benefício de fazer isso é que elimina a necessidade de calcular a saída de cada sub-rede do comitê sequencialmente.

3.7.1 Implementação do BatchEnsemble no modelo base AllRank

A partir da arquitetura original do modelo base AllRank, descrita na Figura 3.2, aplicamos BatchEnsemble na camada totalmente conectada (FC) e na camada de *self-attention (MultiHead)*, conforme descrito na Figura 3.9, onde M é o número de sub-redes neurais do comitê.

$$f(x) = FC(\underbrace{Encoder(Encoder(\cdots Encoder}_{E\times}(FC(x)))))$$

onde

$$Encoder(x) = LayerNorm(z + BatchEnsemble(FC(z), M)),$$

$$z = LayerNorm(x + BatchEnsemble(MultiHead(x), M))$$

Figura 3.9: Implementação do método BatchEnsemble nas camadas escondidas do modelo base AllRank

CAPÍTULO 4

METODOLOGIA

Neste capítulo, é apresentada a metodologia que aplicamos nos experimentos utilizados para avaliar os métodos de sub-redes neurais para o problema de Aprendizado de Ranqueamento. A seção 4.1 apresenta as coleções de dados utilizadas nos experimentos. A seção 4.2 detalha como foram feitas as escolhas dos valores dos hiper-parâmetros, tanto da rede utilizada como base, como dos métodos de sub-redes. Também é descrita a parametrização realizada no treinamento, validação e testes nos métodos estudados. Por último, apresentamos as métricas usadas para avaliar a diversidade, a efetividade e a eficiência geradas pelo comitê de sub-redes neurais, descritas na seção 4.3.

4.1 Coleção de dados

Para este trabalho foram utilizadas as coleções MSLR-WEB10K[1] [70] e Yahoo! Webscope dataset version 1 and set 2[2] [18]. Estas duas coleções são tradicionalmente usadas por trabalhos acadêmicos de AR para comparar a efetividade de métodos de ranqueamento. Nestas coleções, cada consulta q possui uma lista de tamanho l de documentos. Existe um único rótulo r para cada par $(q, d_j), 1 \leq j \leq l$ que indica o grau de relevância de d_j em relação à consulta q. Assim, cada documento d_j é representado por um vetor de atributos $\overrightarrow{d}_j \in \mathbb{R}^k$, sendo k a dimensão deste vetor. As duas coleções utilizadas são descritas a seguir.

4.1.1 MSLR-WEB10K

A coleção MSLR-WEB10K é composta por consultas escolhidas aleatoriamente nos históricos de consultas da máquina de busca Microsoft Bing[3]. O vetor de atributos que representa cada documento possui dimensão 136. Os atributos se dividem em três tipos, descrevendo o relacionamento do par consulta-documento, descrevendo o documento,

[1] Pode ser encontrada em "https://www.microsoft.com/en-us/research/project/mslr/".
[2] Pode ser encontrada em "https://webscope.sandbox.yahoo.com/catalog.php?datatype=c".
[3] bing.com

e descrevendo a consulta. São exemplos de atributos: a) frequências de um termo da consulta em partes do documento (corpo, título, âncora, url e o documento inteiro), b) o valor do *PageRank* do documento, e c) tamanho do documento. Um rótulo r de um par (q,d) pode assumir um valor no conjunto $\{0,1,2,3,4\}$. O rótulo zero indica que d não é relevante para a consulta q e o rótulo quatro indica relevância máxima.

A coleção se encontra dividida em cinco partes pré-estabelecidas, denominadas *folds*. Essas partes são utilizadas para a validação cruzada. Cada parte possui 10.000 consultas, sendo 6.000 para treino, 2.000 para validação e 2.000 para testes. A tabela 4.1 mostra como a coleção está organizada.

MSLR-WEB10K								
Parte	**Treino**		**Validação**		**Teste**		**Total por parte**	
	q	***d***	***q***	***d***	***q***	***d***	***q***	***d***
1	6000	723412	2000	235259	2000	241521	10000	1200192
2	6000	716683	2000	241521	2000	241988	10000	1200192
3	6000	719111	2000	241988	2000	239093	10000	1200192
4	6000	718768	2000	239093	2000	242331	10000	1200192
5	6000	722602	2000	242331	2000	235259	10000	1200192
Total	**30000**	**3600576**	**10000**	**1200192**	**10000**	**1200192**	**50000**	**6000960**
Número de atributos: 136								

Descrição das colunas: q: número de consultas, d: número de documentos.

Tabela 4.1: Descrição da coleção MSLR-WEB10K

4.1.2 Yahoo!

A versão original da coleção Yahoo! *Learning to Rank Challenge* [18] é composta de dois conjuntos. Porém, neste trabalho, para padronizarmos o processo experimental, seguimos o mesmo procedimento adotado no dataset MSLR-WEB10k, dividindo toda a base em 5 conjuntos (folds). Ressaltamos que a mesma abordagem foi adotada em Sousa et. al [76]. A coleção resultante contém então cinco partes, com 6.330 consultas em cada parte, sendo que 3.798 dessas consultas foram utilizadas para treino, 1.266 para validação e 1.266 para teste.

O vetor de atributos que representa cada documento possui a dimensão de 700. Como ocorre na coleção MSLR-WEB10K, os valores dos rótulos dos pares (consulta, documento), se encontram no conjunto $\{0,1,2,3,4\}$, sendo que o rótulo 0 significa nenhuma relevância e o rótulo 4 significa máxima relevância. Na tabela 4.2, estão descritos os detalhes da coleção.

Yahoo! Learning to Rank Challenge								
Parte	**Treino**		**Validação**		**Teste**		**Total por parte**	
	q	***d***	***q***	***d***	***q***	***d***	***q***	***d***
1	3798	104075	1266	34028	1266	34767	6330	172870
2	3798	103484	1266	35034	1266	34352	6330	172870
3	3798	104066	1266	34756	1266	34048	6330	172870
4	3798	103495	1266	34790	1266	34585	6330	172870
5	3798	103042	1266	34710	1266	35118	6330	172870
Total	**18990**	**518162**	**6330**	**173318**	**6330**	**172870**	**31650**	**864350**
Número de atributos: 700								

Descrição das colunas: q: número de consultas, d: número de documentos.

Tabela 4.2: Descrição da coleção Yahoo! Learning to Rank Challenge

4.1.3 Normalização

Em ambas as coleções, considerando as partições de treino, teste e validação separadamente, foi utilizada a normalização Min-Max, como explicado a seguir. Dada uma consulta q, temos L como a lista de documentos associados a q, e para cada atributo x, os valores são representados por $v_{x_1}, \cdots v_{x_{|L|}}$, em cada documento de L; nesse conjunto, $min(v_{x_1}, \cdots v_{x_{|L|}})$ indica o menor valor e $max(v_{x_1}, \cdots v_{x_{|L|}})$ representa o maior valor. Para cada valor de atributo v_{x_j}, do atributo x correspondente a um documento d_j, $1 \leq j \leq |L|$, substituímos o seu valor por v'_{x_j}, calculado como mostra a Equação 4-1 a seguir:

$$v'_{x_j} = \frac{v_{x_j} - \min(v_{x_1}, \cdots v_{x_{|L|}})}{\max(v_{x_1}, \cdots v_{x_{|L|}}) - \min(v_{x_1}, \cdots v_{x_{|L|}})} \tag{4-1}$$

O efeito positivo dessa normalização é deixar todos os valores de características dos documentos de uma consulta q em uma mesma ordem de grandeza.

4.2 Hiperparâmetros dos modelos

Nesta seção, apresentamos os hiperparâmetros do modelo AllRank e dos métodos de comitê de sub-redes neurais que foram configurados. A arquitetura da rede neural do AllRank foi utilizada como arquitetura base para implementarmos os métodos de comitê avaliados, inclusive seus parâmetros são apresentados na seção 4.2.1. Concomitantemente, são configurados os hiperparâmetros dos comitês de sub-redes neurais que são implementados nessa arquitetura de rede neural do AllRank, apresentada na seção 4.2.2.

Além disso, por ser considerado o estado-da-arte em AR, os resultados do AllRank, apresentados no capítulo 5, foram utilizados como referência para comparação com os métodos de comitê de sub-redes neurais nos experimentos. Este modelo de referência também possui os hiperparâmetros configurados, conforme seção 4.2.1.

4.2.1 Hiperparâmetros do AllRank

O AllRank [68] foi utilizado neste trabalho como um modelo base para aplicação dos métodos de sub-redes neurais investigados neste estudo e apresentados no capítulo 3. Mais especificamente, os métodos de sub-redes neurais foram aplicados nas camadas escondidas da rede neural do AllRank. Os valores de hiperparâmetros do AllRank utilizados em todos os experimentos da pesquisa, em questão, foram os mesmos indicados no artigo que propôs o método [68]. Estes valores são descritos na Tabela 4.3.

Número de neurônios da camada de entrada (d_e)		144
Codificador	Número de blocos do codificador (*encoder*) (n_b)	4
	Número de *attention heads* (n_h)	2
	Número de neurônios das camadas escondidas do bloco codificador (d_h)	512
Número de neurônios da camada de saída (d_s)		4
Função de ativação aplicada na camada de saída (*Activation*)		Sigmoid
Taxa de aprendizado (t_a)		1e-3
Algoritmo otimizador do processo de treinamento (*Optimizer*)		Adam
Algoritmo de ajuste da taxa de aprendizado (*Scheduler*)		StepLR
Função de perda (*Loss Function*)		Ordinal
Tamanho do Batch (t_b)		32
Número de épocas no treinamento (t_{ep})		100

Tabela 4.3: Detalhes dos hiperparâmetros utilizados no modelo AllRank

4.2.2 Hiperparâmetros dos comitês de sub-redes neurais

Cada um dos métodos de comitês apresentados no capítulo 3: MSD, MSD-TT, Masksembles e BatchEnsemble foram aplicados nas camadas escondidas da rede neural do método de ranqueamento AllRank. Para a escolha dos valores dos hiperparâmetros destes métodos foi utilizada a técnica Grid Search. O Grid Search é uma abordagem sistemática que permite explorar um espaço pré-definido de faixas de valores de hiperparâmetros, a fim de identificar a configuração que resulta no melhor desempenho do modelo. Nos experimentos deste trabalho as faixas de valores de cada hiperparâmetro foram obtidas a partir dos artigos que propõem os métodos de comitês utilizados. O hiperparâmetro de cada método, bem como as faixas de valores que foram usadas no Grid Search são mostrados na tabela 4.4.

A determinação do número de componentes dos comitês (coluna 2 da tabela) depende do método de comitê utilizado. No caso dos três primeiros métodos, na tabela, esse número corresponde ao número de máscaras de *dropout*. Já, no caso do BatchEmsemble o número de componentes é passado explicitamente ao método que gera uma matriz de pesos distinta para cada componente a ser criado. A taxa de *dropout* (coluna 3 da tabela 4.4) corresponde à taxa de neurônios desativados pelas máscaras dos métodos que utili-

Método	Número de Sub-redes	Taxa Dropout	Sobreposição (S)	Número de Épocas
MSD	2 a 5	0,3 a 0,6	-	50, 100, 150 e 200
MSD-TT	2 a 5	0,3 a 0,6	-	50, 100, 150 e 200
Masksembles	2 a 5	-	$1,0, 2,0 \cdots, 6,0$	50, 100, 150 e 200
BatchEnsemble	2 a 5	-	-	50, 100, 150 e 200

Tabela 4.4: Hiperpârametros dos métodos de comitês

zam máscaras dinâmicas (MSD e MSD-TT). A escala de sobreposição é a quantidade de neurônios em comum entre duas máscaras no método Maskemsemble, sendo que o valor 1.0 implica em um alto grau de compartilhamento de neurônios entre as sub-redes neurais e 6.0 corresponde a um baixo grau de compartilhamento.

A última coluna da tabela 4.4 é, na verdade, um hiperparâmetro da rede em que os métodos de comitês são aplicados. Porém, eles afetam a qualidade dos comitês treinados, pois os componentes das redes são formados por partes da rede onde atuam. No caso deste trabalho, essa rede é a rede neural do AllRank. Portanto, o número de épocas é também um hiperparâmetro para os métodos de comitês.

Já o GridSearch foi executado somente no conjunto de treino da primeira parte (fold 1) de cada coleção. Para cada combinação dos hiperpârametros, geramos um modelo usando o conjunto de treino da primeira parte da coleção e aferimos o resultado utilizando o conjunto de validação desta mesma parte. Ao final, escolhemos a combinação que gerou melhor NDCG@10[4]. Os valores dos hiperparâmetros que geram os melhores resultados para cada método ao final do GridSearch são apresentados na Tabela 4.5, sendo os valores utilizados para treinamento e teste em ambas as coleções: MSLR-WEB10K e Yahoo.

Método	Número de Sub-redes	Taxa Dropout	Sobreposição (S)	Número de Épocas
MSD	4	0,4	-	150
MSD-TT	4	0,4	-	150
Masksembles	2	-	$6,0$	100
BatchEnsemble	2	-	-	100

Tabela 4.5: Hiperpârametros dos métodos de comitês

4.3 Medidas de Avaliação

Nesta seção, são apresentadas três categorias de medidas para avaliação dos métodos. As subseções seguintes descrevem como os métodos foram avaliados em termos

[4]Optou-se por aplicar o GridSearhc apenas no primeiro fold porque sua aplicação em todos os folds das duas coleções teria custo proibitivo, dado o grande número de combinações dos valores dos hiperparâmetros mostrados na tabela 4.4.

de efetividade, diversidade dos componentes do comitê e o custo computacional.

4.3.1 Efetividade

Como medida de efetividade dos experimentos reportamos às médias das medidas $NDCG@1$, $NDCG@5$ e $NDCG@10$ considerando todos os conjuntos de testes das cinco partes em que foram divididas as coleções.

Para um dado *fold* f computamos cada uma das medidas $NDCG*@1$, $NDCG*@5$ e $NDCG*@10$ para cada consulta de f. Em seguida, computamos as médias aritméticas de cada uma dessas medidas no *fold* f. Por fim, computamos e reportamos, no próximo capítulo, as médias das cinco médias obtidas para cada medida.

4.3.2 Viés e Variância

A análise de viés e variância neste trabalho foram realizadas seguindo a metodologia proposta por Shivaswamy e Chandrashekar [75], que descrevem como fazer o Viés e Variância de forma adequada para problemas de Aprender a Ranquear, descrita na seção 2.6.1 e realizada da seguinte forma.

Antes de seguirmos para a apresentação da metodologia mencionada, é apropriado que retomemos alguns conceitos detalhados na seção 4.1 sobre as coleções de dados, para garantir uma melhor compreensão dos assuntos subsequentes. Neste estudo foram utilizadas as coleções MSLR-WEB10K e Yahoo!. A coleção se encontra dividida em cinco partes pré-estabelecidas, denominadas *folds*. Em cada *fold*, existem três conjuntos para aprendizado: conjunto de treinamento, conjunto de validação e conjunto de teste.

Seguindo a metodologia mencionada, o conjunto de treinamento de um *fold* é então dividido em 5 subconjuntos. Cada subconjunto é formado a partir de 50% de amostras aleatórias do conjunto de treinamento original. Assim, treinamos um mesmo algoritmo a partir desses 5 subconjuntos e, para cada subconjunto, obtivemos um modelo. No final, iremos obter as pontuações de relevância dos documentos gerados pelos 5 modelos distintos processados a partir de um mesmo conjunto de teste deste *fold*.

Este processo se repetiu para todos os *folds*. No final, o erro de viés e variância geral foi computado por uma média aplicada nos resultados do erro de viés e variância de cada *fold*. Logo, ao todo, cada algoritmo passou por 25 treinamentos (5 subconjuntos x 5 *folds*) para computar o erro de viés e variância.

4.3.3 Eficiência

Foi feita uma análise de tempo para verificar o custo para treinamento e teste do comitê de sub-redes neurais. Primeiramente, descrevemos a configuração do *Hardware* do

computador disponibilizado pela LaMCAD/UFG, que possui uma GPU NVIDIA Tesla V100, CPU AMD EPYC 7452, 256 GB de RAM, Sistema operacional CentOS 7.8 e Cuda 11.0.

Antes de iniciar a marcação do tempo no treino e no teste, o modelo é inicializado com os hiperpârametros e a coleção de dados é carregada em memória.

Logo depois, iniciamos o cálculo do tempo de treino do comitê de sub-redes neurais, que é explicado a seguir. Marcamos o tempo de treinamento para cada uma das cinco partes de uma coleção. Antes de iniciar o treinamento, carregamos na memória do computador o conjunto treino de uma parte e iniciamos o modelo com os hiperpârametros definidos na seção 4.2. Logo após, iniciamos o treinamento do modelo. Marcamos o início do tempo na primeira época e marcamos o fim do tempo na última época. Então, obtemos o tempo de treinamento do modelo para essa parte da coleção. Repetimos esse processo para cada parte da coleção. No final, fazemos a média do tempo de treinamento das cinco partes.

Para calcular o tempo de teste, marcamos o tempo de execução nas coleções de testes dos modelos treinados em cada parte da coleção. Carregamos o modelo treinado, a coleção de teste e iniciamos o teste. Marcamos o tempo para processar cada exemplo da coleção pelo modelo. Após o processamento, fazemos a média de tempo desta parte da coleção de teste. Repetimos este processo para cada parte da coleção. Fazemos a média do tempo de teste das cinco partes.

4.3.4 Significância estatística

Em nossa análise, usamos o *t-test* pareado como método para mensurar a significância estatística em tarefas de ranqueamento sugerido por [85] nos resultados de **efetividade** (detalhados nas tabelas 5.1 e 5.2). Também aplicamos o método estatístico aos resultados de **eficiência** (tabelas 5.5 e 5.6). Optamos por um nível de confiança de 95%.

CAPÍTULO 5

RESULTADOS

Neste capítulo apresentamos os resultados dos experimentos realizados buscando responder às perguntas de pesquisa apresentadas no Capítulo 1. Desta forma introduzimos na seção 5.1 a análise de efetividade dos modelos, na seção 5.2 o tempo de execução nas fases de treino e teste, e concluímos o capítulo apresentando na seção 5.3 uma análise geral dos resultados.

5.1 Resultados de Efetividade

Considerando o objetivo de analisar a efetividade dos modelos descritos no capítulo anterior, apresentamos nesta seção possíveis respostas às perguntas de pesquisa QP1 e QP2. Incluindo também uma análise dos resultados sobre a decomposição de viés e variância.

Tanto para responder QP1 como QP2, utilizaremos os resultados descritos nas tabelas 5.1 e 5.2, descrevendo os resultados para as coleções Web10K e YAHOO, respectivamente. Além disso, seguindo os trabalhos apresentados na literatura, consideramos três variações da métrica NDCG: NDCG@1, NDCG@5 e NDCG@10. Para ambas as tabelas o sinal ! indica que há uma diferença com significância estatística entre o modelo avaliado e o nosso principal modelo de referência, AllRank. Para cada tabela, o maior valor da métrica (NDCG@1, NDCG@5 e NDCG@10) é mostrado em negrito. Por fim, o sinal $*$ informa se o maior valor possui uma diferença com significância estatística em relação aos demais modelos.

Resultados experimentais visando responder à QP1

Considerando nossa primeira pergunta de pesquisa proposta para este trabalho – QP1: **Quais das técnicas de comitês de sub-redes neurais aplicadas ao estado da arte conseguem superar em efetividade o estado da arte em AR?** – apresentamos os resultados de efetividade dos modelos avaliados nas tabelas 5.1 e 5.2.

	MSLR-WEB10K		
	NDCG@1	NDCG@5	NDCG@10
AllRank	0,47796	0,47096	0,48871
MSD	0,47842	0,47177	0,48968
MSD-TT	0,48163	0,47337	0,49054
BatchEnsemble	0,48743!	**0,48232!***	**0,50047!***
Masksembles	**0,49041!**	0,48027!	0,49676!

Tabela 5.1: Valores das medidas NDCG@1, NDCG@5 e NDCG@10 na coleção Microsoft LETOR Web10K.

	YAHOO		
	NDCG@1	NDCG@5	NDCG@10
AllRank	0,70178	0,71094	0,75136
MSD	0,73481	0,74252	0,77705
MSD-TT	**0,74418!**	0,74662	0,78062
BatchEnsemble	0,73302!	0,74219!	0,77427!
Masksembles	0,74053!	**0,74665!**	**0,78144!**

Tabela 5.2: Valores das medidas NDCG@1, NDCG@5 e NDCG@10 na coleção de teste da YAHOO.

Como se percebe em ambas as tabelas, os dois métodos de comitês de sub-redes neurais BatchEnsemble e Masksembles superam o método estado da arte AllRank em todas as três medidas e nas duas coleções avaliadas. Considerando a coleção Web10K na tabela 5.1, o BatchEnsemble obteve ganhos em relação ao AllRank de 2,0%, 2,4% e 2,4% em relação às medidas NDCG@1, NDCG@5 e NDCG@10, respectivamente. Por exemplo, BatchEnsemble obtive 0,48232 de NDCG@5 em relação a 0,47096 do método AllRank. O Masksembles obteve ganhos de 2,6%, 2,0% e 1,6% em relação a NDCG@1, NDCG@5 e NDCG@10, respectivamente. Para NDCG@1, Masksembles pontuou 0,49041 em NDCG@1, ultrapassando o valor de 0,47796 do AllRank.

Na coleção YAHOO, seguindo a tabela 5.2, ao comparar com o modelo AllRank, o BatchEnsemble obteve ganhos de 4,5%, 4,3% e 3% em NDCG@1, NDCG@5 e NDCG@10, respectivamente. Já o Masksembles conseguiu ganhos de 5,5%, 5,0% e 4,0% em NDCG@1, NDCG@5 e NDCG@10, respectivamente. Os métodos baseados em Multi-Sample Dropout (MSD e MSD treino e teste) não se mostraram superiores ao AllRank de modo consistente nas duas coleções, embora na coleção YAHOO tenha apresentado empate estatístico com os melhores métodos na métrica NDCG@1.

Respondendo a QP1, nossos experimentos mostram que dos quatro métodos avaliados de comitês de sub-redes neurais, dois mostraram efetividade superior ao AllRank, esses métodos foram: o BatchEnsemble e o Masksembles.

Resultados experimentais visando responder à QP2

Referente a nossa pergunta de pesquisa QP2 – **Existe alguma técnica de comitê de sub-redes neurais que é superior em efetividade a outras?** – as tabelas 5.1 e 5.2 nos auxiliam nesta resposta.

No caso da coleção MSLR-WEB10K, tabela 5.1, o método BatchEnsemble é superior aos demais métodos considerando o critério efetividade para as métricas NDCG@5 e NDCG10, inclusive mantendo uma diferença com significância estatística. Em relação ao Masksembles, o BatchEnsemble apresenta superioridade de 0,4% e 0,7%, para NDCG@5 e NDCG@10, respectivamente. Considerando a métrica NDCG@1, há um empate estatística entre os dois modelos avaliados.

Considerando os experimentos na coleção YAHOO descritos na tabela 5.2, não há um modelo superior nesta coleção. O teste estatístico utilizado não indica que o Masksemble seja superior aos demais métodos em termos de NDCG@5 e NDCG@10. Também o valor de NDCG@1 do método MSD-TT não é estatisticamente superior aos demais métodos para essa medida.

Respondendo a QP2, nossos experimentos mostram que há uma pequena superioridade do BatchEnsemble na coleção MSLR-WEB10K nas medidas NDCG@5 e NDCG@10. Os demais métodos de sub-redes neurais nessa coleção apresentam empate estatístico. Há um empate entre todos os métodos de sub-redes neurais na coleção YAHOO.

Decomposição do erro em viés e variância

Com o objetivo de tentar compreender o desempenho dos métodos avaliados, comparamos o grau de relevância dos documentos considerando a decomposição em somatório dos erros, viés e variância. Para isso, utilizou-se a ordenação par-a-par proposta por Shivaswamy e Chandrashekar [75] e explicada na seção 2.6.1. As tabelas 5.3 e 5.4 apresentam o erro total e sua decomposição em viés e variância para todos os métodos e coleções avaliadas neste trabalho. Considerando a principal tarefa de tentar explicar os ganhos de efetividade dos modelos com comitê de sub-rede, mostramos entre parênteses os valores da variação positiva ou negativa quando comparados ao modelo AllRank.

Considerando a Tabela 5.3, percebemos que o modelo BatchEnsemble – que teve a melhor efetividade para a coleção MSLR-WEB10K – apresenta o melhor compromisso entre viés e variância, quando comparado ao AllRAnk. Ou seja, o modelo Batchensemble apresenta uma redução do viés de 0.68% e um aumento na variância de somente 13,37% quando comparado ao AllRank. Em contrapartida, o Masksembles teve uma taxa de viés reduzida de forma muito similar ao BatchEnsemble, mas com uma variância de 27%, duas vezes maior em relação ao Batchensemble. Na prática o Masksembles reduziu o

	MSLR-WEB10K		
	Erro total	**Viés**	**Variância**
AllRank	**0,30560**	0,2899	**0,0157**
MSD	0,30577 (+0,05%)	0,2870 (-1%)	0,0188 (+19,7%)
MSD-TT	0,49147 (+60,8%)	0,3235 (+11,5%)	0,1679 (+969%)
Batchensemble	0,30573 (+0,04%)	0,2880 (-0,68%)	0,0178 (+13,37%)
Masksembles	0,30569 (+0,02%)	**0,2867** (-1,1%)	0,0190 (+21%)

Tabela 5.3: Resultados de viés e variância com a coleção MSLR-WEB10K

	YAHOO!		
	Erro total	**Viés**	**Variância**
AllRank	0,32472	0,2942	**0,0306**
MSD	0,32477 (+0,01%)	0,2866 (-2,5%)	0,0381 (+24,5%)
MSD-TT	0,47257 (+45,5%)	0,3132 (+6,4%)	0,1594 (+420%)
Batchensemble	0,32473 (+0,003%)	0,2830 (-3,8%)	0,0417 (+36,2%)
Masksembles	**0,32452** (-0,06%)	**0,2818** (-4,2%)	0,0427 (+39,5%)

Tabela 5.4: Resultados de viés e variância com a coleção Yahoo!

viés, mas deixou que o grande aumento na variância diminuísse a precisão na inferência do ranqueamento. Já o modelo BatchEnsemble conseguiu o viés sem penalizar tanto a variância. Como mostra a tabela 5.3, entre os modelos com comitê de sub-rede, o modelo BatchEnsemble teve o menor aumento na taxa de variância. Esse bom compromisso de viés e variância do modelo BatchEnsemble deve justificar a qualidade do mesmo modelo na análise de efetividade.

Considerando a base de dados YAHOO, na tabela 5.4, e analisando os modelos BatchEnsemble e Masksembles, percebemos que o empate estatística de efetividade (descrito na tabela 5.2) se justifica pela similar redução do viés e aumento da variância. Ou seja, comparado ao AllRank, BatchEnsemble e Masksembles apresentam uma variação de viés bem parecida, próximo de 0,4%, e variância com valores próximos também, 36% para BatchEnsemble e 39% para Masksembles. Ou seja, ambos reduziram bastante o viés, e apresentaram moderada alteração na variância.

O MSD-TT também apresenta resultados coerentes com a análise de efetividade. Mais especificamente, o modelo MSD-TT apresentou um aumento de viés em ambas as coleções e ao mesmo um grande aumento na variância comparado ao AllRank, de 969% e de 420% para Web10k e YAHOO, respectivamente. Na prática, o aumento de viés associado ao aumento da variância, não apresentaram alterações significativas nos resultados, consequentemente mantendo as análises de efetividade próximos aos AllRank.

Em resumo, percebemos que os modelos que conseguem melhores resultados na efetividade, também apresentam o melhor compromisso entre redução de viés e moderado aumento na variância, fato ocorrido para os métodos Masksembles e BatchEnsemble em ambas as coleções.

5.2 Resultados de Tempo de Execução

Nesta seção fazemos uma análise em termos de tempo de execução dos modelos, visando responder a nossa terceira pergunta de pesquisa – **QP3 - Qual o custo em termos de tempo de treino e, principalmente, tempo de teste (tempo para gerar o ranque) dos comitês de sub-redes neurais que utilizam o estado da arte em relação ao estado da arte em AR?**.

Considerando nossos experimentos, as tabelas 5.5 e 5.6 contêm uma tentativa de resposta à QP3, com os tempos apresentados em segundos. O tempo de treino reportado corresponde ao tempo médio gasto para o treinamento de um *fold*. Como nosso modelo de referência AllRank é o único que não usa componente de sub-rede em seu processamento, já esperávamos que tivesse a melhor eficiência ou menor custo de execução. Desta forma, destacamos em negrito o custo de execução para o modelo AllRank, por ser o mais rápido, e mostramos para os demais a variação de tempo de processamento adicional em parenteses, sinalizado pelo caractere ↑. O caractere $*$ indica uma diferença com significância estatística ao comparar cada modelo em relação ao AllRank. Por fim, o tempo de teste corresponde a média do tempo para processar todas as consultas de teste.

	MSLR-WEB10K	
	Tempo treino	Tempo teste
AllRank	**2505.96**	$\mathbf{1.38x10^{-4}}$
MSD	2893.43 (↑ 15%) $*$	$1.45x10^{-4}$ (↑ 4.5%)
MSD-TT	2893.43 (↑ 15%) $*$	$2.65x10^{-4}$ (↑ 91%) $*$
Masksembles	3395.55 (↑ 35%) $*$	$2.11x10^{-4}$ (↑ 51%)
BatchEnsemble	4629.41 (↑ 84%) $*$	$2.45x10^{-4}$ (↑ 76%) $*$

Tabela 5.5: Tempo em segundos de treinamento e teste dos modelos na coleção MSLR-WEB10K

	YAHOO	
	Tempo treino	Tempo teste
AllRank	**536.37**	$\mathbf{7.22x10^{-5}}$
MSD	975.96 (↑ 82%) $*$	$7.31x10^{-5}$ (↑ 1.2%) $*$
MSD-TT	975.96 (↑ 82%) $*$	$1.42x10^{-4}$ (↑ 97%) $*$
Masksembles	787.03 (↑ 46%) $*$	$1.00x10^{-4}$ (↑ 38%) $*$
BatchEnsemble	1187.93 (↑ 121%) $*$	$1.26x10^{-4}$ (↑ 74%) $*$

Tabela 5.6: Tempo em segundos de treinamento e teste dos modelos na coleção Yahoo!

Seguindo as tabelas 5.5 e 5.6 e considerando uma lista ordenada crescente de modelos por custo de execução na fase de treinamento, temos a seguinte relação de modelos:

a) **MSLR-Web10K**: 1) AllRank, 2) MSD e MSD-TT, 3) Masksembles e 4) BatchEnsemble.
b) **YAHOO**: 1) AllRank, 2) Masksembles, 3) MSD e MSD-TT e 4) BatchEnsemble.

Considerando o custo de treinamento, percebemos que o modelo BatchEnsemble apresentou os maiores custos em ambas as coleções. Algo esperado, já que de todos os modelos é o que mais aplica uma independência dos parâmetros entre os componentes das sub-redes. Por outro lado, o Masksembles varia entre o segundo mais rápido na coleção YAHOO, e o terceiro mais rápido na coleção WEB10k.

De fato, os tempos de treinamento e de teste são de grande importância para a escolha de um método. Porém, em uma máquina de busca, o tempo de execução do teste é crucial, pois ele compõe o tempo de latência que corresponde ao intervalo entre a submissão da consulta e o instante em que o ranque é apresentado ao usuário. Um método muito efetivo mais com tempo muito lento de resposta pode se tornar inviável em uma máquina de busca. Assim, apresentamos também uma lista ordenada crescente para execução na fase de teste, como mostra a seguinte relação:

a) **MSLR-WEB10K**: 1) AllRank, 2) MSD, 3) Masksembles, 4) MSD-TT e 5) BatchEnsemble.
b) **Yahoo!**: 1) AllRank, 2) MSD, 3) Masksembles, 4) BatchEnsemble e 5) MSD-TT.

Sobre essa perspectiva, verificamos que os modelos que apresentaram os melhores resultados de efetividade, BatchEnsemble e Masksembles, tiveram também maior custo de execução no teste em relação ao AllRank. Porém, entre ambos, o Masksembles foi mais eficiente, apresentando um aumento médio máximo de 51% no teste para a coleção WEB10k e 38% para YAHOO. Portanto, é possível que seu uso seja viável em algumas aplicações de ranqueamento. Por outro lado, o tempo de execução no teste do BatchEnsemble foi consistentemente superior a 70% em relação ao AllRank em ambas as coleções, ou seja, apresentando maior custo de processamento. Ainda assim, seu uso pode ser viável em alguns casos, em que o tempo de latência do processamento das consultas não seja crítico, por exemplo em sistemas de recomendação *offline*.

O MSD tem o melhor tempo de execução no teste para ambas as coleções, porém sua efetividade, como comentado na seção anterior, não é superior ao AllRank. Sua variante MSD-TT foi o método que apresentou o pior tempo na fase de teste para a coleção YAHOO.

Em resumo, percebemos que os modelos que exploram mais a independência dos comitês de sub-rede tendem a aumentar o custo de processamento, como é o caso do BatchEnsemble. Os demais métodos, como Masksembles MSD-TT e MSD, que que exploram componentes aceitando um maior grau de sobreposição entre os parâmetros, possuem custos mais moderados.

5.3 Análise dos resultados

Como mostrado nas seções anteriores, os ganhos quanto à generalização e consequente melhora em efetividade dos métodos BatchEnsemble e Masksembles quando combinados com abordagem *self-attention* do AllRank são de fato consistentes. Contudo, a explicação precisa para esses ganhos e o comportamento sobre os dados entre esses métodos é difícil de se obter. Mesmo a compreensão do sucesso de comitês de redes neurais para o problema de classificação ainda é uma área ativa de pesquisa [36, 25, 3]. Como exemplo, nestes estudos há justificativas experimentais distintas para o ganho de efetividade dos comitês de redes neurais em relação a uma única rede neural. Mostrando que ainda não se tem uma conclusão única.

Em especial, considerando o problema de ranqueamento e suas especificidades, não se tem conhecimento na literatura de uma investigação correspondente. Logo, explicar porque os comitês de sub-redes neurais (cujos componentes não são modelos independentes) também têm apresentado bons resultados no problema de classificação [45, 90, 24] e no problema de ranqueamento (como mostram os resultados desse trabalho) é uma tarefa ainda mais desafiadora. Esse trabalho não se propõe a se aprofundar nessa tarefa, entretanto, apresenta a seguir algumas ideias que podem ajudar a explicar o comportamento dos métodos nas coleções consideradas, e fomentar análises para possíveis trabalhos futuros.

Os métodos Masksembles e BatchEnsemble são, entre os métodos de comitês de sub-redes, aqui considerados, os que possuem menor compartilhamento de informações entre as sub-redes durante o treino. Além disso, são os únicos métodos que são realmente comitês, no sentido de serem formados por mais de um componente que emite uma saída. Por isso, as sub-redes componentes desses dois métodos parecem aprender visões distintas do dado de treino que acabam permitindo um desempenho superior desses métodos em relação ao AllRank.

No caso do Masksembles [24], as sub-redes são caracterizadas por máscaras fixas de dropout. O Masksembles permite controlar a interseção dos neurônios nas máscaras. com isso, pelo menos parte dos neurônios (e suas ligações e pesos) são distintos entre as sub-redes, fazendo com que cada sub-rede atualize um subconjunto dos pesos de forma distinta, durante o treinamento. Entretanto, o Masksembles também aumenta o número de neurônios nas camadas em que ocorre dropout de alguns neurônios. Portanto, não fica claro se o bom desempenho do método é devido unicamente ao comitê de sub-redes ou se é também devido ao aumento de neurônios nas camadas. Essa verificação não foi feita no trabalho que propôs o Masksembles [24] e é sugerida aqui como investigação futura.

No caso do BathchEnsemble [90], as sub-redes neurais são caracterizadas não por máscaras de dropout, mas por matrizes de pesos distintos que são computadas durante

a fase de treinamento. Essas matrizes são obtidas a partir de multiplicações da matriz de pesos da rede neural (comum a todos os componentes do comitê) com uma matriz F_i resultante da multiplicação de dois vetores, r_i e s_i. Há esse par de vetores para cada sub-rede i do comitê e é esse par que caracteriza uma sub-rede. Portanto, o BatchEnsemble também consegue obter sub-redes com algum nível de aprendizado que é distinto entre elas.

O mesmo não ocorre com o método Multi-sample Dropout. O Multi-Sample Dropout mantém várias sub-redes durante uma época do treinamento, obtidas por máscaras de dropout distintas. Entretanto, tais sub-redes se perdem de uma época para a outra durante a fase de treinamento e um novo conjunto de sub-redes é gerado na época seguinte. Ou seja, ao final do conjunto de épocas não se tem realmente um conjunto de sub-redes, embora esses conjuntos tenham sido usados nas fases de back-propagation de cada época. De fato, as sub-redes obtidas durante as épocas são usadas apenas como perturbações dos pesos da rede neural base. Portanto, o Multi-Sample Dropout funciona como um mecanismo mais sofisticado de dropout para a regularização da rede. Ao final do treinamento, o que se tem é a rede base treinada com mais dropouts e não um conjunto de sub-redes que emitam cada uma uma saída. Nosso experimentos mostram que a regularização provida pelo Multi-Sample Dropout não foi suficiente para melhorar o desempenho do AllRank que utiliza também mecanismos de droput.

O Multi-Sample Dropout Treino e Teste se diferencia do Multi-Sample Dropout apenas por aplicar máscaras de dropout no modelo já treinado no momento do teste. Com isso, uma sub-rede é criada para cada máscara de dropout, embora devido à desativação dos neurônios da rede treinada, cada sub-rede resultante se torna um modelo inferior à rede treinada. Dessa forma, o Multi-Sample Dropout Treino e Teste não consegue superar o AllRank em efetividade de forma consistente.

Em resumo, mesmo não tendo uma explicação clara sobre como atuam os comitês de sub-rede, as avaliações dos modelos AllRank, MSD, MSD-TT, Masksembles e BatchEnsemble realizadas neste capítulo, observamos uma variedade de resultados que proporcionam uma melhor compreensão sobre sua efetividade e a eficiência dos modelos. Os métodos BatchEnsemble e Masksembles obtiveram os melhores resultados considerando a efetividade, inclusive sendo consistentemente superiores ao método AllRank, estado da arte em tarefas de Aprendizado de Ranqueamento. No entanto, considerando o custo de execução, nossos experimentos mostram uma superioridade do Masksembles em relação ao BatchEsemble. Na prática, Masksembles apresenta um melhor compromisso entre efetividade e eficiência.

Tradicionalmente, os comitês são conhecidos por reduzir tanto o viés quanto a variância [39]. Contudo, nossa pesquisa mostra que os comitês de sub-redes neurais tem um comportamento distinto do convencional, que embora esses comitês sejam eficazes na

redução do viés, nossos experimentos indicam que eles tendem a aumentar a variância do modelo. Esse aumento da variância se deve ao aumento da complexidade dos modelos de sub-reedes em relação à rede neural base (rede do AllRank).

Entretanto, em uma análise empírica, podemos dizer que esse aumento na variância não afetou na efetividade dos resultados nos métodos do Masksembles e BatchEnsemble, porém, os tornaram mais sensíveis a variações nas coleções de dados de treinamento devido a sua complexidade. Portanto, para reduzir essa variância, podemos seguir algumas estratégias como aumentar o tamanho da coleção de dados de treinamento e ajustar seus hiperparâmetros, como diminuir o número de épocas de treinamento, alterar a taxa de aprendizado.

O MSD-TT, por sua vez, teve um aumento significativo na variância de 969% na coleção MSLR-WEB10K e 420% no Yahoo, em comparação com o AllRank. Esse aumento se deve ao fato de que, em nossos experimentos, a melhor configuração obtida através da técnica de GridSearch para o MSD-TT, envolve o uso de quatro sub-redes, as quais, durante a fase de inferência, são selecionadas aleatoriamente. Ou seja, elas não correspondem a sub-redes cujas relações entre os pesos dos neurônios foram evoluídas durante o treinamento, como ocorre com o Masksembles. Portanto, no MSD-TT, a aleatoriedade envolvida na geração das sub-redes individuais do comitê é altamente sensível a pequenas mudanças no conjunto de treinamento, o que resulta em alta variância.

CAPÍTULO 6

Conclusão

A área de Aprender a Ranquear (AR) tem crescido em relevância devido à sua aplicabilidade em recuperação de informação em diferentes contextos que têm se tornados cada vez mais populares. Por exemplo, em máquinas de busca de comércio eletrônico, em máquinas de busca de propósito geral (Google, Bing, etc), em processamento de perguntas e em sistemas de recomendação.

Recentemente (em 2020), Pobrotyn et al. [68] propuseram o AllRank, um método de AR em que não somente a função de perda, mas também a função de pontuação dos documentos é treinada de modo a considerar os documentos que coocorrem com um documento na lista de documentos relevantes a uma consulta. O AllRank utiliza o mecanismo de autoatenção (*self-attention*) para modelar a relação inter-documentos na computação do valor de relevância de cada documento. Com isso, o AllRank conseguiu superar o método de AR estado da arte na época de sua publicação, que era o LambdaMart [15].

Este trabalho investigou se o uso de comitês de sub-redes neurais utilizadas tomando a rede neural do AllRank como base poderia melhorar a efetividade do AllRank original. Para tanto, investigou-se três métodos de comitês de sub-redes propostas na literatura, BatchEnsemble [90], Masksembles [24] e Multi-Sample Dropout [45] e uma variação desse último método, o Multi-Sample Dropout Treino e Teste. Utilizou-se duas coleções de consultas comumente usadas na literatura: Web10K e YAHOO!. Os experimentos mostram que as adaptações feitas no AllRank para ser utilizado com esses comitês (ver seções 3.4, 3.5, 3.6 e 3.7) apresentaram consistes melhoras na efetividade, mesmo que reduzindo a eficiência. Mas especificamente, os métodos BatchEnsemble e Masksembles proporcionaram ganhos de efetividade ao AllRank de até 4,5% e 5,0% em NDCG@1, respectivamente.

Os métodos de comitês também foram avaliados em termos de tempo de execução durante o treino e durante a fase de teste. Os experimentos mostram que os dois métodos que superam o AllRank em efetividade apresentam tempo de execução superior ao AllRank. O BatchEmsemble tem tempo de execução pelo menos 84% e 74% superior ao AllRank nas fases treino e teste, respectivamente. O Masksembles apresentou tempos

de execução pelo menos 46% e 51% superior ao AllRank no treino e no teste, respectivamente.

Os dois outros métodos não foram superiores em efetividade ao AllRank. O Multi-Sample Dropout entretanto, tem tempo de execução de no máximo 84% superior ao AllRank, mas o tempo de teste é pouco maior que o do AllRank. O método Multi-Sample Dropout Treino e Teste além de não melhorar a efetividade do AllRank, foi o que apresentou o pior tempo de teste.

Não se conhece um estudo comparativo de métodos de comitês de sub-redes neurais como o apresentado nesse trabalho. Os artigos que propõem os métodos os comparam com as redes onde os métodos são aplicados ou com no máximo outro método de comitê de sub-rede neural. Portanto, nem mesmo para o problema de classificação (problema para o qual os métodos foram propostos) encontraram-se trabalhos que avaliam vários métodos de comitês de sub-redes neurais entre si. Logo, considera-se que essa é uma contribuição deste trabalho que mostra que o uso desses comitês de sub-redes são promissores em AR, como solução intermediária entre o uso de comitês de redes neurais independentes, e uma única rede neural.

6.1 Sugestões de Trabalhos Futuros

Uma extensão deste trabalho que se faz necessária é a experimentação em mais coleções. Infelizmente, o tempo utilizado para adaptar os métodos para o AllRank e executar os experimentos não permitiram a experimentação dos métodos em mais coleções de ranqueamento.

Como trabalhos futuros, sugere-se esforço de pesquisa no sentido de reduzir o tempo de execução, principalmente durante o teste dos dois métodos que foram superiores em efetividade ao AllRank (BatchEnsemble e Masksembles). Esse é um desafio importante porque neste trabalho esses dois métodos de comitês continham apenas duas sub-redes, cada um[1]. Se para outras coleções o número de sub-redes for maior, o tempo de execução possivelmente aumentará ainda mais, podendo inclusive tornar impeditivo o uso de várias sub-redes neurais para a geração de ranque.

Outras sugestões de trabalho futuros seriam estudos comparativos como o apresentado neste trabalho, porém para outras tarefas em recuperação de informação, como classificação de texto e sistemas de recomendação. Não se conhece até o momento trabalhos que tenham investigado o uso de comitês para outra aplicação que não fosse classificação de imagens. Dado o bom desempenho dos comitês de sub-redes neurais já demons-

[1] Chegou-se a esse número de sub-redes pelo processo de *grid-search*

trado na classificação de imagens e, aqui, no problema de ranque, essas novas aplicações de sub-redes podem ser promissoras.

Referências Bibliográficas

[1] AI, Q.; BI, K.; GUO, J.; CROFT, W. B. **Learning a Deep Listwise Context Model for Ranking Refinement**. In: *The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval*, p. 135–144, Ann Arbor MI USA, June 2018. ACM.

[2] AI, Q.; WANG, X.; GOLBANDI, N.; BENDERSKY, M.; NAJORK, M. **Learning groupwise scoring functions using deep neural networks**. *In Proceedings of 1st International Workshop on Deep Matching in Practical Applications*, 11 2018.

[3] ALLEN-ZHU, Z.; LI, Y. **Towards understanding ensemble, knowledge distillation and self-distillation in deep learning**. *CoRR*, abs/2012.09816, 2020.

[4] ALLEN-ZHU, Z.; LI, Y. **Towards understanding ensemble, knowledge distillation and self-distillation in deep learning**. In: *The Eleventh International Conference on Learning Representations*, 2023.

[5] BAEZA-YATES, R.; RIBEIRO-NETO, B. **Recuperação de Informação - 2ed: Conceitos e Tecnologia das Máquinas de Busca**. Bookman Editora, 2013.

[6] BEJANI, M. M.; GHATEE, M. **A systematic review on overfitting control in shallow and deep neural networks**. *Artif. Intell. Rev.*, 54(8):6391–6438, dec 2021.

[7] BELKIN, M.; HSU, D.; MA, S.; MANDAL, S. **Reconciling modern machine-learning practice and the classical bias–variance trade-off**. *Proceedings of the National Academy of Sciences*, 116(32):15849–15854, jul 2019.

[8] BELLMAN, R. **Dynamic Programming**. Dover Publications, 1957.

[9] BLUNDELL, C.; CORNEBISE, J.; KAVUKCUOGLU, K.; WIERSTRA, D. **Weight uncertainty in neural networks**, 2015.

[10] BOTTOU, L. **Stochastic gradient learning in neural networks**. *Proceedings of Neuro-Nimes*, 91, 01 1991.

[11] BREIMAN, L. **Bagging predictors**. *Machine Learning*, 1996.

[12] BREIMAN, L. **Random forests**. *Machine Learning*, 45(1):5–32, 2001.

[13] BRIN, S.; PAGE, L. **The anatomy of a large-scale hypertextual web search engine**. *Comput. Netw. ISDN Syst.*, 30(1-7):107–117, Apr. 1998.

[14] BRUCH, S.; ZOGHI, M.; BENDERSKY, M.; NAJORK, M. **Revisiting approximate metric optimization in the age of deep neural networks**. In: *Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR'19, p. 1241–1244, New York, NY, USA, 2019. Association for Computing Machinery.

[15] BURGES, C. **From ranknet to lambdarank to lambdamart: An overview**. *Learning*, 11, 01 2010.

[16] BURGES, C. J. C.; RAGNO, R.; LE, Q. V. **Learning to rank with nonsmooth cost functions**. In: *Proceedings of the 19th International Conference on Neural Information Processing Systems*, NIPS'06, p. 193–200, Cambridge, MA, USA, 2006. MIT Press.

[17] CAUCHY, A. **Méhode générale pour la résolution de systemes d'équations simultanées**. *In Compte rendy des séances de l'académie des sciences*, p. 536–538, 1847.

[18] CHAPELLE, O.; CHANG, Y. **Yahoo! learning to rank challenge overview**. *Journal of Machine Learning Research - Proceedings Track*, 14:1–24, 2011.

[19] CHEN, T.; GUESTRIN, C. **Xgboost: A scalable tree boosting system**. *In Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, 2016.

[20] CHU, W.; GHAHRAMANI, Z. **Gaussian processes for ordinal regression**. *Journal of Machine Learning Research*, 6:1019–1041, dec 2005.

[21] CHU, W.; KEERTHI, S. S. **New approaches to support vector ordinal regression**. In: *Proceedings of the 22nd International Conference on Machine Learning*, ICML '05, p. 145–152, New York, NY, USA, 2005. Association for Computing Machinery.

[22] COHEN, D.; MITRA, B.; LESOTA, O.; REKABSAZ, N.; EICKHOFF, C. **Not all relevance scores are equal: Efficient uncertainty and calibration modeling for deep retrieval models**. In: *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval*. ACM, jul 2021.

[23] DONG, X.; YU, Z.; CAO, W.; SHI, Y.; MA, Q. **A survey on ensemble learning**. *Frontiers of Computer Science*, 2019.

[24] DURASOV, N. E. A. **Masksembles for uncertainty estimation**. *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2021.

[25] FORT, S.; HU, H.; LAKSHMINARAYANAN, B. **Deep ensembles: A loss landscape perspective**. *arXiv*, 2020.

[26] FREUND, Y.; SCHAPIRE, R. E. **A decision-theoretic generalization of on-line learning and an application to boosting**. *Journal of Computer and System Sciences*, 1997.

[27] FREUND, Y.; IYER, R.; SCHAPIRE, R. E.; SINGER, Y. **An efficient boosting algorithm for combining preferences**. *J. Mach. Learn. Res.*, 4(null):933–969, dec 2003.

[28] FRIEDMAN, J. H. **Greedy function approximation: A gradient boosting machine**. *Annals of Statistics*, 29:1189–1232, 2000.

[29] FUHR, N. **A probability ranking principle for interactive information retrieval**. *Inf. Retr.*, 11:251–265, 06 2008.

[30] GAL, Y.; GHAHRAMANI, Z. **Dropout as a bayesian approximation: Representing model uncertainty in deep learning**. In: Balcan, M. F.; Weinberger, K. Q., editors, *Proceedings of The 33rd International Conference on Machine Learning*, volume 48 de **Proceedings of Machine Learning Research**, p. 1050–1059, New York, New York, USA, 20–22 Jun 2016. PMLR.

[31] GEY, F. C. **Inferring probability of relevance using the method of logistic regression**. In: *Proceedings of the 17th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '94, p. 222–231, Berlin, Heidelberg, 1994. Springer-Verlag.

[32] GOODFELLOW, I.; BENGIO, Y.; COURVILLE, A. **Deep Learning**. MIT Press, 2016. http://www.deeplearningbook.org.

[33] GRAVES, A. **Practical variational inference for neural networks**. In: Shawe-Taylor, J.; Zemel, R.; Bartlett, P.; Pereira, F.; Weinberger, K., editors, *Advances in Neural Information Processing Systems*, volume 24. Curran Associates, Inc., 2011.

[34] GRIFLITHS, A.; LUCKHURST, H. C.; WILLETT., P. **Using interdocument similarity information in document retrieval systems**. *Journal of the American Society for Information Science*, 1986.

[35] GUO, C.; PLEISS, G.; SUN, Y.; WEINBERGER, K. Q. **On calibration of modern neural networks**. In: Precup, D.; Teh, Y. W., editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 de **Proceedings of Machine Learning Research**, p. 1321–1330. PMLR, 06–11 Aug 2017.

[36] GUPTA, N.; SMITH, J.; ADLAM, B.; MARIET, Z. **Ensembling over classifiers: a bias-variance perspective**. *arXiv preprint arXiv:2206.10566*, 2022.

[37] HANSEN, L. K.; SALAMON, P. **Neural network ensembles**. *IEEE Trans. Pattern Anal. Mach. Intell.*, 12(10):993–1001, oct 1990.

[38] HARMAN, D. **Overview of the second text retrieval conference (trec-2)**. In: *National Institute of Standards and Technology*, p. 31(3):271–289. Information Processing & Management, 1995.

[39] HASTIE, T.; TIBSHIRANI, R.; FRIEDMAN, J. **The Elements of Statistical Learning**. Springer Series in Statistics. Springer New York Inc., New York, NY, USA, 2001.

[40] HAVASI, M.; JENATTON, R.; FORT, S.; LIU, J. Z.; SNOEK, J.; LAKSHMINARAYANAN, B.; DAI, A. M.; TRAN, D. **Training independent subnetworks for robust prediction**. In: *International Conference on Learning Representations*, 2021.

[41] HAYKIN, S. **Neural Networks: A Comprehensive Foundation**. Prentice Hall, 1999.

[42] HAYKIN, S. S. **Neural networks and learning machines**. Pearson Education, third edition, 2009.

[43] HEUMANN, C.; SCHOMAKER, M. **Introduction to Statistics and Data Analysis: With Exercises, Solutions and Applications in R**. Springer International Publishing, 2017.

[44] HUANG, X.; BELONGIE, S. **Arbitrary style transfer in real-time with adaptive instance normalization**. In: *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, Oct 2017.

[45] INOUE, H. **Multi-sample dropout for accelerated training and better generalization**. *CoRR*, abs/1905.09788, 2019.

[46] JÄRVELIN, K.; KEKÄLÄINEN, J. **Cumulated gain-based evaluation of ir techniques**. *ACM Trans. Inf. Syst.*, 20:422–446, 10 2002.

[47] KE, G.; MENG, Q.; FINLEY, T.; WANG, T.; CHEN, W. **Lightgbm: A highly efficient gradient boosting decision tree**. *In Advances in Neural Information Processing Systems*, 2017.

[48] KE, G.; MENG, Q.; FINLEY, T.; WANG, T.; CHEN, W.; MA, W.; YE, Q.; LIU, T.-Y. **Lightgbm: A highly efficient gradient boosting decision tree**. In: Guyon, I.; Luxburg, U. V.; Bengio, S.; Wallach, H.; Fergus, R.; Vishwanathan, S.; Garnett, R., editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

[49] KINGMA, D.; BA, J. **Adam: A method for stochastic optimization**. *International Conference on Learning Representations*, 12 2014.

[50] KRAMER, S.; WIDMER, G.; PFAHRINGER, B.; DE GROEVE, M. **Prediction of ordinal classes using regression trees**. *Fundam. Inf.*, 47(1–2):1–13, jan 2001.

[51] KROGH, A.; VEDELSBY, J. **Neural network ensembles, cross validation, and active learning**. In: *Advances in Neural Information Processing Systems*, p. 231–238. MIT Press, 1995.

[52] LAKSHMINARAYANAN, B.; PRITZEL, A.; BLUNDELL, C. **Simple and scalable predictive uncertainty estimation using deep ensembles**. *Advances in Neural Information Processing Systems*, 2017.

[53] LECUN, Y.; BENGIO, Y.; HINTON, G. **Deep learning**. *nature*, 521(7553):436, 2015.

[54] LI, H. **Learning to rank**. In: *Encyclopedia of Machine Learning and Data Mining*, p. 1–6. Springer US, Boston, MA, 2016.

[55] LIU, T.-Y. **Learning to Rank for Information Retrieval**. Springer Berlin Heidelberg, 2011.

[56] MANNING, C. D.; RAGHAVAN, P.; SCHÜTZE, H. **Introduction to Information Retrieval**. Cambridge University Press, USA, 2008.

[57] MARON, M. E.; KUHNS, J. L. **On relevance, probabilistic indexing and information retrieval.** *J. ACM*, 7(3):216–244, 1960.

[58] MINKA, T. **Selection bias in the letor datasets**. In: *Microsoft LETOR*, August 2008.

[59] NOH, H.; YOU, T.; MUN, J.; HAN, B. **Regularizing deep neural networks by noise: Its interpretation and optimization**. In: Guyon, I.; von Luxburg, U.; Bengio, S.; Wallach, H. M.; Fergus, R.; Vishwanathan, S. V. N.; Garnett, R., editors, *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, p. 5109–5118, 2017.

[60] OPITZ, D.; MACLIN, R. **Popular ensemble methods: An empirical study**. *J. Artif. Int. Res.*, 11(1):169–198, jul 1999.

[61] PANG, L.; XU, J.; AI, Q.; LAN, Y.; CHENG, X.; WEN, J. **SetRank: Learning a Permutation-Invariant Ranking Model for Information Retrieval**. In: *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*. ACM, July 2020.

[62] PASUMARTHI, R. K.; WANG, X.; BENDERSKY, M.; NAJORK, M. **Self-attentive document interaction networks for permutation equivariant ranking**. *CoRR*, abs/1910.09676, 2019.

[63] PASUMARTHI, R. K.; ZHUANG, H.; WANG, X.; BENDERSKY, M.; NAJORK, M. **Permutation equivariant document interaction network for neural learning to rank**. In: *Proceedings of the 2020 ACM SIGIR on International Conference on Theory of Information Retrieval*, ICTIR '20, p. 145–148, New York, NY, USA, 2020. Association for Computing Machinery.

[64] PEARCE, T.; BRINTRUP, A.; ZHU, J. **Understanding softmax confidence and uncertainty**. *CoRR*, abs/2106.04972, 2021.

[65] PEI, C.; OU, W.; PEI, D.; ZHANG, Y.; ZHANG, Y.; SUN, F.; LIN, X.; SUN, H.; WU, J.; JIANG, P.; GE, J. **Personalized re-ranking for recommendation**. In: *RecSys '19: Proceedings of the 13th ACM Conference on Recommender Systems*, p. 3–11, 09 2019.

[66] PFANNSCHMIDT, K.; GUPTA, P.; HÜLLERMEIER, E. **Deep architectures for learning context-dependent ranking functions**. *ArXiv*, abs/1803.05796, 2018.

[67] PLATT, J. **Probabilistic outputs for support vector machines and comparison to regularized likelihood methods**. In: *Advances in Large Margin Classifiers*, 2000.

[68] POBROTYN, P.; BARTCZAK, T.; SYNOWIEC, M.; BIALOBRZESKI, R.; BOJAR, J. **Context-aware learning to rank with self-attention**. *CoRR*, abs/2005.10084, 2020.

[69] QIN, T.; LIU, T.; XU, J.; LI, H. **How to make letor more useful and reliable**. In: *LETOR conference*. In Proceedings of SIGIR 2008 Workshop on Learning to Rank for Information Retrieval, 2008.

[70] QIN, T.; LIU, T. **Introducing LETOR 4.0 datasets**. *CoRR*, abs/1306.2597, 2013.

[71] QIN, Z.; YAN, L.; ZHUANG, H.; TAY, Y.; PASUMARTHI, R. K.; WANG, X.; BENDERSKY, M.; NAJORK, M. **Are neural rankers still outperformed by gradient boosted decision trees?** *Published as a conference paper at ICLR*, 2021.

[72] QIN, Z.; YAN, L.; ZHUANG, H.; TAY, Y.; PASUMARTHI, R. K.; WANG, X.; BENDERSKY, M.; NAJORK, M. **Are neural rankers still outperformed by gradient boosted decision trees?** In: *International Conference on Learning Representations (ICLR)*, 2021.

[73] ROBERTSON, S. **The probability ranking principle in ir**. *Journal of Documentation*, 33:294–304, 12 1977.

[74] SALTON, G. **Automatic Information Organization and Retrieval**. McGraw Hill Text, 1968.

[75] SHIVASWAMY, P.; CHANDRASHEKAR, A. **Bias-variance decomposition for ranking**. *Association for Computing Machinery*, p. 472–480, 2021.

[76] SOUSA, D. X.; CANUTO, S.; GONÇALVES, M. A.; ROSA, T. C.; MARTINS, W. S. **Risk-sensitive learning to rank with evolutionary multi-objective feature selection**. *ACM*, p. 1–34, 2019.

[77] SPARCK JONES, K.; WALKER, S.; ROBERTSON, S. **A probabilistic model of information retrieval: development and comparative experiments: Part 1**. *Information Processing & Management*, 36(6):779–808, 2000.

[78] SPEARMAN, C. **The proof and measurement of association between two things**. *American Journal of Psychology*, 15:88–103, 1904.

[79] SRIVASTAVA, N. T. **Dropout: A simple way to prevent neural networks from overfitting**. *Journal of Machine Learning Research*, 2014.

[80] SZEGEDY, C.; LIU, W.; JIA, Y.; SERMANET, P.; REED, S.; ANGUELOV, D.; ERHAN, D.; VANHOUCKE, V.; RABINOVICH, A. **Going deeper with convolutions**, 2014.

[81] TAGAMI, Y.; ONO, S.; YAMAMOTO, K.; TSUKAMOTO, K.; .; TAJIMA, A. **Ctr prediction for contextual advertising: Learning-to-rank approach**. *In Proceedings of the 7th International Workshop on Data Mining for Online Advertising*, 2013.

[82] TAULLI, T. **Introdução à Inteligência Artificial: Uma abordagem não técnica**. Novatec Editora, 2020.

[83] TROTMAN, A. **Learning to rank**. *Information Retrieval*, 8(3):359–381, Jan. 2005.

[84] TURKOGLU, M. O.; BECKER, A.; GÜNDÜZ, H. A.; REZAEI, M.; BISCHL, B.; DAUDT, R. C.; D' ARONCO, S.; WEGNER, J.; SCHINDLER, K. **Film-ensemble: Probabilistic deep learning via feature-wise linear modulation**. In: Koyejo, S.; Mohamed, S.; Agarwal, A.; Belgrave, D.; Cho, K.; Oh, A., editors, *Advances in Neural Information Processing Systems*, volume 35, p. 22229–22242. Curran Associates, Inc., 2022.

[85] URBANO, J.; LIMA, H.; HANJALIC, A. **Statistical significance testing in information retrieval: An empirical analysis of type i, type ii and type iii errors**. In: *Proceedings of the 42Nd International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR'19, p. 505–514, New York, NY, USA, 2019. ACM.

[86] VASWANI, A.; SHAZEER, N.; PARMAR, N.; USZKOREIT, J.; JONES, L.; GOMEZ, A. N.; KAISER, L. U.; POLOSUKHIN, I. **Attention is all you need**. In: Guyon, I.; Luxburg, U. V.; Bengio, S.; Wallach, H.; Fergus, R.; Vishwanathan, S.; Garnett, R., editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

[87] VELIKANOV, M.; KAIL, R.; ANOKHIN, I.; VASHURIN, R.; PANOV, M.; ZAYTSEV, A.; YAROTSKY, D. **Embedded ensembles: Infinite width limit and operating regimes**, 2022.

[88] WEBB, G. I. **Occam's Razor**, p. 1–1. Springer US, Boston, MA, 2016.

[89] WELLING, M.; TEH, Y. W. **Bayesian learning via stochastic gradient langevin dynamics**. *International Conference on Machine Learning*, 2011.

[90] WEN, Y.; TRAN, D.; BA, J. **Batchensemble: An alternative approach to efficient ensemble and lifelong learning**. *conference paper at ICLR*, 2020.

[91] WOLPERT, D. H. **Stacked generalization**. *Neural Networks*, 1992.

[92] YAN, L.; QIN, Z.; WANG, X.; BENDERSKY, M.; NAJORK, M. **Scale calibration of deep ranking models**. *Association for Computing Machinery*, p. 4300–4309, 2022.

[93] Z., Z.; V., G.; M., S. **Ex uno plures: Splitting one model into an ensemble of subnetworks**. *Computer Vision and Pattern Recognition*, 2021.

[94] ZHANG.; MA, Y. **Ensemble machine learning: methods and applications**. Springer, 2012.

[95] ZHOU, Z.-H. **Ensemble methods**. Chapman & Hall/CRC, Philadelphia, PA, Aug. 2011.

[96] ZHUANG, T.; OU, W.; WANG, Z. **Globally optimized mutual influence aware ranking in e-commerce search**. In: *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence*, p. 3725–3731, 07 2018.