UNIVERSIDADE FEDERAL DE GOIÁS
INSTITUTO DE INFORMÁTICA

LUCAS ELIAS CARDOSO ROCHA

Análise da Técnica Deep Forest para o Problema de Aprendizado de Ranqueamento

Goiânia
2022

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

2. Nome completo do autor

Lucas Elias Cardoso Rocha

3. Título do trabalho

Análise da Técnica Deep Forest para o Problema de Aprendizado de Ranqueamento

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 13/05/2022, às 15:09, 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 **LUCAS ELIAS CARDOSO ROCHA**, **Discente**, em 17/05/2022, às 09:57, 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 **2903037** e o código CRC **B17394C1**.

Referência: Processo nº 23070.014021/2022-06 SEI nº 2903037

LUCAS ELIAS CARDOSO ROCHA

Análise da Técnica Deep Forest para o Problema de Aprendizado de Ranqueamento

Dissertação apresentada ao Programa de Pós–Graduação do Instituto de Informática da Universidade Federal de Goiás, como requisito parcial 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. Dr. Thierson Couto Rosa

Co-Orientador: Prof. Dr. Daniel Xavier de Sousa

Goiânia
2022

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

Elias Cardoso Rocha, Lucas
Análise da Técnica Deep Forest para o Problema de Aprendizado de Ranqueamento [manuscrito] / Lucas Elias Cardoso Rocha. - 2022.
LXXIV, 74 f.: il.

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, 2022.
Bibliografia.
Inclui siglas, lista de figuras, lista de tabelas.

1. Recuperação de Informação. 2. Learning to Rank. 3. Métodos Ensemble. 4. Deep Forest. I. Couto Rosa, Thierson, orient. II. Título.

CDU 004

UNIVERSIDADE FEDERAL DE GOIÁS

INSTITUTO DE INFORMÁTICA

ATA DE DEFESA DE DISSERTAÇÃO

Ata nº **08/2022** da sessão de Defesa de Dissertação de **Lucas Elias Cardoso Rocha**, que confere o título de Mestre em Ciência da Computação, na área de concentração em Ciência da Computação.

Aos vinte dias do mês de abril de dois mil e vinte e dois, a partir das catorze horas, via sistema de webconferência da RNP, realizou-se a sessão pública de Defesa de Dissertação intitulada **"Análise da Técnica Deep Forest para o Problema de Aprendizado de Ranqueamento"**. Os trabalhos foram instalados pelo Orientador, Professor Doutor Thierson Couto Rosa (INF/UFG), com a participação dos demais membros da Banca Examinadora: Professor Doutor Daniel Xavier de Sousa (IFG), coorientador; Professor Doutor Leonardo Chaves Dutra da Rocha (UFSJ), membro titular externo; e Professor Doutor Sérgio Daniel Carvalho Canuto (IFG), membro titular externo. A realização da banca ocorreu por meio de videoconferência. 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, aos vinte dias do mês de abril de dois mil e vinte e dois.

TÍTULO SUGERIDO PELA BANCA

Documento assinado eletronicamente por **Daniel Xavier de Sousa**, **Usuário Externo**, em 20/04/2022, 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 **Leonardo Chaves Dutra da Rocha**, **Usuário Externo**, em 20/04/2022, 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 **Thierson Couto Rosa**, **Professor do Magistério Superior**, em 20/04/2022, 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 **LUCAS ELIAS CARDOSO ROCHA**, **Discente**, em 20/04/2022, às 16:05, 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 20/04/2022, às 17:12, 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 **2783507** e o código CRC **6387AB1C**.

Referência: Processo nº 23070.014021/2022-06 SEI nº 2783507

Lucas Elias Cardoso Rocha

Bacharel em Ciência da Computação na Universidade Federal de Goiás - UFG (2019)

.

Dedico esse trabalho a todos que se sentirem beneficiados de algum modo pelo processo de construção ou pela leitura do conteúdo aqui contido.

Agradecimentos

Agradeço primeiramente o apoio, amor e incentivo incondicional da minha família durante toda a pesquisa.

Agradeço à Mariana por ser a maior motivação que eu poderia ter e promover meu bem estar quando nada mais seria capaz.

Agradeço também aos meus amigos que me acompanharam e me auxiliaram durante todo esse período, e espero que continuemos reciprocamente cativos dessa amizade.

Aos professores Thierson Rosa e Daniel Sousa pela orientação no desenvolvimento deste trabalho e pelos conhecimentos transmitidos.

Por fim, agradeço ao CNPq - Conselho Nacional de Desenvolvimento Científico e Tecnológico, Capes - Coordenação de Aperfeiçoamento de Pessoal de Nível Superior, UFG - Universidade Federal de Goiás e INF - Instituto de Informática por viabilizarem o desenvolvimento desse trabalho.

À todos os citados, minha sincera gratidão.

"O aspecto mais triste da vida de hoje é que a ciência ganha em conhecimento mais rapidamente que a sociedade em sabedoria."

Isaac Asimov,
Fundação.

Resumo

Rocha, Lucas. **Análise da Técnica Deep Forest para o Problema de Aprendizado de Ranqueamento**. Goiânia, 2022. 74p. Dissertação de Mestrado. Instituto de Informática, Universidade Federal de Goiás.

Learning to Rank (LeToR) é uma especialização do problema de ranqueamento dentro do campo de estudo de Recuperação de Informação (RI). Em LeToR, algoritmos de aprendizado de máquina são utilizados para produzir uma lista ordenada de objetos. A posição relativa dos objetos nessa lista se dá de acordo com seus graus de relevância ou importância, conforme a aplicação do problema. Uma estratégia presente nos algoritmos do estado-da-arte para tratar LeToR são os métodos *ensemble* baseados em árvores de decisão. LambdaMART exemplifica essa estratégia e mostram bons resultados comparados com outros modelos, incluindo redes neurais profundas, se colocando como estado-da-arte de LeToR. Um modelo *ensemble* da literatura ainda não estudado em LeToR, o Deep Forest, busca realizar aprendizado profundo sem a utilização de redes neurais profundas. Para tal, o Deep Forest aplica um processamento camada a camada e uma transformação de atributos dentro do modelo através do *ensemble* de *Random Forest*. Pelo bom desempenho mostrado pelo Deep Forest em diversas tarefas e observando a boa aplicabilidade de métodos *ensemble* baseados em árvores em Learning to Rank, se mostra coerente estudar a aplicação do Deep Forest sob a perspectiva de LeToR. Tendo esse objetivo geral, o presente trabalho investiga experimentalmente aspectos do Deep Forest como hiperparametrização, possíveis avanços do modelo original, o comportamento do modelo em viés e variância e a comparação com redes neurais profundas, tudo no contexto de LeToR. Essa investigação oferece, além dos resultados da aplicação do Deep Forest em LeToR, uma visão analítica do comportamento de modelos *ensemble* em LeToR e uma análise comparativa dos resultados com as redes neurais profundas.

Palavras–chave

Recuperação de Informação, Learning to Rank, Métodos Ensemble, Deep Forest

Abstract

Rocha, Lucas. **Analysis of Deep Forest Technique for Learning to Rank Task**. Goiânia, 2022. 74p. MSc. Dissertation. Instituto de Informática, Universidade Federal de Goiás.

Learning to Rank (LeToR) is a specialization of the ranking problem within the Information Retrieval (IR) field of study. In LeToR, machine learning algorithms are used to produce an ordered list of objects. The relative position of objects is given according to their degree of relevance or importance, depending on the problem application. A strategy present in state-of-the-art algorithms to handle LeToR is tree-based *ensemble* methods. LambdaMART exemplifies this strategy and shows good results compared to other models, including deep neural networks, placing itself as state-of-the-art in LeToR. A tree-based *ensemble* method not yet studied in LeToR, the Deep Forest, aims to perform deep learning without use deep neural networks. To do so, Deep Forest applies a layer-by-layer processing and an attribute transformation within the model through a *ensemble* of *Random Forest*. Due to the good performance shown by Deep Forest in several tasks and observing the good applicability of tree-based *ensemble* methods in Learning to Rank, it is coherent to study the application of Deep Forest under the LeToR perspective. Having this general objective, the present work experimentally investigates Deep Forest aspects such as hyperparametrization, possible improvements of the original model, the model's behavior by bias and variance and the comparison with deep neural networks, all in the context of LeToR. It is expected that this investigation offers, in addition to the results of the application of Deep Forest in LeToR, an analytical view of the behavior of *ensemble* models in LeToR and a comparative analysis of the results with deep neural networks.

Keywords

Information Retrieval, Learning to Rank, Ensemble Learning, Deep Forest

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Siglas e Abreviaturas

DCG – *Discounted Cumulative Gain*
DF – *Deep Forest*
LeToR – *Learning to Rank*
NDCG – *Normalized Discounted Cumulative Gain*
OOB – *Out-of-bagging*
RF – *Random Forest*
RNP – *Rede Neural Profunda*
RI – *Recuperação de Informação*

CAPÍTULO 1

INTRODUÇÃO

Learning to Rank (LeToR) é uma área de pesquisa dentro do campo de estudo de Recuperação de Informação (RI) que aborda ranqueamento de documentos considerando a relevância destes nas consultas. A tarefa principal em LeToR é aplicar técnicas de aprendizado supervisionado para construção de modelos capazes de gerar permutações de documentos com base no(s) relacionamento(s) aprendido entre os seus atributos e o grau de relevância dos mesmos para uma consulta [38].

Há diversas tarefas em RI nas quais algoritmos de LeToR podem ser considerados parte central das soluções, como, por exemplo, Sistemas de Recomendação (*recommendation systems*) [30] e Sistemas de Pergunta e Resposta (*question answering*) [49]. Contudo, o problema mais comumente abordado com LeToR é recuperação de documentos na Web por meio das máquinas de busca. De fato, fica clara nesta tarefa a importância de um sistema de Recuperação de Informação, visto que ao passar dos anos temos um aumento significativo e contínuo de conteúdo na Internet, o que requer métodos cada vez mais efetivos para recuperar as informações almejadas [38].

1.1 Motivação do Trabalho

Dentre os diversos algoritmos que tratam da tarefa de LeToR, modelos que se apoiam nas estrategias de *ensemble* se configuram no estado-da-arte [50]. Na prática, esses algoritmos combinam múltiplos métodos de aprendizado com a expectativa de obter um resultado preditivo melhor do que seria produzido com um único algoritmo. LambdaMART [67] é o principal representante dos algoritmos da categoria *ensemble* do estado-da-arte de LeToR, apresentando ótimos resultados [31, 33, 58]. São vários os trabalhos que o destacam pela alta efetividade no ranqueamento dos documentos [1, 12, 27, 34] e, mesmo passados 12 anos de sua criação, se configura como o principal algoritmo no estado-da-arte na área de LeToR. Isso fica ainda mais claro quando verificado que o LambdaMART apresenta resultados próximos, em alguns casos mais efetivos, que os algoritmos mais atuais de Redes Neurais Profundas (RNP), como observado em [50].

Como consequência, percebemos que os modelos da categoria ensemble conseguem gerar boas representações dos dados para a tarefa LeToR, algo que nos motiva a explorar outros algoritmos de mesma categoria e com potencial de melhorar ainda mais a qualidade de ranqueamento dos documentos.

Neste sentido, e ainda não explorado no contexto de Learning To Rank, o algoritmo *Deep Forest* (DF) [76] surge como uma latente possibilidade de avançar os resultados de ranqueamento. O algoritmo DF tem mostrado uma certa notoriedade no meio acadêmico, sendo aplicado em várias tarefas de Aprendizado de Máquina [14, 23, 40, 55, 68, 69, 72, 73, 74, 77] e ao mesmo tempo sendo foco de estudos que tentam aprimorar seu funcionamento [18, 21, 37, 59, 60].

O funcionamento principal do Deep Forest está focado em uma estrutura denominada Floresta em Cascata. Nela, camadas subsequentes compostas originalmente por Random Forests realizam o processamento de atributos originais e meta atributos. Os meta atributos são predições realizadas por cada componente de uma camada que são passadas à camada seguinte. O número de camadas é definido durante o treinamento, a partir do ponto em que não há ganho em adicionar novas camadas ao modelo.

Como pode-se observar, o algoritmo do DF se baseia no processamento camada a camada e na transformação de atributos durante o processamento das redes neurais profundas, contudo, utilizando de estratégias *ensemble* em uma arquitetura auto-escalável a nível de treinamento e com uma quantidade reduzida de hiperparâmetros. Sua proposta tenta responder a pergunta “O aprendizado profundo pode ser realizado com módulos não diferenciáveis?". Ou seja, tenta produzir um modelo não neural e sem otimização baseada em gradiente, mas aplicando *ensembles* e permitindo a realização de aprendizado profundo.

1.2 Objetivos

Considerando os bons resultados do Deep Foreest em diversas aplicações, e a aderência dos dados da tarefa de LeToR para representação via modelos *ensemble* baseados em árvores de regressão, nos parece intuitivo a proposta descrita neste trabalho. Ou seja, nosso trabalho apresenta um estudo sobre a aplicação do algoritmo Deep Forest em LeToR, ampliamos assim a pergunta originalmente proposta para do DF para: “O aprendizado profundo pode ser realizado com módulos não diferenciáveis capaz de melhorar a efetividade nas tarefas de LeToR?". Essa pergunta ganha ainda mais força ao considerarmos a afirmação em Zhen Qin et al. [50], em que modelos *ensemble* não neurais podem superar modelos neurais em LeToR.

O estudo da pesquisa, então, é guiado pela seguinte pergunta principal **P**:

P O aprendizado profundo com módulos não diferenciáveis, seguindo a estratégia Deep Forest, é capaz de melhorar a efetividade nas tarefas de LeToR?

Deep Forest foi proposto como uma alternativa à aprendizagem profunda (*Deep Learning*) realizada principalmente por redes neurais profundas. Assim, além dos resultados do DF para o problema de LeToR, a pesquisa busca validar se o aprendizado profundo pode ser realizado com módulos não diferenciáveis nas tarefas de LeToR, como colocado pela pergunta de pesquisa principal **P**.

Pretende-se responder a essa pergunta principal através de perguntas de pesquisa secundárias que buscam encontrar uma arquitetura satisfatória para o DF, analisar o comportamento do modelo sobre o problema LeToR e comparar os resultados com RNP. As perguntas secundárias são:

S1 Qual a influência dos hiperparâmetros do Deep Forest em LeToR?
S2 Sendo Deep Forest um modelo *ensemble* baseado em Random Forest, qual a comparação dos resultados de ambos os modelos em LeToR?
S3 Considerando uma seleção de trabalhos propostos na literatura que visam avançar na efetividade da estrutura do DF, podemos dizer que algum desses se mostram efetivos para a tarefa de LeToR?
S4 Quais seriam os resultados de efetividade ao utilizarmos o LambdaMART como *base learner* do Deep Forest?
S5 Qual o comportamento de viés e variância do modelo Deep Forest para LeToR?
S6 Qual a comparação de resultados entre Deep Forest e Redes Neurais Profundas para LeToR?

As perguntas secundárias se colocam como um meio de minuciar **P** e alcançar as respostas pretendidas. Assim, a pergunta **S1** visa encontrar uma boa configuração de hiperparâmetros do modelo para LeToR, comparada com a configuração padrão do DF colocada por Zhou e Feng [76]. A análise de configuração será pautada em dois hiperparâmetros centrais da estrutura do Deep Forest: número de florestas por camada e número de árvores por floresta. Relembramos que, no trabalho original [76], os autores destacam não ser necessário um alto custo para ajuste dos parâmetros, característica que esperamos avaliar para a tarefa LeToR.

A pergunta de pesquisa **S2** tem por objetivo avaliar a efetividade do DF em relação ao seu *base learner* (Random Forest) processado de forma individual. Considerando que o Deep Forest é um modelo *ensemble*, é esperado encontrar melhores resultados em sua aplicação em comparação com o RF como modelo único. Esse cenário justificaria o uso da estratégia *ensemble* implementada no DF. Poré, verificamos isso em **S2**.

Posteriormente, a pergunta de pesquisa **S3** tem como meta avaliar estratégias encontradas na literatura que buscam avançar o estado original do algoritmo DF, visando

verificar a ocorrência desses avanços em LeToR. Assim, propomos a aplicação em LeToR de cinco estratégias de avanço para o Deep Forest a partir de grupos de trabalhos da literatura, como: propagar os meta atributos por toda a floresta em cascata, e aplicar *boosting* entre as camadas da floresta em cascata. As estratégias passaram por adaptações para se adequarem à tarefa de LeToR e experimentos foram aplicados para comparar com o desempenho do DF em sua estrutura original.

Levantando possíveis alterações no modelo base (ou *base learner*) dos modelos *ensemble*, propomos na pergunta de pesquisa **S4** a utilização do LambdaMART como método base para o Deep Forest, consequentemente avaliando os resultados obtidos dessa estratégia.

Tendo em vista uma análise ainda mais quantitativa dos modelos, a pergunta de pesquisa **S5** intenta analisar o comportamento de viés e variância do DF em LeToR. Essa análise contribui para o entendimento do modelo e para explicar a forma com que as adaptações sugeridas em DF impactam nos resultados obtidos com LeToR, se pela correção da variância ou ajuste do viés, conduzindo assim a maior interpretabilidade do modelo de forma mais analítica.

Por fim, **S6** busca comparar os resultados obtidos até então com os resultados recentemente publicados por redes neurais profundas. A resposta para essa pergunta traz a qualificação do uso de aprendizado profundo com módulos não diferenciáveis, por meio do Deep Forest, para o contexto de Learning to Rank. Outro ponto é revisitar, através desses resultados, o comportamento apontado por Zhen Qin et al. [50] em que modelos *ensemble* não neurais superam modelos neurais em LeToR.

Destacamos que **S6** ganha ainda mais importância, pois se de um lado percebe-se um grande avanço nas tarefas de Aprendizado de Máquina com os modelos de Redes Neurais, por outro, alguns autores ainda questionam a capacidade destes no tratamento dos dados em LeToR. Como exemplo disso, os autores em [50] destacam três desvantagens das redes neurais profundas em LeToR: transformação dos atributos de entrada, arquitetura da rede e escassez de dados. Cada característica, quando analisada sob a perspectiva do Deep Forest, indica uma vantagem teórica da aplicação deste em LeToR.

Em mais destalhes, a primeira característica diz respeito à sensibilidade das redes neurais à escala dos atributos de entrada. Em LeToR são encontrados dados com atributos de diversas escalas o que prejudica o desempenho de modelos neurais. Por outro lado, modelos baseados em árvores são conhecidos por particionar de forma eficaz o espaço de atributos, o que é benéfico para conjuntos de dados majoritariamente formados por atributos numéricos, como em LeToR.

A segunda característica, arquitetura da rede, está relacionada ao uso convencional de redes neurais com camadas completamente conectadas. Essa arquitetura de rede

neural é conhecida por não ser eficiente em gerar interações de atributos de alta-ordem. Por meio de um funcionamento de geração de meta atributos baseado em predição, o DF produz e propaga informações que, em comparação às redes neurais profundas, podem mostrar maior capacidade de carregar interações de alta-ordem dos dados.

Por último, a escassez de dados, trata do uso em LeToR de redes neurais profundas com complexidade reduzida. Essa redução busca evitar superajuste aos dados, visto que as coleções de LeToR em geral são menores que as coleções de outras tarefas. Porém, o sucesso das redes neurais se dá pelo uso de modelos neurais com alta complexidade. O Deep Forest ataca esse problema através da sua arquitetura auto ajustada durante o treinamento do modelo. Assim, o Deep Forest é capaz de auto ajustar a sua complexidade tanto no sentido de aumentar a complexidade de seu *base learner* quanto no sentido de limitar a complexidade do *ensemble* realizado e impedir superajuste aos dados. A pergunta de pesquisa **S6** intenta, ainda, validar os aspectos positivos do Deep Forest em relação às redes neurais profundas de acordo com essas três características levantadas.

Levando em consideração as perguntas de pesquisa, este trata-se de um estudo exploratório onde será investigado aspectos não conhecidos do Deep Forest em LeToR, como possíveis avanços e relação com outros modelos *ensembles* do estado-da-arte. O estudo, guiado pelas perguntas de pesquisa, utilizou de metodologias para experimentação de algoritmos de aprendizagem de máquina e LeToR. Três coleções de tamanhos distintos foram escolhidas para avaliar os modelos (detalhadas no capítulo 5): WEB10K, Yahoo! e MQ2007. Testes específicos de parametrizações precederam os testes de avaliação e uma metodologia específica foi utilizada para a análise de viés e variância.

1.3 Resultados

A seguir, destacam-se algumas conclusões obtidas dos resultados alcançados nesta pesquisa:

- A estratégia ensemble aplicada pelo Deep Forest é efetiva em LeToR, dado o aprimoramento dos resultados apresentados pelo Deep Forest em relação ao Random Forest. O DF superou estatisticamente o RF nas três coleções avaliadas com um ganho chegando a 5,82% em NDCG.
- Propostas de avanço estrutural para o Deep Forest da literatura – realizadas originalmente para tarefas de classificação – não indicam efeitos positivos para tarefas de LeToR. Ponderamos que a aplicação direta do DF da tarefa de classificação para regressão impacta diretamente nas propostas de avanço estrutural avaliadas.
- A utilização do LambdaMART como *base learner* do Deep Forest é eficaz para as tarefas de LeToR. Essa estratégia supera o Deep Forest usando Random Forest

como modelo base, obtendo ganho de até 2,94% em NDCG. Se comparado ao LambdaMART, DF usando LamdbaMART como modelo base obtém um ganho máximo de 2,92% em NDCG.

- Este trabalho realiza uma análise de viés e variância do Deep Forest e os *base learners* aplicados. Nessa análise, podemos observar, entre outros fenômenos, que o Deep Forest apresenta uma redução do erro geral de suas predições de ranqueamento por meio de uma redução significativa da variância em relação ao *base learner* utilizado.

Ao final, os resultados mostram que o Deep Forest é capaz de realizar aprendizado profundo sem a utilização de módulos diferenciáveis em LeToR. Sobretudo, o Deep Forest se coloca como um indicador da possibilidade dessa nova abordagem de aprendizado profundo no contexto também de LeToR.

1.4 Estrutura do Documento

Esse trabalho está organizado da seguinte forma: o capítulo 2 apresenta o referencial teórico para o entendimento sobre LeToR (2.1), métodos *ensemble* (2.2), modelos *ensemble* em LeToR (2.3) e viés-variância (2.4); o capítulo (3) é reservado à explicação do Deep Forest e descrição de trabalhos relacionados ao modelo (3.3); o capítulo 4 descreve as propostas de avanço para o DF; no capítulo 5 é detalhada toda metodologia dos experimentos realizados; posteriormente, no capítulo 6 são apresentados e discutidos os resultados dos testes realizados sobre os modelos estudados; por último, temos o capítulo 6 que traz as conclusões finais do trabalho com suas contribuições e possibilidades de trabalhos futuros

CAPÍTULO 2

FUNDAMENTAÇÃO TEÓRICA

Buscando estudar a aplicação do modelo Deep Forest em Learning to Rank, precisamos, primeiramente, prover um esclarecimento a repeito da tarefa LeToR. Posteriormente, se faz adequado pormenorizar as estratégias *ensemble* utilizadas pelo modelo Deep Forest. Para tal, a seção 2.1 traz um detalhamento de LeToR, com suas abordagens e formas de avaliação; a seção 2.2 discute sobre as técnicas *ensemble*, buscando elucidar as diferentes estratégias e os algoritmos aplicados no Deep Forest (2.2); finalmente, a seção 2.3 discute trabalhos da literatura que aplicam modelos *ensemble* em LeToR.

2.1 Learning to Rank

Dentro do campo de estudo de Recuperação de Informação (RI), *Learning to Rank* (LeToR) é considerado uma especialização do problema de ranqueamento, pois refere-se à aplicação de algoritmos de aprendizagem de máquina para a tarefa de ranqueamento baseado em ordem de relevância. LeToR se mostra num aspecto central em diferentes problemas de RI, como sistemas de recomendação (*recommendation systems*) [30], resposta a perguntas (*question answering*) [49] e sistemas de busca (*search-system*)[32]. Considerando uma estratégia mais didática, usaremos a tarefa de recuperação de documentos em máquina de busca para descrever *Learning to Rank* nesse trabalho.

Uma máquina de busca tem por objetivo responder a uma consulta de um usuário com um conjunto de documentos ordenados por relevância em relação à própria consulta. Como podemos observar na figura 2.1, uma máquina de busca possui dois processos consecutivos de ranqueamento. O primeiro é um ranqueamento base, enquanto o segundo é um ranqueamento que procura aumentar a precisão, obtido por LeToR.

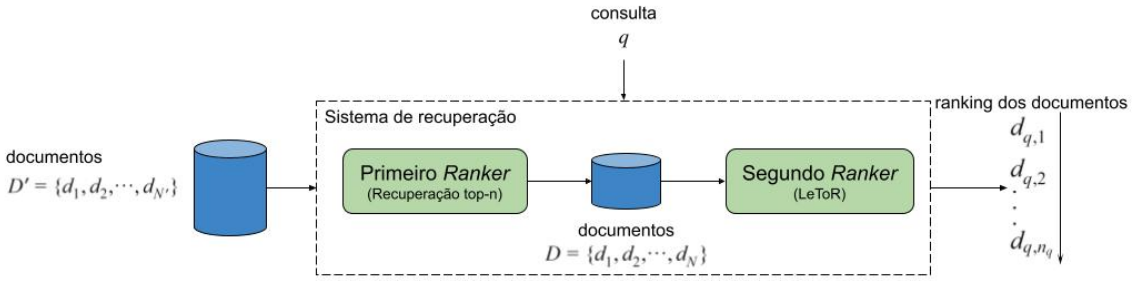

Figura 2.1: Visão global de uma máquina de busca com destaque para os processos de ranqueamento.

Considerando um conjunto D' contendo uma grande quantidade de documentos, nem todos são relevantes para uma consulta qualquer q. Assim, o primeiro ranqueamento ocorre sobre a coleção de documentos $D' = \{d_1, d_2, \ldots, d_{N'}\}$ e visa selecionar um subconjunto D de D' que reúna N documentos possivelmente relevantes à consulta q realizada pelo usuário. O objetivo dessa primeira etapa é maximizar o *recall* do sistema de recuperação [10]. Ou seja, o sistema irá retornar a maioria dos documentos relevantes à consulta, mesmo que alguns documentos não relevantes sejam selecionados. Um algoritmo bastante utilizado para esse primeiro ranqueamento é o BM25 [41] que, de forma geral, tenta recupera e ordenar os documentos em função do número de vezes que os termos da consulta ocorrem nos documentos.

Embora o valor computado pelo BM25 possa ser o único atributo utilizado para a ordenação final nos sistemas de busca, vários outros atributos podem contribuir na ponderação de relevância dos documentos. Entre eles pode-se citar o valor da função PageRank [7] de um documento (neste caso página Web), o número de vezes que um documento foi acessado em consultas anteriores, o número de termos da consulta que aparecem no título do documento, entre outros [11].

Idealmente, uma função única de ranqueamento capaz de utilizar os vários atributos deveria ser aplicada diretamente ao conjunto D' de documentos. Entretanto, muitos desses atributos precisam ser construídos *on-the-fly*, ou seja, no momento em que a consulta está sendo processada (como por exemplo o número de termos da consulta que o ocorrem no título do documento). Portanto, o custo computacional para construir os diversos atributos para todos os documentos de D' durante o processamento da consulta se torna proibitivo. Assim, somente o subconjunto $D = \{d_1, d_2, \ldots, d_N\}$ de documentos selecionados no primeiro ranqueamento será utilizado na geração dos atributos e, consequentemente, no segundo ranqueamento.

O segundo ranqueamento tem como objetivo combinar os valores dos diversos atributos dos documentos, visando maximizar a precisão de relevância do ordenamento final dos documentos de D. Contudo, há relacionamentos complexos entre os atributos que dependem fortemente dos documentos de D, o que torna difícil projetar uma função direta

ou determinística para o segundo ranqueamento que tenha boa precisão para distintas consultas. Com isso, na prática, prefere-se utilizar uma função de ranqueamento derivada automaticamente por meio de alguma técnica de aprendizado de máquina supervisionado.

Consequentemente, a tarefa Learning to Rank – utilizada no segundo ranqueamento – aplica um modelo de ranqueamento $f(q, D)$ supervisionado que tem como objetivo ordenar os documentos em D com base em diversos atributos que descrevem D e a relação de relevâncias com a consulta q, gerando o resultado final $\{d_{q,1}, d_{q,2}, \ldots, d_{q,n_q}\}$. Como mostrado na literatura [15, 70], a aplicação de $f(q, D)$ melhora em média 30% a precisão do ranqueamento final.

A construção de $f(q, D)$ é realizada a partir de um conjunto de consultas previamente avaliadas. Ou seja, cada consulta q_i é composta por uma coleção de documentos $D_i = \{d_{i,1}, d_{i,2}, \ldots, d_{i,n_i}\}$ e seus valores de relevância, que denotaremos por *valores de relevância real*. Os valores de relevância real informam, por exemplo, se o documento é pouco, razoavelmente ou muito relevante à consulta.

Por ser uma tarefa de aprendizado supervisionado, LeToR contêm as fases de treino e de teste. Essas fases são ilustradas na figura 2.2 através dos sistemas de aprendizagem e de ranqueamento, respectivamente. Ou seja, o sistema de aprendizagem refere-se à fase de treinamento do modelo que, na fase de teste, será tratado como um sistema de ranqueamento. Uma vez treinado o modelo, na fase de teste o sistema de ranqueamento é capaz de, dado um conjunto de documentos D relacionados a uma consulta q, criar um ranqueamento (permutação) dos documentos. A avaliação dessa permutação é discutida mais detalhadamente na subseção 2.1.2.

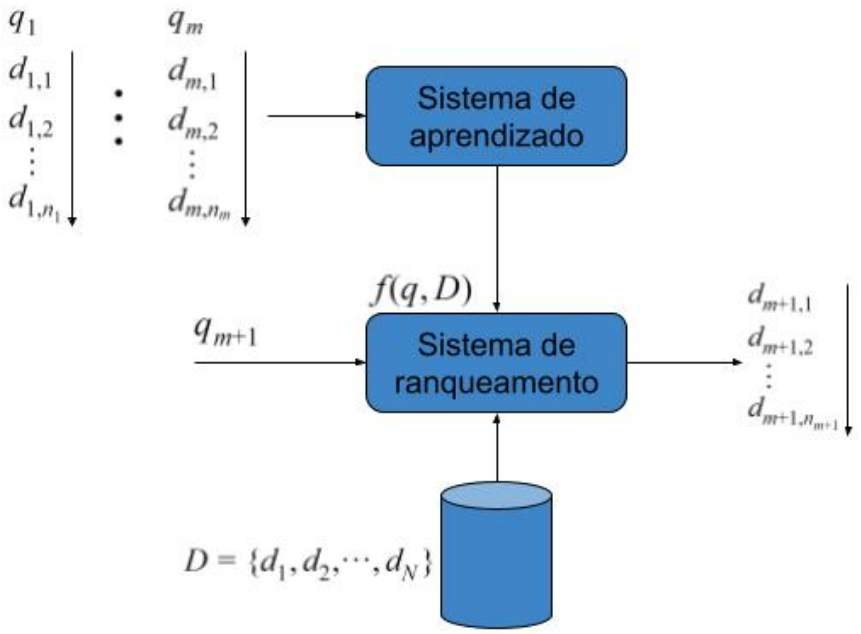

Figura 2.2: LeToR para recuperação de documento. [33]

Há diferentes abordagens para construir a função $f(q, D)$. Essas abordagens se diferenciam pela forma de relacionar os documentos de uma mesma consulta durante o treinamento dos modelos. Os detalhes dessas abordagens serão discutidos a seguir.

2.1.1 Abordagens LeToR

Tie-Yan Liu [38] destaca quatro componentes gerais de algoritmos de aprendizagem de máquina:

- **Representação do Espaço de entrada:** forma de representação dos objetos, geralmente dado por vetores de atributos;
- **Espaço de saída:** formado pelas pontuações alvo da aprendizagem em relação aos objetos do espaço de entrada;
- **Hipótese:** função de mapeamento do espaço de entrada para o espaço de saída, sendo o objetivo da aprendizagem. Assim, a hipótese corresponde à função $f(q, D)$ e utiliza o vetor de atributos dos documentos gerando uma predição de acordo com o espaço de saída;
- **Função de perda:** modo de avaliação da capacidade de generalização da hipótese gerada pelo treinamento.

Através desses quatro componentes é possível classificar a abordagem dos algoritmos de LeToR em três categorias distintas que descrevem diferentes perspectivas com que esses algoritmos podem trabalhar os espaços de entrada e saída, assim como a definição da hipótese e a escolha da função de perda para avaliação do modelo. Como será melhor detalhado à seguir, cada abordagem modela $f(q, D)$ de forma distinta, tratando os documentos do conjunto $D_i = \{d_{i,1}, d_{i,2}, \ldots, d_{i,n_i}\}$ de forma individual (*Pointwise*), por pares (*Pairwise*) ou pelo conjunto como um todo (*Listwise*).

Abordagem *Pointwise*

O espaço de entrada é constituído por um documento representado por um vetor de atributos, enquanto o espaço de saída é o grau de relevância do documento. O vetor de atributos pondera a relação do documento com a consulta, assim, muitas vezes é descrito como vetor par documento-consulta.

Em geral, o grau de relevância é definido por uma pontuação de acordo com uma faixa de valores. Por exemplo, utilizando uma pontuação de relevância dos documentos em $\{0, 1, 2, 3, 4\}$, um documento com pontuação 4 possui máximo grau de relevância, enquanto 0 indicaria a menor relevância.

A hipótese é denominada *função de pontuação* pois calcula a pontuação de relevância do documento de entrada. Por último, a função de perda avalia o erro da

função de pontuação aplicada individualmente a um documento, comparando o valor de relevância predito com o valor de relevância real (*ground truth*).

A abordagem *Pointwise* trata LeToR como um problema de classificação ou regressão. Assim, o modelo treinado tenta predizer a relevância de um documento dado seus atributos. Aplicando o modelo em diferentes documentos, obtêm-se a relevância individual deles e assim podemos gerar uma ordenação final.

Abordagem *Pairwise*

O espaço de entrada é constituído por pares de documentos, ambos representados por vetores de atributos. O espaço de saída é dado pela preferência entre os documentos que formam o par de entrada, ou seja, em geral, é um valor em $\{+1, -1\}$ que indica se um documento é mais relevante que o outro. Dessa forma, considerando $y_{(u,v)}$ o valor de referência para o par de documentos u e v, teríamos:

$$y_{(u,v)} = \begin{cases} +1, & \text{se } u \text{ é mais relevante que } v \\ -1, & \text{senão.} \end{cases}$$

A hipótese é uma função bivariada, pois recebe um par de documentos na representação do espaço de entrada, e computam a ordem relativa dos dois documentos, indicando quem é mais relevante. A função de perda em *Pairwise* compara a ordem do par de documentos indicada pela hipótese com a ordem correta, indicada pelos valores de relevância real dos elementos que formam o par.

Abordagem *Listwise*

O espaço de entrada é dado pelo conjunto de documentos D relacionado à consulta q sendo que cada documento em D é representado pelo seu vetor de atributos, enquanto o espaço de saída é definido por uma lista já ranqueada dos documentos. Por receber um conjunto de documentos, a hipótese é uma função multivariada que produz uma ordenação dos documentos de entrada. A função de perda na abordagem *Listwise* avalia o ranqueamento final produzido sobre todo o conjunto de documentos.

2.1.2 Avaliação em LeToR

Em problemas tradicionais de aprendizagem de máquina (classificação e regressão), a avaliação de resultados é feita verificando se a predição dada para uma instância está correta, ou seja, comparando a predição para a instância com o seu valor de relevância real. Por outro lado, em LeToR, a avaliação deve qualificar a ordenação de documentos produzida. Isto é, indicar o quanto a ordenação gerada pelo modelo se aproxima da melhor

ordenação possível do mesmo conjunto de documentos. Assim, LeToR demanda métricas específicas para avaliar qualidade de ranqueamentos.

A avaliação em LeToR ocorre através da análise da permutação decorrente das pontuações de relevância preditas pelo modelo em relação aos valores de relevância real dos documentos. Para tal, consideremos ranqueamento baseado na ordenação pelos valores de relevância real aqueles em que não há documentos menos relevantes precedendo documentos mais relevantes. Notemos que, na existência de documentos com mesmo grau de relevância em um conjunto relacionado a uma consulta, existem diferentes ranqueamentos baseados na ordenação pelos valores de relevância real. As métricas descritas nessa seção não discriminam a avaliação desses ranqueamentos.

Há diversas métricas para avaliar o desempenho de um modelo para LeToR. Algumas delas são MAP (*Mean Average Precision* – Taxa de Precisão Média), Kendall's Tau e NDCG (*Normalized Discounted Cumulative Gain* – Ganho Cumulativo Descontado Normalizado). Nesta subseção é descrita a métrica NDCG, que é utilizada como métrica principal em diversos trabalhos ([67], [16], [29]) e será utilizada nas avaliações realizadas nos experimentos apresentados nesse trabalho.

Para o entendimento do NDCG, antes é necessário descrever DCG (*Discounted Cumulative Gain* – Ganho Cumulativo Descontado). Sendo q_i uma consulta e D_i o conjunto de documentos associados à essa consulta, consideremos y_i os valores de relevância real e π_i um ranqueamento sobre D_i. O DCG de uma posição k da permutação π_i pode ser definido pela seguinte função [33], onde j é o índice de um documento em D_i:

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

O DCG avalia a qualidade do ranqueamento π_i em uma determinada posição k, onde $\pi_i(j)$ é a posição do documento $d_{i,j}$ em π_i e $y_{i,j}$ é o valor de referência para a relevância do documento $d_{i,j}$. O DCG pode ser interpretado como o ganho cumulativo dado ao recuperar informação a partir da primeira posição do ranqueamento até a posição k. Para cada posição, é dado um ganho de acordo com a relevância do documento em relação à posição. Ou seja, quanto mais relevante o documento e menor for a posição do ranqueamento, maior será o ganho.

Chegamos ao NDCG utilizando $G^{-1}_{max,i}(k)$ para realizar uma normalização do DCG. $G_{max,i}(k)$ é o fator de normalização definido de acordo com o ranqueamento baseado na ordenação pelos valores de relevância real, onde o NDCG para a posição k é 1. Logo, chegamos à seguinte função para o NDCG [33]:

$$NDCG(K) = G^{-1}_{max,i}(k) \cdot \sum_{j:\pi_i(j)\leq k} \frac{2^{y_{i,j}} - 1}{\log_2(1 + \pi_i(j))}.$$

Para efeito interpretativo, NDCG varia no intervalo $[0, 1]$ e indica, assim como o DCG, valores mais altos quando documentos relevantes estão mais ao topo da permutação. Portanto, para ranqueamentos baseados na ordenação pelos valores de relevância real, NDCG atribui 1 à todas as posições. Na avaliação do ranqueamento como um todo é realizada a média dos valores NDCG de todas as posições, sendo essa a aplicação do NDCG nesse trabalho.

2.2 Ensemble

Ensemble é uma estratégia de aprendizado em que é aplicado uma combinação de múltiplos algoritmos de aprendizagem de máquina, denominados *base learners* em uma estrutura única [61]. De forma geral, cada *base learner* produz um resultado preditivo fraco de acordo com uma determinada projeção de instâncias e/ou atributos dos dados. Por fim, os resultados dos diversos *base learners* são combinados com mecanismos de votação ou por um último algoritmo de aprendizagem, produzindo a predição final.

Os modelos *ensemble* visam encontrar diferentes informações geradas pelos *base learners*, com o objetivo de que a integração dessas informações produza uma capacidade preditiva maior do que cada *base learner* individualmente [17]. Para que esse objetivo seja alcançado, é necessário pensar na acurácia e no quão complementares são as informações geradas entre os *base learners*. Ou seja, uma boa acurácia individual dos *base learners* garante que eles gerem resultados assertivos, enquanto a complementabilidade das informações é definida pela diversidade entre eles.

Quanto à diversidade, Zhou e Feng [76] colocam quatro mecanismos principais utilizados individualmente ou em conjunto para incrementar esse aspecto em uma modelo *ensemble*:

- Manipulação de instâncias dos dados, consistindo em gerar diferentes instâncias de treino.
- Manipulação de atributos de entrada, onde é gerado diferentes subespaços de atributos para cada *base learner*.
- Manipulação dos parâmetros de aprendizagem, trabalhando com a utilização de diferentes parametrizações nos *base learners*.
- Manipulação da representação de saída, que diferencia as representações da saída dos *base learners*.

Buscando essa diversificação para os *base learners*, os modelos *ensemble* integram os algoritmos de forma que as informações geradas por cada um se complementem e permita uma performance melhor do que os algoritmos individualmente. A diversidade

dos *base learners* dificulta o superajuste do aprendizado aos dados de treinamento, levando à redução da variância do modelo (variabilidade da previsão do algoritmo para as instâncias de dados [25]). Por outro lado, a redução do viés (diferença entre a previsão média do modelo e o valor correto a ser alcançado [25]) ocorre pelo aumento da complexidade do modelo ao unir diversos algoritmos. Essa busca pela redução do viés e da variância do modelo visa a redução do erro total, ou seja, uma maior qualidade final do modelo em treino e teste [17].

Com base nessas estratégias de diversidade, há diferentes formas de construir um modelo *ensemble*. Utkin [59] coloca uma divisão dos métodos baseados em *ensemble* em 3 categorias de estratégia: *bagging* [5], *stacking* [66] e *boosting* [19]. À seguir temos uma descrição de cada uma delas.

Bagging

Em modelos *ensemble* que utilizam a estratégia *bagging*, para cada *base learner* treinado individualmente, é realizada uma amostragem aleatória de instâncias da coleção de treinamento [51]. A amostragem é efetuada com reposição de instâncias, logo os *base learners* podem compartilhar instâncias de treino, porém, de modo geral, os *base learners* possuem conjuntos de amostras únicos [5]. Dado o conjunto de amostras selecionadas para o treinamento de um *base learner*, denomina-se *out-of-bagging* (OOB) as instâncias não selecionadas do conjunto de treinamento, geralmente utilizadas para avaliação do *base learner*.

A figura 2.3 ilustra a estrutura geral de um modelo *ensemble bagging*. Como podemos observar, para computar a predição de uma instância em fase de teste ou aplicação do modelo *ensemble*, um esquema de voto é utilizado sobre as predições de cada *base learner*.

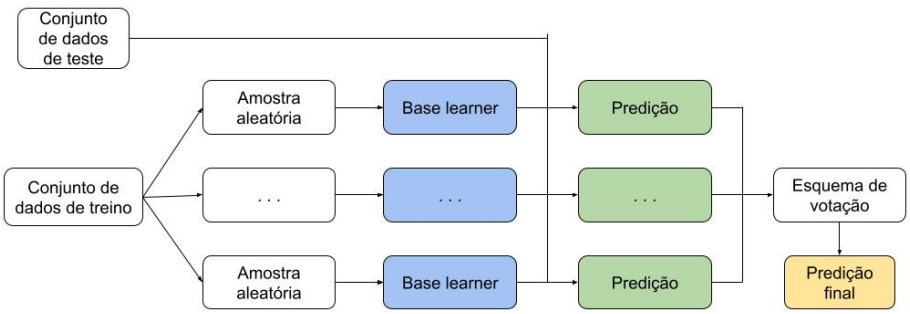

Figura 2.3: Estrutura geral da estratégia *bagging*.

O sistema de amostragem da estratégia *bagging* faz com que cada *base learner* realize suas predições individualmente de acordo com subregiões diferentes de instâncias de treino. Ou seja, os *base learners* são treinados com perspectivas distintas dos dados, o

que os levam a produzir predições através de informações diferentes. Essa característica é responsável pela diversidade entre os *base learners*.

Stacking

Reconhecido na literatura também como *Stacked Generalization*, o método *stacking* realiza o treinamento de diferentes *base learners* separadamente e, em seguida, o treinamento de um metamodelo que irá combinar as predições dos *base learners* [66], como ilustrado na figura 2.4.

O metamodelo é treinado com as predições de cada *base learner* e, de acordo com os valores de relevância real de instâncias de validação, tenta identificar os *base learners* confiáveis para cada instância. Ou seja, o metamodelo tenta realizar o mapeamento entre as predições dos *base learners* e o valor de saída correto [47]. Destaca-se o uso de um conjunto de validação para o metamodelo, diferente do conjunto de treino utilizado para os *base learner*, com o objetivo de evitar o superajuste do metamodelo aos dados de treino (*overfitting*).

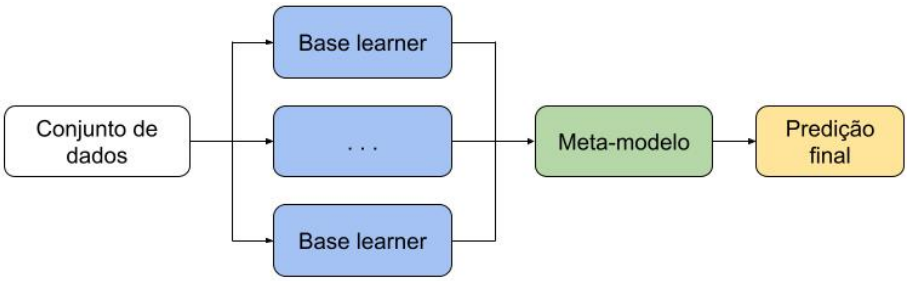

Figura 2.4: Estrutura geral da estratégia *stacking*.

A estratégia *stacking* pode ser utilizada como uma forma de produzir meta-atributos através dos *base learners* para o metamodelo computar a predição final do modelo *ensemble* [59]. Com diferentes *base learners* é produzido diferentes meta-atributos, o que gera informações distintas das instâncias para o metamodelo. Com essa diversidade de informações das instâncias, espera-se que o metamodelo desenvolva uma boa capacidade e generalização.

Boosting

A estratégia *boosting* consiste em treinar *base learners* de forma subsequente, onde as instâncias de treinamento de um *base learner* são estrategicamente selecionadas de forma a compor dados de treinamento mais informativos [19]. O modelo *ensemble* utiliza das predições de cada *base learner* intermediário para ponderar as instâncias de treino de forma que seja selecionado um subconjunto específico de instâncias para o

treinamento do *base learner* seguinte. Esse funcionamento pode ser observado na figura 2.5.

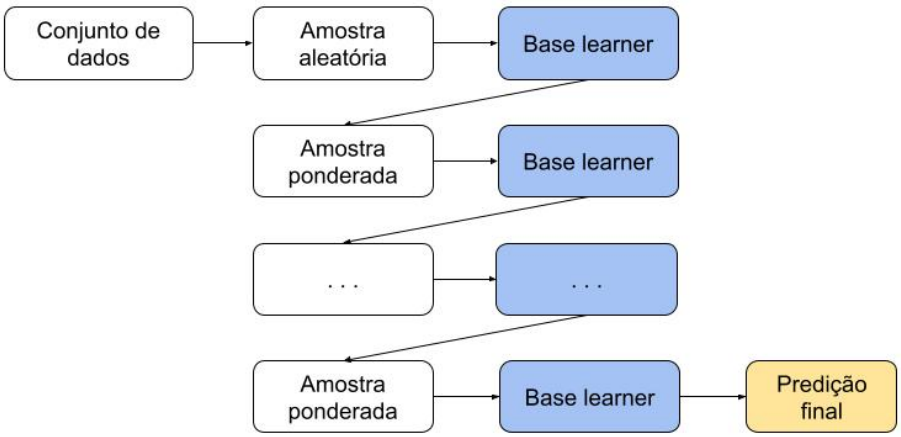

Figura 2.5: Estrutura geral da estratégia *boosting*.

A estratégia *boosting* permite que os *base learners* priorizem instâncias em seu treinamento [47]. Em geral, essa prioridade ocorre sobre instâncias que não foram bem avaliadas nos modelos anteriores, levando cada *base learner* consecutivo a treinar sobre regiões mais difíceis do problema. Para isso, são aplicados pesos mais altos – ou seja, maior prioridade de seleção – para instâncias em que o *base learner* anterior apresentou maior diferença entre o valor predito e o rótolo da instância.

2.2.1 Random Forest

Random Forest (RF) [6] é um modelo *ensemble* do tipo *bagging*. Ele combina árvores de regressão como *base learners*, sendo um dos algoritmos *ensemble* mais difundidos na literatura [64, 69, 76]. RF utiliza uma abordagem como é ilustrado na figura 2.6: para uma dada instância, cada árvore computa uma predição e a pontuação final da floresta é dada pela médias dessas predições.

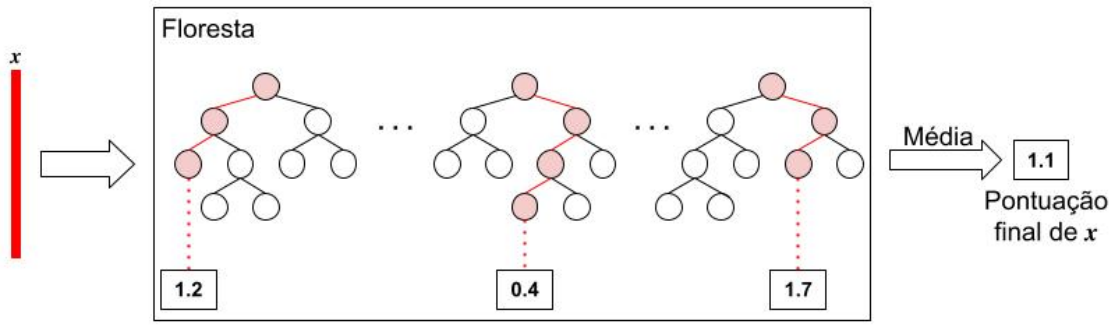

Figura 2.6: Exemplificação de uma regressão pelo RF. Inspirado em [76].

Durante o treinamento, o RF aplica a estratégia de *bagging* na qual constrói diferentes subconjuntos de treinamento para cada árvore, sendo um mecanismo para aperfeiçoar a diversidade dos *base learners*. Essa construção é realizada pela seleção aleatória de instâncias e atributos para o treinamento de cada árvore. É uma estratégia que faz com que cada árvore da floresta tenha acesso a uma sub região do universo de dados de treino e do espaço de atributos, produzindo para o modelo diferentes perspectivas do problema agrupadas pela estratégia *ensemble*.

Além de regressão, o RF também pode ser aplicado para o problema de classificação, onde são utilizadas árvores de decisão como *base learner*. Nesse caso, no lugar de um valor único referente à pontuação da instância, temos um vetor de probabilidade de classe que indica a distribuição da probabilidade da instância pertencer à cada classe do problema. A classificação final é dada por um esquema de votos agrupando os vetores de probabilidade de classe de todas as árvores de decisão. Nessa pesquisa, de acordo com as características de LeToR, será utilizado exclusivamente RF para regressão.

2.3 Modelos Ensemble para LeToR

Em Learning to Rank, diversos *base learners* podem ser utilizados para um modelo ensemble, incluindo algoritmos de regressão e outros modelos de ranqueamento. Jung et al. [29] buscam melhorar a performance do modelo Ranking SVM [26] utilizando-os como *base learner* de um método ensemble. Também encontramos na literatura ensemble de modelos neurais com modelos não neurais [35]. Shuguang Han et al. [24] estuda ensemble de modelos LeToR pointwise, pairwise e listwise, mostrando a possibilidade de gerar ensembles de abordagens LeToR diferentes.

Como podemos observar em BROOF-L2R [16] e LambdaMART [67], ensemble de árvores de regressão é aplicado em modelos para LeToR do estado-da-arte [58]. Esses modelos apresentam a vantagem de que as métricas tradicionais de RI podem ser aplicadas diretamente como função de perda.

BROOF-L2R [16] combina *boosting* e Random Forests tendo como foco a tarefa de LeToR. Como descrito sobre *boosting* na seção 2.2, BROOF-L2R realiza iterações de treinamento de RF onde em cada iteração é definido um subconjunto de treino específico de acordo com o desempenho do modelo até a iteração anterior. O aspecto principal está em como é realizado a seleção do subconjunto de treino, e para isso, são consideradas diferentes funções de perda voltadas para LeToR.

2.3.1 LambdaMART

LambdaMART [67] é um modelo voltado para a tarefa de *Learning to Rank*. Possui dois predecessores – RankNet e LambdaRank – que também são modelos com foco em LeToR. LambdaMART é um modelo *ensemble* de árvores impulsionadas por gradiente (*gradient-boosted trees*) com abordagem *pairwise*, e seu entendimento passa pela compreensão de seus predecessores.

RankNet [8] utiliza modelos com módulos diferenciáveis, tipicamente redes neurais. Sua estratégia é o uso de uma função de perda que calcula o número de inversões em pares de documentos do ranqueamento predito pelo modelo. O objetivo é minimizar o número de inversões e, para isso, é realizado a otimização da função de perda através do gradiente descendente. Ou seja, em uma rede neural, o gradiente descendente reajusta os pesos dos neurônios (módulos diferenciáveis) de forma que as novas pontuações de relevância preditas resultem em um menor número de inversões de documentos na ordenação final.

Os gradientes (λ – lambda) trabalham como indicadores de movimentação dos documentos no ranqueamento de forma a reduzir inversões. Apesar de reduzir o erro *pairwise* (quantidade de inversões), essa estratégia pode não corresponder tão bem com outras métricas como NDCG. LambdaRank [9] aprimora o RankNet escalando os gradientes de acordo com o ganho em NDCG alcançado com a inversão de documentos. Ao analisar cada par de documentos no ranqueamento, é computado o ganho em NDCG resultante da troca de ordem dos documentos no par. Esse ganho pondera os gradientes que, então, passam a ser influenciados por uma métrica de avaliação em LeToR.

LambdaMART combina LambdaRank e MART [20] (árvores de decisão impulsionadas por gradiente). LambdaMART aplica *boosting* em árvores de regressão utilizando gradiente descendente. Ao final da construção de cada árvore, ocorre um ajuste dos valores das folhas e, portanto, das pontuações de relevância que serão preditas pela árvore.

De modo geral, LambdaMART se baseia em três aspectos:

- *Ensemble* baseado em árvores de decisão ajustadas por gradiente descendente;
- Gradiente descendente baseado no erro *pairwise* (quantidade de inversões entre documentos);
- Escalonamento do gradiente de acordo com ganho em NDCG (ou outra métrica de avaliação em LeToR) ao inverter a ordem de documentos.

O uso do gradiente baseado no erro *pairwise* corrige a função de perda no sentido de reduzir o número de inversões. Ponderado por uma métrica de avaliação, como NDCG, o gradiente passa a contribuir para um melhor ranqueamento final em termos de métricas de LeToR. Por último, a aplicação desse gradiente em um *ensemble* de árvores de decisão

realiza a otimização da função de perda por meio do reajuste dos valores das folhas das árvores, levando em consideração todo o ranqueamento predito pela cadeia de árvores anteriores do *boosting*.

2.4 Viés e variância

Em aprendizado de máquina, para discutir resultados preditos por modelos, o erro de uma predição pode ser decomposto sob a perspectiva de viés e variância. Esses são conceitos bem definidos na literatura e são utilizados para decompor o erro total de predição [25]. O entendimento clássico para viés-variância diz respeito à complexidade do modelo que produziu o erro analisado. Na definição dessa complexidade, viés e variância se encontram em um *tradeoff*, ou seja, são características inversamente proporcionais e busca-se o menor valor para ambos. Encontrar uma boa complexidade para um modelo pode ser traduzido pelo encontro de uma combinação de valores reduzidos de viés e variância.

Viés é dado como a diferença média entre o valor predito pelo modelo e o valor correto [25]. Um alto viés revela um alto índice de erro tanto nos dados de treino quanto nos dados de teste (dados não conhecidos pelo modelo). Assim, um alto viés indica um modelo simples, com poucos parâmetros de ajuste ou poucas regras aprendidas. Ou seja, neste caso o modelo apresenta complexidade insuficiente para se ajustar aos dados de treinamento e que, portanto, não apresenta aprendizado. Para resolver esse problema, busca-se um baixo viés com o aumento da complexidade do modelo e, consequentemente, maior aprendizado durante o treinamento e possivelmente menor erro de treinamento.

A variância, por sua vez, é a dispersão dos valores preditos para cada instância dos dados [25]. Assim, modelos com alta variância apresentam diferentes predições para os dados em diferentes treinamentos. Ou seja, o modelo está com alta complexidade, bem adaptado aos dados, o que leva a um super ajuste no treinamento (*overfitting*) e perda da capacidade de generalização sobre dados de teste. Logo, a redução da variância se dá por meio da redução da complexidade do modelo.

Como podemos perceber na Figura, a redução do erro total do modelo passa pela redução do viés e pela redução da variância. Porém, a redução do viés se dá pelo aumento da complexidade do modelo, enquanto a redução da variância se dá pela redução da complexidade do modelo. A figura 2.7, mostra esse *tradeoff* entre viés e variância em relação à complexidade do modelo e suas consequências sobre o erro de treinamento e de teste.

Considerando diferentes arranjos de um mesmo modelo, analisar os vieses e as variâncias apresentadas esclarece a alteração de comportamento e de complexidade que cada arranjo induz no modelo. Nesse sentido, o presente trabalho utiliza de uma

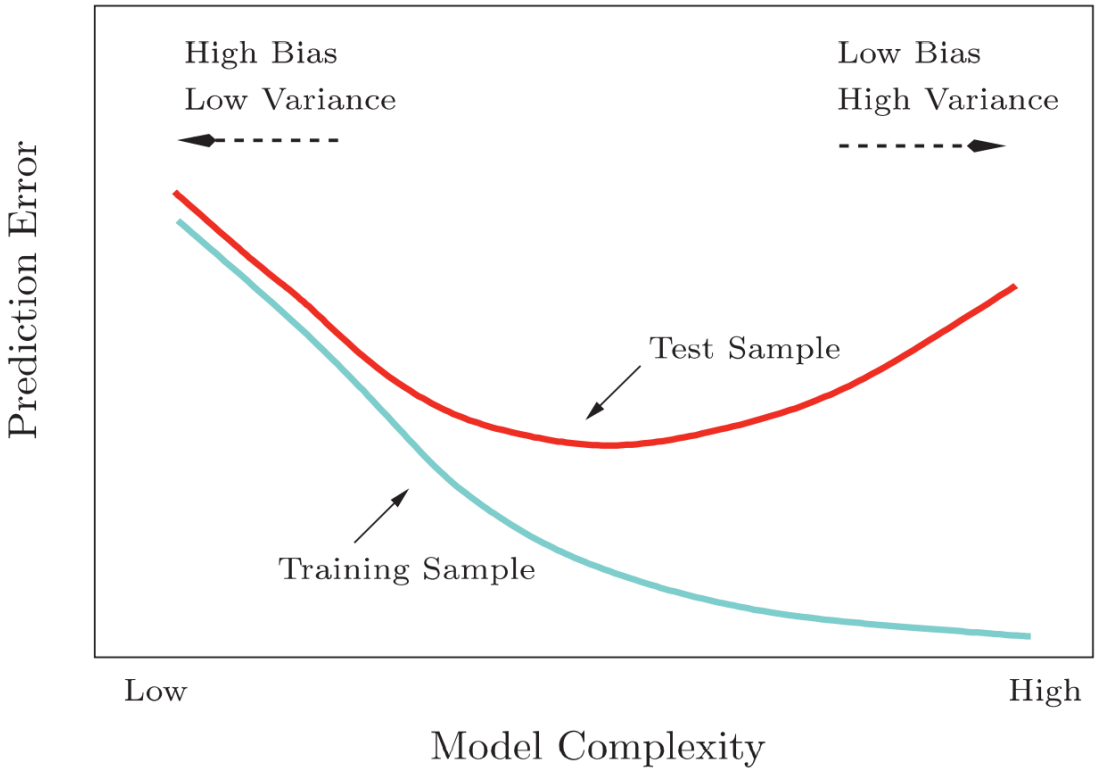

Figura 2.7: Erro de teste e treinamento em função da complexidade do modelo [25].

abordagem de decomposição do erro em viés e variância específica para LeToR. Através dessa abordagem, discutiremos o comportamento do Deep Forest em LeToR sob a perspectiva de dois *base learners*: Random Forest e LambdaMART.

2.4.1 Viés e variância em LeToR

A decomposição do erro total em viés e variância se dá de acordo com a função de perda utilizada como métrica de avaliação para o problema a ser lidado. Como mostrado na seção 2.1.2, temos, em LeToR, funções de perda específicas para modelos de ranqueamento. Dessa forma, é necessário elaborar decomposições de erro total em viés e variância de forma específica para *Learning to Rank*.

Shivaswamy e Chandrashekar [56] propõem uma decomposição do erro total em viés e variância específica para LeToR sob uma perspectiva *pairwise*. Assim, eles definem viés e variância em LeToR da seguinte forma:

- **Viés:** medida de distância entre o ranqueamento baseado nos valores de referência reais (ou rótulos) e o ranqueamento produzido por um modelo.
- **Variância:** frequência com que um modelo produz ranqueamentos diferentes em treinamentos diferentes.

Aplicando essas definições em uma abordagem *pairwise* em *Learning to Rank*, Shivaswamy e Chandrashekar [56] especificam o erro de ranqueamento de um modelo f sobre um conjunto de dados D (documentos relacionados a uma consulta) e os valores de referência y como:

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

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

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

enquanto o termo $var_D(f)$ representa a variância e é dado por:

$$var_D(f) = \mathbf{P}[f(d_i) > f(d_j)]\mathbf{P}[f(d_i) \leq f(d_j)].$$

Em outras palavras, o viés é, para todos os pares de documentos associados $(d_i, d_j) \in D$, a diferença entre a probabilidade de d_i preceder d_j nos ranqueamentos produzidos pelos modelos e no ranqueamento produzido pelos valores de referência. 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. Assim, as probabilidades são – na avaliação de um algoritmo por diferentes treinamentos – a contagem de ocorrências do respectivo evento em relação à todas as ordenações entre o par de documentos analisado.

Dessa forma, é possível computar o viés e a variância de algoritmos para *Learning to Rank* e entender melhor suas complexidades e comportamentos. Esse é um ferramental importante para justificar o melhor desempenho de algoritmos em relação a outros algoritmos em LeToR, como será realizado na seção 6.4.

CAPÍTULO 3

DEEP FOREST

O Deep Forest é a parte central desse trabalho e, portanto, foi separado um capítulo para explicar suas inspirações e funcionamento. Também serão expostos estudos com aprimoramentos e aplicações que trabalhos relacionados já realizaram sobre Deep Forest para diferentes cenários e problemas.

Deep Forest (DF) [76] é um modelo *ensemble* que aborda aprendizagem profunda sem a utilização de redes neurais, tentando responder à pergunta "O aprendizado profundo pode ser realizado com módulos não diferenciáveis?". Para tal, o DF foca nas características das Redes Neurais Profundas, se inspirando nos aspectos vantajosos e trazendo soluções para os aspectos problemáticos.

Uma Rede Neural Profunda (RNP) é uma arquitetura de aprendizagem de máquina onde sua composição de camadas de neurônios conectadas auxiliam na identificação de atributos de alta ordem a partir dos dados brutos [64]. As RNPs estão presentes na grande maioria dos modelos de aprendizagem profunda em diferentes domínios [64]. Porém, elas carregam problemas para esses modelos, como: dificuldade de parametrização, demanda por grande quantidade de dados para treinamento, possibilidade de apresentar complexidade acima do necessário e dificuldade de interpretabilidade do modelo [76].

Para lidar com esses problemas, o DF traz uma arquitetura com a utilização de *ensemble* que possui parametrização reduzida e um auto ajuste na complexidade do modelo em relação ao problema. Inspirado no processamento camada a camada e na transformação de atributos dentro modelo das RNPs, as principais vantagens apresentadas pelo Deep Forest são [76]:

- Deep Forest pode ser aplicado em diversos domínios utilizando a configuração padrão de hiperparâmetros.
- O treinamento do Deep Forest pode ser realizado de forma paralelizada.
- A análise teórica do Deep Forest é mais fácil de ser realizada do que a análise de uma RNP.

O DF é, assim como as RNPs, uma arquitetura de AM com diversas aplicações. Zhou e Feng [76] mostram que o DF tem bons resultados quando comparado com RNPs

em diferentes cenários, como nos problemas de categorização de imagens e análise de sentimento, e com dados de baixa dimensionalidade. Assim, percebemos o poder de adaptabilidade do Deep Forest, sendo aplicável em regressão e classificação.

A arquitetura completa do Deep Forest contém dois mecanismos principais: floresta em cascata (*cascade forest*) e escaneamento multi-granulado (*multi-grained scanning*). A floresta em cascata é o mecanismo principal, onde busca-se realizar o aprendizado com processamento camada a camada e transformação interna de atributos. Por sua vez, o escaneamento multi-granulado é um mecanismo que precede a floresta em cascata e pré-processa os dados originais por meio de predições sobre múltiplas faixas de atributos. Seu objetivo é capturar relações entre atributos, tais quais relações espaciais entre pixeis de uma imagem e relações sequenciais em dados sequenciais.

Zhou e Feng [76] colocam a aplicação do DF sem a utilização do escaneamento multi-granulado quando não existe relações espaciais e sequenciais entre atributos. Algo que pode ser observado nestes trabalhos [40, 74, 22, 60, 64]. Esse é exatamente o cenário de LeToR, pois os atributos dos documentos não possuem relações explicitas entre si. Portanto, no desenvolvimento do presente trabalho, será considerado o Deep Forest apenas com o mecanismo floresta em cascata, desconsiderando o escaneamento multi-granulado.

Adequado à LeToR, esse trabalho traz uma descrição do DF – constituído pela floresta em cascata – tendo como base o problema de regressão, em uma abordagem *pointwise*. O DF também é aplicável para o problema de classificação da mesma forma que será explicado à seguir. A diferença entre as duas abordagens está no número de atributos gerados entre as camadas, que se tratando de classificação é gerado um vetor de probabilidades das classes existentes como novos atributos.

3.1 Floresta em Cascata

Observando o embasamento das redes neurais profundas em uma estrutura camada a camada, o DF utiliza de uma abordagem *ensemble*. Assim, foi estabelecido o mecanismo denominado floresta em cascata conforme será explicado à seguir.

A floresta em cascata é composta por camadas de conjuntos de florestas independentes. Essas florestas são *base learners* baseados em árvores. Originalmente o Deep Forest utiliza Random Forest e florestas de árvores completamente aleatórias (*completely-random tree forests*) [75].

Cada floresta gera um valor de pontuação para cada instância. Como foi mostrado anteriormente no discorrer sobre RF da subseção 2.2.1 e ilustrado na figura 2.6, dado a pontuação resultante de cada árvore, é realizado uma média para definir a pontuação final da instância pela floresta como um todo.

Dentro de uma camada, cada floresta atua de forma independente e suas pontuações resultantes são concatenadas aos atributos originais das instâncias e passados como entrada para as florestas da camada seguinte. Ou seja, os resultados das florestas das camadas internas são utilizados como meta-atributos para as florestas da camada seguinte. Chamaremos esses meta-atributos de atributos aumentados (*augmented features*).

Assim, cada floresta recebe os atributos originais das instâncias mais as pontuações geradas por cada floresta da camada anterior. A primeira camada, por não ter camada antecedente, recebe apenas os atributos originais das instâncias. A última camada define a predição final do modelo através da média das pontuações resultantes de suas florestas. Todo esse funcionamento é esquematizado na figura 3.1.

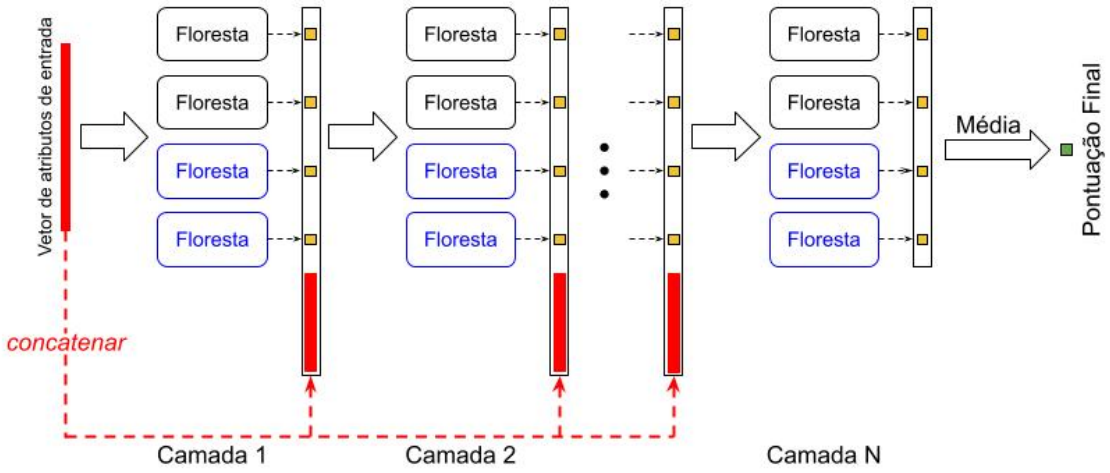

Figura 3.1: Arquitetura da floresta em cascata implementada no modelo Deep Forest. Exemplo mostrando dois RFs (preto) e duas florestas de árvores completamente aleatórias (azul) por camada da cascata. Considerando um problema de regressão, cada floresta gera uma pontuação (amarelo). Assim, cada camada, com exceção da primeira, recebe o vetor de atributos inicial concatenado com a pontuação de cada floresta da camada anterior. A última camada faz a média da pontuação de suas florestas calculando a pontuação final (verde). Inspirado em [76].

Durante o treinamento, cada camada ao ser treinada é avaliada como uma possível camada final definindo o desempenho do modelo até esse nível de complexidade. Dada uma camada C_i qualquer, se após o treinamento das camadas subsequentes $C_{i+1}, C_{i+2}, ..., C_{i+k}$ a avaliação de nenhuma delas superar o desempenho da camada C_i, então o treinamento é finalizado tendo C_i como última camada do modelo. Nesse caso, k é um hiperparâmetro do treinamento.

Dessa forma, a complexidade do modelo, ou seja, o número total de camadas, é dado ao final do treinamento onde é determinada a parada do crescimento. Isso torna o modelo auto escalável, ajustando sua complexidade ao problema que está sendo tratado.

Zhou e Feng [76] definem experimentalmente uma configuração padrão de hiperparâmetros para o DF. Nessa configuração o modelo é constituído por 8 florestas

com 500 árvores em cada camada da Floresta em Cascata. O crescimento das árvores é dado até atingir folhas puras e $k = 3$.

O Deep Forest utiliza abordagens *ensemble* em momentos diferentes. Entre as camadas intermediárias é realizado *stacking*, onde a saída das florestas de uma camada alimentam o treinamento das florestas da camada seguinte. Na camada final, a saída das florestas são combinadas com uma média para definição da predição final do modelo. Por fim, em cada floresta temos *bagging* para o treinamento das árvores, como explicado na seção 2.2.1.

As abordagens *ensemble* permitem diversas modificações no funcionamento do Deep Forest. Como será mostrado na próxima seção, modificações do modelo, como aplicação de *boosting* entre camadas, podem trazer melhorias ao aprendizado. Alguns trabalhos na literatura buscam introduzir essas possíveis melhorias que serão discutidas à seguir.

3.2 Deep Forest para Learning to Rank

A aplicação do Deep Forest em LeToR se dá sob dois aspectos: tarefa de regressão e abordagem *pointwise*. O DF atende ao problema de regressão, como descrito na seção 3.1, por meio da floresta em cascata constituída por florestas (*base leaners*) que abordam regressão. Assim, o DF realizará predições de valores numéricos contínuos que podem ser associados à pontuação de relevância em LeToR. Portanto, a abordagem *pointwise* se encontra na predição da pontuação de relevância individual de cada documento da tarefa de LeToR.

A figura 3.2 esquematiza a aplicação do Deep Forest em LeToR. Cada um dos d_i documentos têm sua pontuação de relevância y_i predita individualmente pelo Deep Forest. Com a pontuação de relevância predita de todos os n documentos é possível realizar o ranqueamento deles, ou seja, ordena-los de acordo a pontuação predita. Dessa maneira, obtemos o ranqueamento inferido pelo modelo.

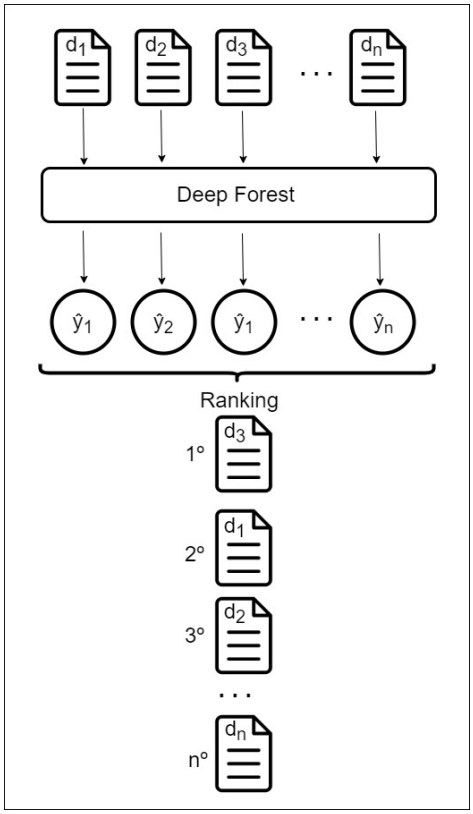

Figura 3.2: Aplicação do Deep Forest em LeToR sob uma abordagem *pointwise*.

3.3 Trabalhos Relacionados

Apesar de ser um modelo relativamente novo, encontra-se diversos trabalhos na literatura propondo diferentes aplicações e modificações para o Deep Forest [63, 21, 71]. Focando nas modificações e possíveis evoluções do DF, organizamos a apresentação desses trabalhos agrupando-os de acordo com o tipo de alteração aplicado no modelo original. Assim, definimos cinco grupos de trabalhos que modificam o modelo DF e um grupo extra de trabalhos que aplicam DF em contextos de classificação/regressão automática. A descrição desses grupos é feita à seguir.

Grupo 1: Modificação da estrutura dos atributos aumentados

Alguns trabalhos alteram a estrutura dos atributos aumentados que são passados de camada à camada da floresta em cascata ([21], [42], [63]). Originalmente, os atributos aumentados gerados por uma camada da floresta em cascata são compostos pela pontuação (no caso de regressão) ou vetor de classes (quando classificação) de todas as florestas. Fu et al. [21] calcula uma única pontuação por camada através de uma média ponderada das pontuações dada por cada floresta, reduzindo a dimensionalidade de atributos em relação ao modelo original. Por outro lado, Miller et al. [42] propõe o aumento

da dimensionalidade de atributos passando a considerar a saída das árvores de cada floresta como atributos aumentados. Miller et al. [42] avaliam a proposta em uma coleção de classificação e mostram ganho de 0,02% de acurácia sobre o Deep Forest.

Exemplificando as propostas anteriormente, consideremos o DF com a seguinte estrutura para um problema com dez atributos originais: quatro florestas por camada e vinte árvores por floresta. No modelo DF original teríamos quatro atributos aumentados por camada e as camadas internas receberiam quatorze atributos (dez originais mais os quatro atributos aumentados) da camada anterior. A modificação proposta por Fu et al. [21] resultaria em atributos aumentados de dimensão (01) um (média das pontuações dada pelas quatro florestas) sendo passado para a próxima camada onze atributos (dez originais mais um atributo aumentado). Por sua vez, na proposta de Miller et al. [42] encontraríamos atributos aumentados de dimensão oitenta (quatro florestas com vinte árvores cada) e passariam de camada à camada noventa atributos (dez originais mais oitenta atributos aumentados).

Grupo 2: Seleção de atributos entre camadas da floresta em cascata

Dada a adição dos atributos aumentados entre as camadas, certos trabalhos selecionam os atributos desse conjunto que serão propagados ([18], [39], [44], [57], [71]). O trabalho de Sun at al. [57] junta os atributos aumentados produzidos por uma camada ao seu conjunto de atributos de entrada, aplica uma seleção com base na importância dos atributos para as florestas e propaga os atributos selecionados para a camada seguinte. Como podemos perceber, essa estratégia pode fazer com que os atributos originais sejam completamente substituídos pelos meta-atributos ao longo da floresta em cascata, e a justificativa dada para a realização dessa seleção é a busca pela eliminação da redundância de atributos. A estratégia aplicada por Sun at al. [57] apresenta ganho de até 2,38% de acurácia para o Random Forest. Porém, não há comparação com o Deep Forest original.

Ni e Kao [44] aplicam uma seleção de atributos sobre os atributos aumentados de uma camada com base no *erro out-of-bagging* (OOB *error*) [43], apontando os seguintes objetivos: reduzir a dimensionalidade de atributos, diminuindo também o consumo de memória e tempo, e destacar os atributos importantes para o aprendizado. Apesar de focar também no consumo de memória e tempo, o trabalho apresenta um ganho médio em acurácia de 1,58% em cinco coleções de classificação em relação ao Deep Forest.

Grupo 3: Filtro de instâncias nas camadas da floresta em cascata

Visto que, para algumas instâncias, as camadas intermediárias da floresta em cascata podem demonstrar uma predição melhor do que a camada final do modelo, certos trabalhos propõem a possibilidade de encerrar o processamento de instâncias em

camadas intermediárias ([60], [45]), tornando qualquer camada do modelo uma camada final para distintas instâncias. Pang et al. [45] define um mecanismo denominado Triagem de Confiança (*Confidence Screening*) que seleciona as instâncias que devem passar para a próxima camada ou terem predição dada pela camada atual. A ideia que embasa essa proposta está vinculada à necessidade de uma instância em requerer um nível a mais de aprendizado. Assim, uma instância que recebe predição confiável em uma camada pode considerar essa como sua predição final, enquanto uma predição com baixa confiança indica necessidade da mais camadas de aprendizado para a instância. Pang et al. [45] mostram que a aplicação da Triagem de Confiança avança os resultados do Deep Forest com ganho em acurácia de até 0,37% em coleções utilizando somente o mecanismo de floresta em cascata.

Grupo 4: Perpetuação dos atributos aumentados

No modelo original do Deep Forest os atributos aumentados produzidos por uma camada são propagados apenas para a camada seguinte. Alguns trabalhos na literatura propõem a propagação permanente dos meta-atributos à partir da camada em que são produzidos até a camada final do modelo ([37], [44], [64]). Wang et al. [64] se inspira nas redes convolucionais densamente conectadas [28], justificando com a possibilidade de que o compartilhamento de atributos entre todas as camadas subsequentes do modelo resulte na correção das falhas das camadas anteriores. Essa proposta apresenta melhores resultados para 14 coleções de classificação com ganho em acurácia alcançando 1,29%. Liu et al. [37] também se baseia nas redes convolucionais densamente conectadas, enquanto Ni et al. [44] utiliza dessa estratégia combinada com a seleção de atributos já descrita anteriormente, alcançado, como já mencionado, uma ganho médio em acurácia de 1,58% em relação ao DF.

Grupo 5: Ponderação das florestas

A proposta desses trabalhos é ponderar os meta-atributos de acordo com uma avaliação das florestas que os produzem ([21], [59], [61], [62], [63]). De forma geral, um peso é calculado para cada floresta que é utilizado para ponderar seus respectivos atributos aumentados. O cálculo dos pesos é realizado de acordo com o erro apresentado pelas florestas durante o treinamento. Na proposta de Utkin et al. [61] o atributo aumentado produzido por uma floresta é resultado de uma média ponderada de suas árvores, ou seja, uma floresta possui uma série de pesos de acordo com o erro de suas árvores. Essa estratégia apresentou avanço do DF em 15 coleções de classificação e regressão com ganho em acurácia entre 0,08% e 2,23%. Fu et al. [21] utiliza os pesos das florestas para produzir um único atributo aumentado por camada com a média ponderada das

predições de suas florestas. A estratégia de ponderar as florestas busca avaliar a qualidade das informações produzidas. Essa avaliação impacta diretamente na importância dada aos atributos aumentados durante treinamento e/ou predições do modelo.

Grupo 6: Aplicações em problemas específicos

Diversos trabalhos na literatura não buscam melhorar o Deep Forest ([14], [23], [40], [55], [68], [69], [72], [73], [74], [77]), ao contrário dos grupos anteriores. Esses trabalhos apenas adaptam DF a um contexto específico, sem realizar modificações, como Korycki e Krawczyk [77] que trocam o *base learner* padrão (RF e florestas de árvores completamente aleatórias) por uma versão adaptada para lidar com fluxos de dados (*Adaptive Random Forest*).

Por tratarem de problemas particulares, esses trabalhos utilizam coleções de dados específicas para os problemas em questão, como coleções para detecção de defeitos de software [74], para estimativa de danos potenciais de eletrodomésticos [68] ou para classificação de produtos de comércio eletrônico [14]. Porém, os trabalhos que lidam com regressão e classificação de forma genérica utilizam diversas coleções de dados para avaliarem seus modelos. Assim, reunimos na tabela 3.1 as coleções mais utilizadas, as respectivas tarefas associadas e os trabalhos que as utilizam.

Tabela 3.1: Coleções mais utilizadas nos experimentos dos trabalhos relacionados ao Deep Forest.

Coleção	**Tarefa Associada**	**Trabalhos**
MINIST	Classificação	[76], [62], [37], [42], [22], [44], [45], [40]
sENG	Classificação	[76], [22], [45]
IMDB	Regressão/Classificação	[76], [60], [64], [22]
LETTER	Classificação	[76], [60], [64], [37], [22], [45], [40]
Adult Income	Classificação	[76], [61], [60], [64], [37], [22], [45]
YEAST	Classificação	[76], [60], [59], [64], [37], [44]
CIFAR-10	Classificação	[76], [37], [22], [44], [45]
Default of Credit Card Clients	Classificação	[61], [64], [37]
Diabetic Retinopathy	Classificação	[61], [60], [59]
Haberman's Breast Cancer Survival	Classificação	[61], [63], [60], [59], [37]
Ionosphere	Classificação	[61], [63], [60], [59], [62], [44]
Seeds	Classificação/Agrupamento	[61], [63], [60], [59], [44]
Teaching Assistant Evaluation	Classificação	[61], [60], [59]
Tic-Tac-Toe Endgame	Classificação	[61], [60], [40]
Wholesale Customer Region	Classificação/Agrupamento	[61], [60], [59]
Parkinsons	Classificação	[61], [63], [59], [64], [62], [44]
Breast Cancer Wisconsin (Diagnostic)	Classificação	[61], [64], [37]
Ecoli	Classificação	[63], [60], [59], [62], [22]
Mammographic masses	Classificação	[63], [59], [37], [22]
Car	Classificação	[60], [59], [64], [37], [44]
Wine	Classificação	[37], [22], [18]

CAPÍTULO 4

Estratégias de avanço para o modelo Deep Forest

Nesse capítulo são apresentadas propostas que visam melhorar os resultados de LeToR quando aplicamos alterações com os modelos DF. As estratégias de avanço podem explorar a adequação de *ensembles* e seus *base learners*, ou somente na definição do *base learner* que integra o Deep Forest. Esses dois tipos de estratégias serão detalhadas a seguir.

4.1 Estratégias de avanço estrutural

Com foco nas estratégias de avanço que alteram a estrutura *ensemble* do DF, temos a pergunta de pesquisa **S3** "*Considerando uma seleção de trabalhos propostos na literatura que visam avançar na qualidade dos resultados do DF, podemos dizer que alguma estratégia se mostra mais efetiva para a tarefa de LeToR?*". Foram definidas cinco estratégias de avanço do DF para LeToR de acordo com propostas da literatura (seção 3.3), onde cada uma delas foi assimilada e adaptada em conformidade com a tarefa de *Learning to Rank*, mantendo Random Forest como *base learner*. As perguntas de pesquisa a seguir representam o que cada estratégia tenta responder:

S3.1 Qual o ganho em considerar as saídas das árvores de cada floresta como atributos aumentados?

S3.1.1 Dado o aumento substancial de atributos e a possível similaridade entre eles, qual o impacto de selecionar os atributos aumentados entre camadas da floresta em cascata?

S3.2 Qual o ganho da estratégia de perpetuar os atributos aumentados?

S3.3 Qual o ganho em aplicar *boosting* entre camadas da floresta em cascata?

S3.3.1 Há ganho em aplicar *boosting* utilizando NDCG para cada consulta e seus documentos relacionados como função de perda?

S3.3.2 Há ganho em aplicar *boosting* utilizando o erro de cada documento, com base no calculo de erro da função de perda *Height* do BROOF-L2R [16]?

A implementação das estratégias propostas serão nomeadas de acordo com a respectiva pergunta de pesquisa relacionada. A descrição de cada uma se encontra a seguir.

S3.1 - Qual a influência de considerar as saídas das árvores de cada floresta como atributos aumentados?

Com base nos trabalhos do grupo 1 da seção 3.3, a pergunta **S3.1** tem como foco a modificação da estrutura dos atributos aumentados. Essa alteração consiste na troca da pontuação gerada por cada floresta pelas pontuações geradas pelas árvores das florestas. Logo, as camadas passam a produzir um atributo aumentado para cada árvore de cada floresta, aumentado a dimensionalidade dos dados.

S3.1 visa ampliar a diversidade dos atributos aumentados, dada a utilização das informações individuais de cada árvore ao invés da informação média de todas elas. Como colocado na seção 2.2, as florestas aplicam *bagging*, logo busca-se treinar cada árvore com um subconjunto diferente da coleção de treinamento. Isso faz com que as árvores tenham perspectivas distintas dos dados e produzam informações diversas. Como consequência, há uma redução da assertividade dos atributos aumentados, compensada pelo aumento da variabilidade nos atributos aumentados, e gerando um consequente aumento na dimensionalidade.

Esse efeito de aumento da dimensionalidade ocorre pois o atributo aumentado de uma floresta é substituído pelo atributo produzidos por cada árvore da floresta. Além disso, também pode ocorrer redundância de atributos, visto que, dado o grande número de árvores, muitas podem gerar informações semelhantes, apesar do *bagging*. Por outro lado, o aumento da variância entre os atributos aumentados pode ajudar na convergência do modelo para resultados mais efetivos.

S3.1.1 - Dado o aumento substancial de atributos e a possível similaridade entre eles, qual o impacto de selecionar os atributos aumentados entre camadas da floresta em cascata?

Tendo em vista a possível ocorrência de redundância de atributos trazida por **S3.1**, é coerente acrescentar à estratégia **S3.1** uma seleção de atributos entre camadas do DF. Esse é objetivo de **S3.1.1**, tendo como foco a qualidade e distinção dos atributos. A seleção de atributos entre camadas da floresta em cascata é uma abordagem dos trabalhos citados no grupo 2 da seção 3.3. De acordo com a pergunta **S3.1.1**, a estratégia proposta estipula a qualidade do atributo de acordo com o NDCG apresentado por ele, visto que cada atributo é uma pontuação de relevância para os documentos. Por outro lado, a

distinção é dada pela correlação média com os demais atributos. Logo, a modificação visa selecionar, durante o treinamento, os atributos com maior NDCG e menor correlação média com os outros atributos. A seleção se trata da identificação da fronteira de Pareto, uma abordagem multiobjetivos, visando a maximização do NDCG e minimização do coeficiente de correlação de Pearson [4] média dos atributos.

S3.2 - Qual a influência da estratégia de perpetuar os atributos aumentados?

A pergunta **S3.2** busca uma análise de uma modificação do Deep Forest baseada no grupo 4 da seção 3.3: perpetuação dos atributos aumentados. A proposta desta alteração do modelo é mudar a forma de propagação dos atributos aumentados. Originalmente, os atributos aumentados produzidos por uma camada C_i são propagados apenas para a camada C_{i+1}. Nessa alteração, os atributos aumentados produzidos por uma camada C_i passam a ser propagados para as camadas $C_{i+1}, C_{i+2}, ..., C_j$, onde C_j é a camada final do modelo. Essa estratégia visa propagar o ganho de informação de cada camada não apenas para a camada seguinte, mas para toda a floresta em cascata. Assim, os meta-atributos, a partir de suas construções, participarão de todo aprendizado até o final do modelo e esperamos que o próprio algoritmo *base learner* seja capaz de fazer a seleção dos atributos internamente para a geração de novos atributos aumentados.

S3.3 - Qual o ganho em aplicar *boosting* entre camadas da floresta em cascata?

Considerando as camadas da floresta em cascata *base learners* em série, percebe-se a compatibilidade do Deep Forest com a aplicação de *boosting*. Se tratando da abordagem *pointwise*, poderíamos tratar a seleção de instâncias para treinamento em duas perspectivas:

- Consulta, selecionando as consultas e todos os documentos que a compõem;
- Documento, selecionando os documentos como instâncias independente das consultas.

Para abordar cada perspectiva, foram escolhidas duas métricas como função de perda para o *boosting*. A nível de consulta será utilizado o NDCG da ordenação dada pelo modelo para a consulta, enquanto a nível de documento será utilizado o erro dos documentos nas ordenações preditas pelo modelo. Cada perspectiva é representado por uma pergunta de pesquisa.

A nível de consulta temos a seguinte pergunta de pesquisa:

S3.3.1 Há ganho em aplicar *boosting* utilizando NDCG para cada consulta e seus documentos relacionados como função de perda?

A modificação trazida pela pergunta **S3.3.1** propõe realizar a seleção de instâncias sobre todo o conjunto de documentos relacionados a uma consulta. Ou seja, dado uma consulta q_i, o conjunto de documentos relacionados $D_i = \{d_{i,1}, d_{i,2}, ..., d_{i,n_i}\}$ participa do treinamento de uma camada se na camada anterior $NDCG(q_i) < \alpha$, onde α é um hiperparâmetro constante de treinamento e $NDCG(q_i)$ é calculado sobre a ordenação produzida de D_i pela camada.

A nível de documento, temos a pergunta de pesquisa:

S3.3.2 Há ganho em aplicar *boosting* utilizando o erro de cada documento, com base no calculo de erro da função de perda *Height* do BROOF-L2R [16]?

A pergunta **S3.3.2** propõe uma seleção de instâncias considerando o erro de cada documento. Dessa forma, é necessária uma maneira de ponderar os documentos individualmente e, para isso, será utilizado uma normalização do erro da função de perda *Height* do BROOF-L2R [16]. Por meio dela, será possível capturar o erro apresentado por cada documento dentro das ordenações preditas pelo modelo. Nessa função de perda, dada uma permutação de documentos t, temos o seguinte calculo de erro para um documento $x_{i,j}$ relacionado à consulta q_i:

$$e^t_{i,j} = \begin{cases} \#\text{ documentos acima de } x_{i,j}, & \text{com relevância menor que} x_{i,j} \\ \#\text{ documentos abaixo de } x_{i,j}, & \text{com relevância maior que} x_{i,j}. \end{cases}$$

Desta forma, temos, como função de perda, a contagem de ordens incorretas (inversões) que ocorrem para cada documento em relação aos demais documentos do ranqueamento. Por fim, a distribuição de probabilidade de seleção de treino para um documentos é dada de maneira proporcional ao número de inversões que ele apresentou no ranqueamento produzido pelo modelo até aquela iteração. Na floresta em cascata, cada documento participa do treinamento da camada seguinte se a camada atual apresenta um erro para o documento maior que β, um hiperparâmetro constante de treinamento.

4.2 (DF-LM) LambdaMART como *base learner* do Deep Forest

A utilização de um *base learner* alternativo ao Random Forest para o Deep Forest é uma estratégia que busca explorar uma possível sensibilidade do DF ao problema tratado, ajustando para isso o *base learner* utilizado. Com uma baixa sensibilidade, o DF apresentaria pouca alteração em seu comportamento e/ou resultados. Caso contrário,

utilizar um *base learner* mais especifico para LeToR pode provocar um avanço nos ranqueamentos produzidos.

A seção 2.3.1 mostra que o modelo LambdaMART se assemelha ao Random Forest por serem modelos *ensembles* baseados em árvores, além de ser um modelo do estado-da-arte de LeToR. Seu uso como *base learner* do Deep Forest pode influenciar na qualidade dos atributos aumentados, melhorando o ganho e propagação de informações da floresta em cascata. Assim, essa estratégia de avanço será denominada **DF-LM** e é contemplada pela pergunta de pesquisa **S4** "*Quais seriam os resultados de efetividade ao utilizarmos o LambdaMART como base learner do Deep Forest?*".

Essa estratégia de avanço do Deep Forest objetiva o melhoramento de seus ranqueamentos pelo uso de um *base learner* do estado-da-arte de LeToR. A despeito do uso de um *base learner* diferente, a estratégia mantém as vantagens do Deep Forest para com as redes neurais profundas, como a facilidade de parametrização e a auto-escalabilidade do modelo.

CAPÍTULO 5

Metodologia

Neste capítulo será explicada toda metologia que acompanha os experimentos elaborados na pesquisa. Serão descritas as coleções de dados de LeToR assim como a maneira pela qual elas foram dividas para treino, validação e teste. Outra informação metodológica repassada nesse capítulo será a descrição dos testes de parametrização que precederam os testes de avaliação dos modelos estudados. Também será mostrada a elaboração metodológica utilizada para a avaliação de viés e variância, através da qual quatro modelos serão comparados. Por último, descrevemos a arquitetura de rede neural profunda utilizada para comparação com o Deep Forest. Todos os valores apresentados tanto para avaliação de parametrização quanto para avaliação de modelos são valores de NDCG, descrito na seção 2.1.2.

Coleções de dados

Três coleções de dados de LeToR foram utilizadas nos experimentos realizados: WEB10K [1], YAHOO! [2] e MQ2007 [3]. Cada coleção é dividida em cinco grupos (*folds*) de mesmo tamanho separados em conjuntos de treino, validação e teste, seguindo a proporção 60%, 20% e 20%, respectivamente. Os conjuntos de validação foram utilizados para verificação de hiperparâmetros, enquanto os conjuntos de teste foram utilizados para avaliação dos modelos, onde também foi aplicado t-teste pareado [52] e os valores indicados por * apresentam significância estatística com 95% de confiança. Na tabela 5.1 encontram-se os detalhes referentes à quantidade de atributos, documentos e consultas das coleções.

[1] Pode ser encontrada em "https://www.microsoft.com/en-us/research/project/mslr".
[2] Pode ser encontrada em "https://webscope.sandbox.yahoo.com".
[3] Pode ser encontrada em "https://www.microsoft.com/en-us/research/project/letor-learning-rank-information-retrieval".

Tabela 5.1: Quantidade de atributos, consultas e documentos das coleções utilizadas.

	Atributos	Consultas	Documentos
WEB10K	136	10.000	1.200.192
YAHOO!	700	6.295	171.988
MQ2007	46	1.700	69.623

Configuração de hiperparâmetros do DF

Para responder à pergunta de pesquisa **S1** "*Qual a influência dos hiperparâmetros do Deep Forest em LeToR?*", foram testados diferentes parametrizações do modelo DF, incluindo a parametrização padrão indicada por Zhou e Feng [76], 8 florestas com 500 árvores por camada da floresta em cascata. Ressalta-se que, independente da quantidade de florestas, metade é composta por RF e a outra metade por florestas de árvores completamente aleatórias.

Nas tabelas 5.2, 5.3 e 5.4 temos os resultados das diferentes configurações avaliadas. O caractere * indica uma diferença com significância estatística em relação ao melhor resultado, destacado em **negrito**. Para as coleções WEB10k e YAHOO!, a melhor configuração encontrada foi de 8 florestas com 80 árvores. Já para a coleção MQ2007, a melhor configuração foi 8 florestas com 40 árvores. Essas foram as respectivas configurações utilizadas para avaliar os modelos em cada coleção de dados.

Tabela 5.2: Teste de parametrização do DF na coleção WEB10K.

	8 florestas	4 florestas
40 árvores	0,4514	0,4498*
80 árvores	**0,4519**	0,4510*
160 árvores	0,4516	0,4496*
300 árvores	0,4517	0,4503*
500 árvores	0,4512	0,4507*

Tabela 5.3: Teste de parametrização do DF na coleção YAHOO!.

	8 florestas	4 florestas
40 árvores	0,7230*	0,7226*
80 árvores	**0,7283**	0,7274*
160 árvores	0,7276	0,7269*
300 árvores	0,7251*	0,7247*
500 árvores	0,7235*	0,7233*

Tabela 5.4: Teste de parametrização do DF na coleção MQ2007.

	8 florestas	4 florestas
40 árvores	**0,4290**	0,4261*
80 árvores	0,4251*	0,4249*
160 árvores	0,4241*	0,4242*
300 árvores	0,4239*	0,4237*
500 árvores	0,4234*	0,4230*

Autores do Deep Forest [76] apontam que a parametrização padrão de 8 florestas com 500 árvores por camada da floresta em cascata tende a oferecer bons resultados em diferentes tarefas. De acordo com os valores mostrados pelas tabelas 5.2, 5.3 e 5.4, observa-se uma variação estatisticamente significativa de NDCG entre as diferentes configurações. Essa variação indica que a parametrização do Deep Forest impacta no seu resultado em LeToR, diferente do que foi afirmado em [76] sobre a robustez dos parâmetros para casos gerais.

Para as três coleções avaliadas, observamos que há diferença estatística nos valores de NDCG quando se reduz o número de florestas por camada da floresta em cascata. Esse comportamento indica que o número de florestas por camada é um hiperparâmetro que requer otimização. A redução do número de florestas altera a quantidade de meta-atributos produzidos pelo modelo o que impacta seu desempenho.

Por outro lado, a variação do número de árvores por floresta gerou variações estatisticamente relevantes de NDCG de acordo com o tamanho da coleção. Na WEB10k, maior coleção, esse hiperparâmetro não indicou variar o modelo significativamente. Na coleção YAHOO!, uma coleção intermediária em termos de número de consultas/documentos, o melhor valor pra o número de árvores apresentou NDCG que superou estatisticamente alguns valores e não superou estatisticamente outros. Por último, na menor coleção avaliada MQ2007, todos os valores de hiperparâmetros foram superados estatisticamente pelo melhor resultado.

Acho que precisa de uma conclusão aqui, tipo: Ou seja, diferente do que foi afirmado originalmente para caso geral [76], em LeToR a análise de parâmetros com DF é uma tarefa importante. Assim, daqui para frente utilizaremos os parâmetros que apresentaram os melhores resultados.

Configuração de hiperparâmetros para propostas de avanço estrutural

Olhando a parametrização das propostas de avanço estrutural para o DF, **S3.3.1**[4] e **S3.3.2**[5] apresentam hiperparâmetros próprios, α e β, respectivamente. α é o limiar com base no NDCG para seleção de consultas, enquanto β é um limiar com base no erro da função de perda *Height* para seleção de documentos. Esses hiperparâmetros foram avaliados no conjunto de validação da coleção WEB10K considerando os valores $\{0.3, 0.5, 0.7\}$ contendo um alto, médio e baixo rigor de corte para o *boosting* aplicado. De acordo com os resultados da figura 5.1, os valores definidos para os testes foram $\alpha = 0.3$ e $\beta = 0.7$.

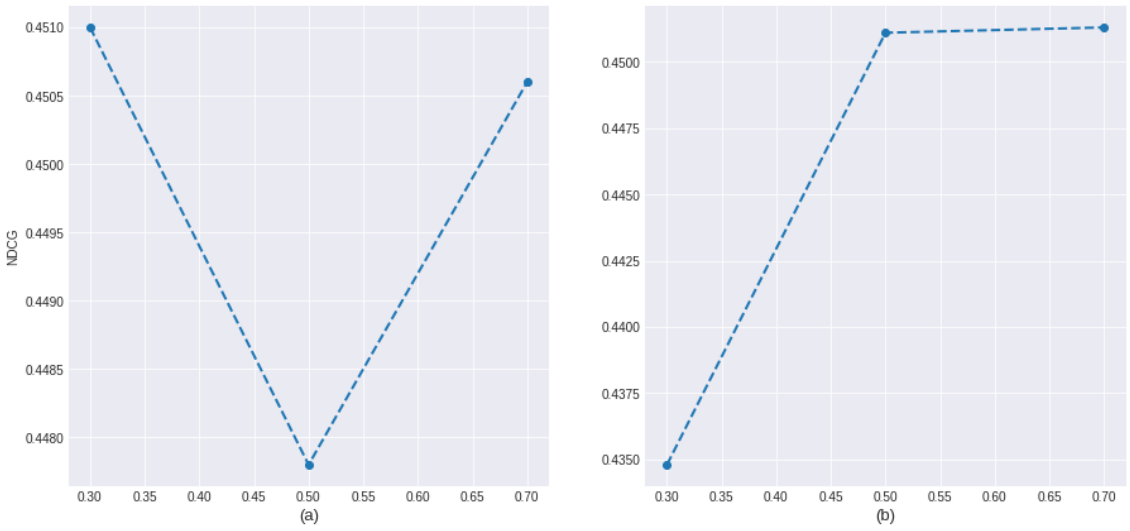

Figura 5.1: Resultado dos testes dos parâmetros das estratégias avaliadas (a) S3.3.1 e (b) S3.3.2. No eixo x temos os valores avaliados para os hiperparâmetros.

Como podemos observar, **S3.3.1** se beneficiou de um α baixo e, portanto, de uma seleção de consultas que apresentaram menor NDCG na predição realizada pela camada anterior do Deep Forest. Por sua vez, **S3.3.2** apresentou melhor resultado com valor alto para β. Ou seja, um limiar de *boosting* que seleciona individualmente os documentos que mostraram maior erro na predição da camada anterior. Portanto, ambas as estratégias indicaram melhores resultados com um *boosting* mais restrito à região de documentos mais difíceis de realizar a predição de ordenação.

[4] Aplicar boosting utilizando NDCG de cada consulta como função de perda.

[5] Aplicar boosting utilizando o erro de cada documento, com base no calculo de erro da função de perda Height do BROOF-L2R

Configuração de hiperparâmetros para DF-LM

O uso do LambdaMART como *base learner* para o Deep Forest (DF-LM) traz a necessidade de avaliar as configurações de parametrização novamente. Porém, por se tratar apenas da troca do *base learner* – ou seja, do tipo de floresta – foi mantido a quantidade de florestas (8) definida na avaliação de hiperparâmetros realizada anteriormente. Essa manutenção também auxilia na comparação entre os modelos – Deep Forest com Random Forest e Deep Forest com LambdaMART – visto que não interfere na complexidade específica do Deep Forest, e sim dos *base learners*. Isso conduz para que os impactos encontrados no modelo sejam motivados apenas pela troca do *base learner*. Logo, a avaliação de parametrização do DF-LM passou apenas pelo parâmetro de número de árvores por LambdaMART da floresta em cascata, como mostrado tabela 5.5.

Tabela 5.5: Teste de parametrização de DF-LM com 8 LambdaMARTs por camada.

	WEB10K	YAHOO!	MQ2007
40 árvores	0,4587*	0,7229*	0,4480*
80 árvores	0,4591*	0,7238*	0,4496
160 árvores	0,4625*	0,7284*	**0,4508**
300 árvores	**0,4649**	**0,7316**	0,4498
500 árvores	0,4648	0,7310	0,4489*

Os resultados mostrados na tabela 5.5 indicam o melhor valor de NDCG com 300 árvores para as coleções WEB10k e YAHOO!, e 160 árvores para a coleção MQ2007. Nas coleções WEB10k e YAHOO!, o aumento do número de árvores indica melhora estatística do modelo até 300 árvores, onde o aumento para 500 árvores não mostra mudança estatisticamente relevante. Já na coleção MQ2007, os valores de 80, 160 e 300 árvores não apresentaram uma diferença estatística. Nessa coleção, o menor e maior valor avaliado de número de árvores mostraram ser inferiores estatisticamente aos três valores intermediários.

Análise de viés e variância

A análise de viés e variância desse trabalho foi realizada de acordo com a definição para decomposição de erro em viés e variância abordada na seção 2.4.1. Como destacado por Shivaswamy e Chandrashekar [56], viés e variância são definidos sobre uma distribuição de modelos a partir de treinamentos. Assim, treinando um mesmo algoritmo com diferentes conjuntos de dados de treino e obtemos uma distribuição de modelos para o algoritmo onde pode ser calculado viés e variância.

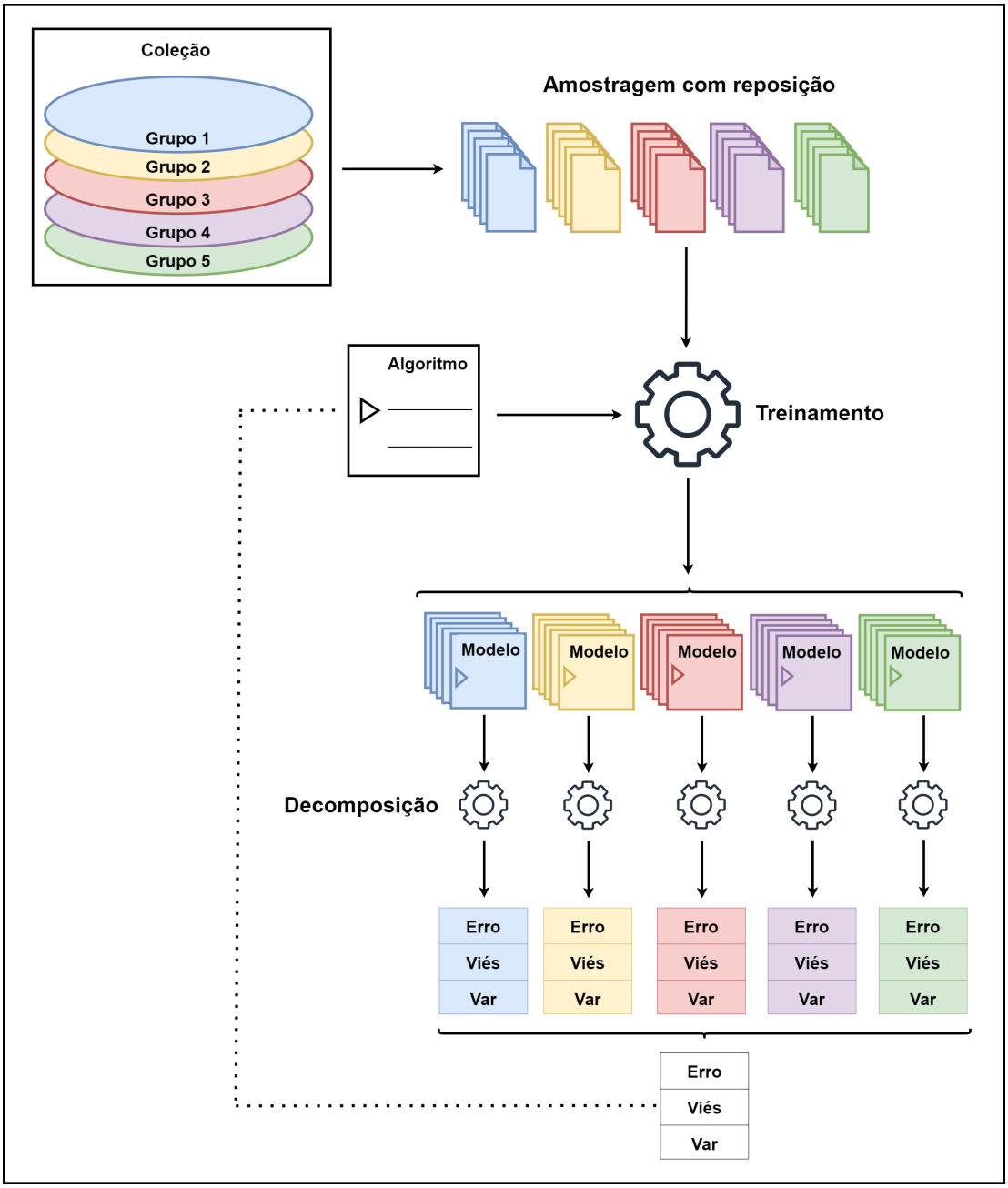

Figura 5.2: Metodologia de treinamento de modelos para a análise de viés e variância. Cada grupo da coleção produziu cinco amostras de treino com reposição. O treinamento do algoritmo em cada amostra resultou em um modelo. Ao final temos 25 modelos do algoritmo utilizados para cálculo de decomposição do erro em viés e variância.

A figura 5.2 ilustra a metodologia aplicada para produzir a distribuição de modelos de acordo com cada algoritmo utilizado na avaliação. Considerando que as coleções utilizadas foram divididas em cinco grupos (ou cinco *folds*), os algoritmos a serem efetuada a análise passaram por cinco treinamentos em cada grupo. Todos os treinamentos foram realizados sobre uma amostragem com reposição do conjunto de treino. O erro, viés e variância foram computados sobre os cinco treinamentos de cada grupo e, ao final,

através da média, obtemos o erro, viés e variância geral do algoritmo. Logo, ao todo, cada algoritmo passou por 25 treinamento (5 grupos $\times$ 5 treinamentos) para computar seu erro, viés e variância (Var).

Redes Neurais Profundas

Para responder à pergunta **S6** "*Qual a comparação de resultados entre Deep Forest e Redes Neurais Profundas para LeToR?*", foi utilizado o trabalho de Pobrotyn et al. [46]. Esse trabalho avalia redes neurais profundas em LeToR, e é apontado por diversos trabalhos como indicador de estado da arte de RNP em LeToR [2, 3, 13, 48]. Pobrotyn et al. [46] avaliam diferentes funções de perda e atingem os melhores resultados para MLP (*Multilayer Perceptron*) com *NDCGLoss 2++* [65].

No presente trabalho, utilizamos a mesma configuração para MLP utilizada por Pobrotyn et al. [46]. Essa configuração é:

- Cinco camadas ocultas de dimensões [256, 512, 1024, 512, 256].
- Função de ativação ReLU.
- Taxa de aprendizado $1e^{-3}$.
- *Dropout* de 0,3.

CAPÍTULO 6

Resultados

De acordo com a metodologia estabelecida no Capítulo 5, diversos testes foram realizados buscando responder às perguntas de pesquisa do capítulo 1. Assim, o presente capítulo compila os resultados finais dos experimentos realizados. Assim como no capítulo anterior, nas tabelas apresentadas, o caractere * indica uma diferença com significância estatística em relação ao melhor resultado, destacado em **negrito**, em cada tabela de resultados.

6.1 Deep Forest e Random Forest

Seguindo a pergunta de pesquisa **S2** "*Sendo Deep Forest um modelo ensemble de Random Forest, qual a comparação dos resultados de ambos os modelos em LeToR?*", a tabela 6.1 mostra os valores de NDCG10 obtidos para o DF e RF. Considerando o RF *base learner* do DF, foi mantida a dimensionalidade para o RF de 80 árvores – WEB10K e YAHOO! – e 40 árvores – para a coleção MQ2007 – utilizadas nas florestas do DF.

Tabela 6.1: Avaliação comparativa entre Random Forest e Deep Forest.

	WEB10K	Yahoo!	MQ2007
Random Forest	0,4417*	0,7140*	0,4132*
Deep Forest	**0,4522**	**0,7241**	**0,4376**

Como podemos observar, o Deep Forest é capaz de superar Random Forest, inclusive mantendo uma diferença com significância estatística. Isso monstra a eficácia e justifica o uso do método *ensemble* aplicado com DF no contexto de Learning to Rank.

6.2 Deep Forest e as estratégias de avanço estrutural

A seguir, temos a recapitulação das estratégias de avanço do Deep Forest referentes à pergunta de pesquisa **S3** propostas no capítulo 4:

S3.1 Considerar as saídas das árvores de cada floresta como atributos aumentados;

S3.1.1 Considerar as saídas das árvores de cada floresta como atributos aumentados e aplicar uma seleção de atributos entre camadas;

S3.2 Perpetuar os atributos aumentados;

S3.3.1 Aplicar *boosting* utilizando NDCG de cada consulta como função de perda;

S3.3.2 Aplicar *boosting* utilizando o erro de cada documento, com base no calculo de erro da função de perda *Height* do BROOF-L2R [16].

A tabela 6.2 traz os resultados obtidos para as estratégias de avanço baseadas na literatura. Podemos observar que nenhuma das estratégias testadas apresentaram ganho em relação ao modelo original do Deep Forest. Isso mostra que as estratégias não foram capazes de melhorar a representação dos dados e/ou nível de informação do modelo em LeToR.

Um possível fator determinante para esse comportamento trata-se da diferença entre o problema de regressão e o de classificação, o qual as estratégias de avanço foram utilizadas originalmente. Quando em classificação, a presença do vetor de probabilidade de classes traz para os meta atributos uma distribuição das classes predita pelas florestas. Assim, toda predição traz uma indicação das probabilidades das instâncias pertencerem a cada classe e essa distribuição probabilística compõem os meta atributos. A maior parte dos trabalhos na literatura se baseia nessa distribuição para realizar suas estratégias de avanço do Deep Forest. Quando trazido para regressão, o DF pode sofrer uma perda de significado dos meta atributos, onde passam a representar apenas um valor único de predição.

Tabela 6.2: Avaliação comparativa das propostas de avanço para o Deep Forest baseadas na literatura.

	WEB10K	Yahoo!	MQ2007
Deep Forest	**0,4522**	**0,7241**	**0,4376**
S3.1	0,4388*	0,7067*	0,4351
S3.1.1	0,4392*	0,7064*	0,4347*
S3.2	0,4515	0,7235	0,4232*
S3.3.1	0,4513	0,7222*	0,4368*
S3.3.2	0,4520	0,7226*	0,4365*

6.3 Deep Forest e LambdaMART

Para a pergunta de pesquisa **S4** "*Quais seriam os resultados de efetividade ao utilizarmos o LambdaMART como base learner do Deep Forest?*", a tabela 6.3

apresenta os resultados obtidos para os três modelos envolvidos: DF, DF-LM (DF com LambdaMART como *base learner*) e o LambdaMART.

Tabela 6.3: Avaliação comparativa do Deep Forest, DF-LM e LambdaMART. Entre parênteses temos a diferença de NDCG aproximada em relação ao DF-LM.

	WEB10K	Yahoo!	MQ2007
Deep Forest	0,4522* (-2,94%)	0,7241* (-0,62%)	0,4376* (-2,83%)
LambdaMART	0,4578* (-1,68%)	0,7188* (-1,36%)	0,4372* (-2,93%)
DF-LM	**0,4655**	**0,7286**	**0,4500**

Como pode ser observado, o LambdaMART, como um algoritmo do estado-da-arte de LeToR, apresenta melhores resultados que o Deep Forest. Porém, quando utilizado como *base learner* do DF, visto que é um modelo *ensemble* baseado em árvores assim como o Random Forest, produz uma combinação que supera ambos Deep Forest e LambdaMART. Novamente, o Deep Forest confirma sua capacidade enquanto modelo *ensemble* de aprimorar os resultados de seu *base learner*.

6.4 Análise de viés e variância

Buscando compreender o comportamento do Deep Forest em LeToR, uma análise de viés e variância traz o entendimento de suas contribuições para o erro apresentado pelo modelo. De forma interpretativa, como mostrado na seção 2.4, viés e variância indicam a capacidade de aprendizado e generalização do modelo atrelados à complexidade dele. Comparativamente, essa análise pode apontar as diferenças de complexidade e comportamento entre modelos distintos ou compostos, como o RF e DF.

Considerando o Deep Forest, DF-LM, Random Forest e LambdaMART – respectivamente, o modelo de referência, o modelo com melhor NDCG, e seus *base learners* RF e LambdaMART – a tabela 6.4 compila os resultados obtidos para viés e variância.

Tabela 6.4: Erro, viés e variância para Random Forest, Deep Forest, DF-LM e LambdaMART.

	WEB10K			YAHOO!			MQ2007		
	Erro	Viés	Var	Erro	Viés	Var	Erro	Viés	Var
Random Forest	0,4133	**0,3656**	0,0477	0,3592	0,3170	0,0422	0,4344	0,3752	0,0592
Deep Forest	0,4124	0,3718	0,0406	0,3547	0,3143	0,0404	0,4341	0,3812	0,0529
DF-LM	**0,4078**	0,3941	**0,0137**	**0,3528**	0,3325	**0,0203**	**0,4283**	0,4083	**0,0200**
LambdaMART	0,4118	0,3668	0,0450	0,3553	**0,2909**	0,0644	0,4335	**0,3482**	0,0853

Em alguns trabalhos, o Random Forest é conhecido por sofrer de *overfitting* (super ajuste aos dados de treino), principalmente em dados que tenham muito ruído [25,

36, 54]. Esse overfitting pode prejudicar a generalização dos modelos, e está associado a alta variância nos dado de treinamento. Algo que pode ser observado na Tabela 6.4. A redução da variância em conjunto com uma estabilidade – ou, ao menos, um pequeno aumento – do viés é um fenômeno que pode contribuir para a diminuição do erro total do RF, como descrito no trabalho de [53].

Como podemos observar nos resultados mostrados na tabela 6.4, o Deep Forest apresenta o mesmo fenômeno descrito em [53] em relação ao Random Forest, nas três coleções para a tarefa de LeToR. Utilizando os valores da coleção WEB10K para ilustrar, apesar do viés sofrer um aumento – *(RF) 0,3656 para (DF) 0,3718* – (+0,0062), a perda de variância – *(RF) 0,0477 para (DF) 0,0406* – (-0,0071) é responsável pela diminuição da taxa de erro – *(RF) 0,4133 para (DF) 0,4124* (-0,0009).

O mesmo comportamento pode ser verificado para DF-LM em relação ao LambdaMART nas três coleções, porém de forma mais acentuada. Continuando com o uso dos valores da coleção WEB10K para ilustrar a queda na variância – *(LambdaMART) 0,0450 para (DF-LM) 0,0137* – (-0,0313) compensa, em maior grau, o ganho de viés – *(LambdaMART) 0,3668 para (DF-LM) 0,3941* – (+0,0273). O resultado é que DF-LM reduz o erro do LambdaMART em 0,004 para a coleção WEB10K, 0,0025 para YAHOO! e 0,0052 para MQ2007.

Esses resultados indicam uma redução de complexidade dos *base learners* para o Deep Forest e se relaciona com a capacidade do Deep Forest de autoajustar a complexidade de acordo com os dados de treino, como indicado pelos autores Zhou e Feng [76]. Ou seja, o auto escalonamento da floresta em cascata produz uma ajuste na complexidade do próprio DF em relação ao seu *base learner*, o que resulta na redução da taxa de erro.

Quanto ao Random Forest e LambdaMART nas coleções YAHOO! e MQ2007, o algoritmo LambdaMART apresenta redução na taxa de erro em relação ao algoritmo Random Forest (YAHOO! -0,0039 e MQ2007 -0,0009) por meio de uma redução no viés (YAHOO! -0,0261 e MQ2007 -0,027). Acompanhado a isso, encontramos um ganho em variância inversamente proporcional ao tamanho da coleção (YAHOO! +0,0222 e MQ2007 +0,0261), ou seja, a menor coleção, no caso MQ2007, tem maior aumento de variância. Esse é um comportamento esperado dado que a redução da dimensionalidade e instância do conjuntos de dados pode conduzir a um super ajuste, e à maior variância do modelo [25].

Apesar dos algoritmos Random Forest e o LambdaMART estarem mais suscetíveis à mudança na variância. O Deep Forest e o DF-LM não apresentam o mesmo comportamento entre si, o que poderia ser esperado por terem como *base learners* o RF e o LambdaMART, respectivamente. O que identificamos nas três coleções, é que o DF apresenta um comportamento de redução da variância sobrepondo o aumento de viés e,

assim, reduzindo o erro. Ou seja, o Deep Forest atua, de modo geral, na redução da variância. Dada essa análise, entendemos que um possível ponto de avanço no modelo DF seria buscar uma redução de viés – ou, ao menos, uma redução do ganho de viés –, dado que a redução da variância já é realizada pelo algoritmo.

6.5 Deep Forest e Redes Neurais Profundas

Zhou e Feng [76] colocam o Deep Forest como alternativa às redes neurais profundas para a realização de aprendizado profundo. Essa asserção, contudo, não havia sido verificada para a tarefa de *Learning to Rank*. A tabela 6.5 a seguir compara o desempenho, em termos de NDCG, dos modelos Deep Forest, DF-LM e rede neural profunda (MLP – Multilayer Perceptron) conforme definido no capítulo 5.

Tabela 6.5: Avalação comparativa entre Deep Forest, DF-LM e RNP (Rede Neural Profunda).

	WEB10K	Yahoo!	MQ2007
Deep Forest	0,4522*	0,7241*	0,4376*
DF-LM	0,4655*	0,7286*	**0,4500**
MLP	**0,4702**	**0,7384**	0,4490

Os resultados mostram que o Deep Forest não supera redes neurais profundas na tarefa de LeToR. DF apresenta valores superados estatisticamente pela RNP em até 3,98%. Por sua vez, DF-LM alcança NDCG mais próximo ao da RNP – 1% menor na WEB10k e 1,34% menor na Yahoo!, aproximadamente – chegando a um NDCG maior na coleção MQ2007, porém sem relevância estatística.

Como descrito no capítulo 1, três características são levantadas quanto a desvantagens das redes neurais profundas em LeToR: transformação dos atributos de entrada, arquitetura da rede e escassez de dados [50]. Os resultados da tabela 6.5 mostram que o Deep Forest não foi capaz de cobrir essas lacunas de forma a superar a RNP. Quanto à arquitetura da rede, vemos que o mecanismo de produção de meta atributos baseado em predição do DF pode não ter sido tão capaz, em comparação à RNP, de capturar interações de alta ordem dos dados das coleções avaliadas. Por outro lado, quanto à escassez de dados, vemos que o DF apresentou resultados mais próximos à RNP nas coleções menores, sobretudo o DF-LM. Esse é um comportamento que reforça a qualidade de auto ajuste de complexidade do Deep Forest

O Deep Forest, como destacado diversas vezes anteriormente, foi proposto como um método de aprendizado profundo enfatizando dois aspectos: processamento camada a camada e transformação de atributos durante o processamento. Sob essa perspectiva o DF

realiza aprendizado profundo com módulos não diferenciáveis. Apesar disso, não mostrou satisfatoriedade em contrapor os resultados de redes neurais profundas nas coleções avaliadas para LeToR.

Como indicado por Zhou e Feng [76], o Deep Forest sinaliza a possibilidade de abordar aprendizado profundo sem a utilização de módulo diferenciáveis. Observamos que essa hipótese também se valida em LeToR, porém de forma não satisfatória dado a superioridade das redes neurais profundas. Por fim, o DF não se propõe a ser, em última instância, um modelo final de aprendizado profundo sem módulos diferenciáveis, mas sim um indicativo da possibilidade dessa abordagem também em LeToR, sobretudo para o avanço de outros algoritmos baseados em árvores.

CAPÍTULO 7

Conclusão

O presente trabalho buscou estudar a aplicação do modelo Deep Forest à tarefa de Learning to Rank. Esse estudo deriva do bom desempenho do DF em diferentes tarefas e do sucesso em LeToR de modelos ensemble baseados em árvores. Um estudo com esse foco contribui, também, para o entendimento do comportamento de viés e variância de modelos *ensemble* baseados em árvores, o que inclui algoritmos do estado-da-arte como LambdaMART.

De modo específico, o trabalho buscou responder às perguntas de pesquisa propostas no capítulo 1, onde o objetivo final é a verificação da possibilidade da realização de aprendizado profundo com módulos não diferenciáveis, seguindo a estratégia Deep Forest. Zhou e Feng [76] apontam essa possibilidade para diferentes tarefas de aprendizado, porém não avaliando-a para Learning to Rank.

Como apresentado no capítulo 1, questões de pesquisa secundárias foram definidas para responderam à pergunta principal. As perguntas de pesquisa são:

P O aprendizado profundo com módulos não diferenciáveis, seguindo a estratégia Deep Forest, é capaz de melhorar a efetividade nas tarefas de LeToR?

S1 Qual a influência dos hiperparâmetros do Deep Forest em LeToR?

S2 Sendo Deep Forest um modelo *ensemble* baseado em Random Forest, qual a comparação dos resultados de ambos os modelos em LeToR?

S3 Considerando uma seleção de trabalhos propostos na literatura que visam avançar na efetividade da estrutura do DF, podemos dizer que algum desses se mostra efetivo para a tarefa de LeToR?

S4 Quais seriam os resultados de efetividade ao utilizarmos o LambdaMART como *base learner* do Deep Forest?

S5 Qual o comportamento de viés e variância do modelo Deep Forest para LeToR?

S6 Qual a comparação de resultados entre Deep Forest e Redes Neurais Profundas para LeToR?

Os autores do Deep Forest destacam não ser necessário um alto custo para ajuste de seus parâmetros [76]. Observamos em LeToR que há uma variação significativa de desempenho em diferentes configurações do DF. Sobretudo o número de árvores por floresta é um hiperparâmetro que necessita de otimização em relação aos dados utilizados. Mostramos que o tamanho da coleção impacta diretamente na parametrização do Deep Forest.

Sendo o Deep Forest um modelo *ensemble*, é importante validar sua estratégia *ensemble* em relação ao *base learner* original, o Random Forest. Deve-se ressaltar que um dos hiperparâmetros do DF avaliados são, na verdade, um hiperparâmetro do Random Forest, o número de árvores por floresta. Os resultados obtidos mostraram que o Deep Forest supera o Random Forest na tarefa de LeToR. Portanto, observamos a eficácia do DF como modelo *ensemble* em LeToR.

Na literatura, diversos trabalhos estudam a estrutura *ensemble* do Deep Forest e propõem avanços ao estado original do modelo. Esses trabalhos, entretanto, não avaliam a possibilidade dos avanços se mostrarem eficazes em LeToR. De fato, as propostas de avanço selecionadas para serem avaliadas aqui não manifestaram melhorias no desempenho do Deep Forest em LeToR. Lembrando que, originalmente, os avanços foram propostos majoritariamente (como indicado na tabela 3.1) para tarefas de classificação, onde o mecanismo de transformação de atributos conta com maior grau de informações por meio do vetor de distribuição de classes, o que não é encontrado em regressão – problema base de LeToR. Outro avanço, proposto no presente trabalho, é a aplicação de *boosting* entre as camadas da floresta em cascata do Deep Forest. Também foi um avanço que se mostrou ineficaz tanto a nível de consulta quanto a nível de documento.

Também é encontrado na literatura propostas de avanço/uso do Deep Forest onde trocam-se o *base learner* original, Random Forest, por *base learners* distintos. Essa troca é possibilitada pela estrutura *ensemble* do Deep Forest, e pode ser realizada buscando tanto avançar o desempenho do modelo quanto possibilitar o seu uso em cenários específicos de dados. No caso de Learning to Rank, a troca do Random Forest por um algoritmo do estado-da-arte de LeToR também baseado em árvores, o LambdaMART, apresentou melhora no desempenho do DF. Essa estratégia traz não só o avanço do Deep Forest em LeToR, como também coloca o Deep Forest como meio de avanço para outros modelos não neurais, em destaque àqueles baseados em árvores. Algoritmos como o LambdaMART podem encontrar no Deep Forest um modelo *ensemble* que avança seus resultados.

Analisando o comportamento de viés e variância dos modelos para entender o comportamento do Deep Forest em relação aos seus algoritmos base, é possível perceber que o Deep Forest atua para uma redução significativa da variância de seu *base learner*. Apesar de causar também um aumento de viés, a redução da variância possui grau

suficiente para reduzir o erro geral do modelo em relação ao *base learner*. Destacado que o Deep Forest atua na redução da variância, encontramos um ponto de avanço por meio da redução de viés do algoritmo. Percebemos que esse ponto pode ser trabalhado, em coleções menores, através da escolha do *base learner*. Quando utilizado um *base learner* de menor viés, o Deep Forest atua na redução da variância, e o ganho de viés é reduzido pelo baixo viés do *base learner*.

Dado que aprendizado profundo é composto por três características principais [76], isto é, transformação de atributos na execução do modelo, complexidade ajustada do modelo e processamento em camadas, todas encontradas no Deep Forest, podemos concluir que o aprendizado profundo com módulos não diferenciáveis é viável para o aprimoramento de outros algoritmos baseados em árvores ainda que no estado-da-arte de LeToR. Porém, em comparação com redes neurais profundas, o DF indica uma possível contraposição apenas em coleções com menor quantidade de consultas/documentos.

Ainda assim, o Deep Forest oferece uma lacuna de avanço para Learning To Rank. Como Zhou e Feng [76], também encontramos, mas para LeToR, a necessidade de desenvolver uma forma alternativa para a representação e transformação dos atributos aumentados. Um processo que agregue mais informações aos meta-atributos, sobretudo informações específicas de ranqueamento, poderia levar a um avanço relevante do Deep Forest em LeToR.

Complementando um possível estudo de desenvolvimento de uma forma alternativa de representação e transformação dos atributos aumentados, uma análise do comportamento do Deep Forest camada-a-camada traria luz ao funcionamento interno do algoritmo. A aplicação de uma análise de viés e variância em cada camada do Deep Forest, por exemplo, pode mostrar o impacto dos atributos aumentados e esclarecer como o mecanismo de auto escalonamento atua sobre a complexidade do modelo. Ainda em tarefas específicas como Learning to Rank, o estudo do Deep Forest se mostra coerente e aberto a desenvolvimento.

Referências Bibliográficas

[1] AMIGÓ, E.; FANG, H.; MIZZARO, S.; ZHAI, C. **Are we on the right track? an examination of information retrieval methodologies**. In: *The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval*, SIGIR '18, p. 997–1000, New York, NY, USA, 2018. Association for Computing Machinery.

[2] BEKOULIS, G.; PAPAGIANNOPOULOU, C.; DELIGIANNIS, N. **A review on fact extraction and verification**. *ACM Comput. Surv.*, 55(1), nov 2021.

[3] BEKOULIS, G.; PAPAGIANNOPOULOU, C.; DELIGIANNIS, N. **Understanding the impact of evidence-aware sentence selection for fact checking**. In: *Proceedings of the Fourth Workshop on NLP for Internet Freedom: Censorship, Disinformation, and Propaganda*, p. 23–28, Online, June 2021. Association for Computational Linguistics.

[4] BENESTY, J.; CHEN, J.; HUANG, Y.; COHEN, I. **Pearson Correlation Coefficient**, p. 1–4. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009.

[5] BREIMAN, L. **Bagging predictors**. *Machine Learning*, 24(2):123–140, Aug. 1996.

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

[7] BRIN, S.; PAGE, L. **The anatomy of a large-scale hypertextual web search engine**. *Computer Networks*, 30:107–117, 1998.

[8] BURGES, C.; SHAKED, T.; RENSHAW, E.; LAZIER, A.; DEEDS, M.; HAMILTON, N.; HULLENDER, G. **Learning to rank using gradient descent**. In: *Proceedings of the 22nd International Conference on Machine Learning*, ICML '05, p. 89–96, New York, NY, USA, 2005. Association for Computing Machinery.

[9] BURGES, C.; RAGNO, R.; LE, Q. **Learning to rank with nonsmooth cost functions**. In: Schölkopf, B.; Platt, J.; Hoffman, T., editors, *Advances in Neural Information Processing Systems*, volume 19. MIT Press, 2006.

[10] CAPANNINI, G.; DATO, D.; LUCCHESE, C.; MORI, M.; NARDINI, F. M.; ORLANDO, S.; PEREGO, R.; TONELLOTTO, N. **Quickrank: A c++ suite of learning to rank algorithms**. *CEUR Workshop Proceedings*, 1404, 01 2015.

[11] CAPRA, R.; PEREZ-QUINONES, M. **Using web search engines to find and refind information**. *Computer*, 38(10):36–42, 2005.

[12] CHAPELLE, O.; CHANG, Y. **Yahoo! learning to rank challenge overview**. In: *Proceedings of the 2010 International Conference on Yahoo! Learning to Rank Challenge - Volume 14*, YLRC'10, p. 1–24. JMLR.org, 2010.

[13] CHEN, M.; LIU, C.; SUN, J.; HOI, S. C. **Adapting Interactional Observation Embedding for Counterfactual Learning to Rank**, p. 285–294. Association for Computing Machinery, New York, NY, USA, 2021.

[14] DAI, J.; WANG, T.; WANG, S. **A deep forest method for classifying e-commerce products by using title information**. In: *2020 International Conference on Computing, Networking and Communications (ICNC)*, p. 1–5, 2020.

[15] DE CASTRO MENDES GOMES, G.; DE OLIVEIRA, V. C.; DE ALMEIDA, J. M.; GONÇALVES, M. A. **Is learning to rank worth it? a statistical analysis of learning to rank methods**, 2013.

[16] DE SÁ, C. C.; GONÇALVES, M. A.; SOUSA, D. X.; SALLES, T. **Generalized broof-l2r: A general framework for learning to rank based on boosting and random forests**. In: *Proceedings of the 39th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '16, p. 95–104, New York, NY, USA, 2016. Association for Computing Machinery.

[17] DONG, X.; YU, Z.; CAO, W.; SHI, Y.; MA, Q. **A survey on ensemble learning**. *Frontiers of Computer Science*, 14(2):241–258, Aug. 2019.

[18] FAN, Y.; QI, L.; TIE, Y. **The cascade improved model based deep forest for small-scale datasets classification**. In: *2019 8th International Symposium on Next Generation Electronics (ISNE)*, p. 1–3, 2019.

[19] FREUND, Y.; SCHAPIRE, R. E. **A decision-theoretic generalization of on-line learning and an application to boosting**. *Journal of Computer and System Sciences*, 55(1):119 – 139, 1997.

[20] FRIEDMAN, J. H. **Greedy function approximation: A gradient boosting machine.** *The Annals of Statistics*, 29(5):1189 – 1232, 2001.

[21] FU, M.; YUAN, J.; BEI, C. **Early sepsis prediction in ICU trauma patients with using an improved cascade deep forest model**. In: *2019 IEEE 10th International Conference on Software Engineering and Service Science (ICSESS)*. IEEE, Oct. 2019.

[22] GAO, J.; LIU, K.; WANG, B.; WANG, D.; HONG, Q. **An improved deep forest for alleviating the data imbalance problem**. *Soft Computing*, Aug. 2020.

[23] HAN, M.; LI, S.; WAN, X.; LIU, G. **Scene recognition with convolutional residual features via deep forest**. In: *2018 IEEE 3rd International Conference on Image, Vision and Computing (ICIVC)*. IEEE, June 2018.

[24] HAN, S.; WANG, X.; BENDERSKY, M.; NAJORK, M. **Learning-to-rank with bert in tf-ranking**, 2020.

[25] HASTIE, T.; TIBSHIRANI, R.; FRIEDMAN, J. **The Elements of Statistical Learning**. Springer New York, 2009.

[26] HERBRICH, R.; GRAEPEL, T.; OBERMAYE, K. **Large Margin Rank Boundaries for Ordinal Regression**. In: *Advances in Large-Margin Classifiers*. The MIT Press, 09 2000.

[27] HU, Z.; WANG, Y.; PENG, Q.; LI, H. **Unbiased lambdamart: An unbiased pairwise learning-to-rank algorithm**. In: *The World Wide Web Conference*, WWW '19, p. 2830–2836, New York, NY, USA, 2019. Association for Computing Machinery.

[28] HUANG, G.; LIU, Z.; WEINBERGER, K. Q. **Densely connected convolutional networks**. *CoRR*, abs/1608.06993, 2016.

[29] JUNG, C.; SHEN, Y.; JIAO, L. **Learning to rank with ensemble ranking SVM**. *Neural Processing Letters*, 42(3):703–714, Oct. 2014.

[30] KARATZOGLOU, A.; BALTRUNAS, L.; SHI, Y. **Learning to rank for recommender systems**. In: *Proceedings of the 7th ACM Conference on Recommender Systems*, RecSys '13, p. 493–494, New York, NY, USA, 2013. Association for Computing Machinery.

[31] 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.

[32] LI, H. **Learning to Rank for Information Retrieval and Natural Language Processing**. Morgan and Claypool Publishers, 2011.

[33] LI, H. **A short introduction to learning to rank**. *IEICE Transactions on Information and Systems*, E94.D(10):1854–1862, 2011.

[34] LI, J.; LIU, G. **An improved lambdamart algorithm based on the matthew effect**. *Mathematical Problems in Engineering*, 2018:1–11, 11 2018.

[35] LI, P.; QIN, Z.; WANG, X.; METZLER, D. **Combining decision trees and neural networks for learning-to-rank in personal search**. In: *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '19, p. 2032–2040, New York, NY, USA, 2019. Association for Computing Machinery.

[36] LIU, F. T. **The utility of randomness in decision tree ensembles**. 2005.

[37] LIU, P.; WANG, X.; YIN, L.; LIU, B. **Flat random forest: a new ensemble learning method towards better training efficiency and adaptive model size to deep forest**. *International Journal of Machine Learning and Cybernetics*, 11(11):2501–2513, May 2020.

[38] LIU, T.-Y. **Learning to rank for information retrieval**. *Found. Trends Inf. Retr.*, 3(3):225–331, Mar. 2009.

[39] LIU, Y.; GE, Z. **Deep ensemble forests for industrial fault classification**. *IFAC Journal of Systems and Control*, 10:100071, 2019.

[40] LUONG, A. V.; NGUYEN, T. T.; LIEW, A. W.-C. **Streaming active deep forest for evolving data stream classification**, 2020.

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

[42] MILLER, K.; HETTINGER, C.; HUMPHERYS, J.; JARVIS, T.; KARTCHNER, D. **Forward thinking: Building deep random forests**, 2017.

[43] MITCHELL, M. W. **Bias of the random forest out-of-bag (OOB) error for certain input parameters**. *Open Journal of Statistics*, 01(03):205–211, 2011.

[44] NI, S.; KAO, H.-Y. **Psforest: Improving deep forest via feature pooling and error screening**. In: Pan, S. J.; Sugiyama, M., editors, *Proceedings of The 12th Asian Conference on Machine Learning*, volume 129 de **Proceedings of Machine Learning Research**, p. 769–781, Bangkok, Thailand, 18–20 Nov 2020. PMLR.

[45] PANG, M.; TING, K.-M.; ZHAO, P.; ZHOU, Z.-H. **Improving deep forest by confidence screening**. In: *2018 IEEE International Conference on Data Mining (ICDM)*. IEEE, Nov. 2018.

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

[47] POLIKAR, R. **Ensemble based systems in decision making**. *IEEE Circuits and Systems Magazine*, 6(3):21–45, 2006.

[48] PURPURA, A.; BUCHNER, K.; SILVELLO, G.; SUSTO, G. A. **Neural feature selection for learning to rank**. In: *Lecture Notes in Computer Science*, p. 342–349. Springer International Publishing, 2021.

[49] PÎRTOACĂ, G.-S.; REBEDEA, T.; RUSETI, S. **Answering questions by learning to rank – learning to rank by answering questions**, 2019.

[50] 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.

[51] SAGI, O.; ROKACH, L. **Ensemble learning: A survey**. *WIREs Data Mining and Knowledge Discovery*, 8(4):e1249, 2018.

[52] SAKAI, T. **Statistical reform in information retrieval?** *SIGIR Forum*, 48(1):3–12, June 2014.

[53] SALLES, T.; ROCHA, L.; GONÇALVES, M. **A bias-variance analysis of state-of-the-art random forest text classifiers**. *Advances in Data Analysis and Classification*, 15(2):379–405, July 2020.

[54] SEGAL, M. **Machine learning benchmarks and random forest regression**. Technical report, eScholarship Repository, University of California, 2004.

[55] SHAO, L.; ZHANG, D.; DU, H.; FU, D. **Deep forest in adhd data classification**. *IEEE Access*, 7:137913–137919, 2019.

[56] SHIVASWAMY, P.; CHANDRASHEKAR, A. **Bias-Variance Decomposition for Ranking**, p. 472–480. Association for Computing Machinery, New York, NY, USA, 2021.

[57] SUN, L.; MO, Z.; YAN, F.; XIA, L.; SHAN, F.; DING, Z.; SHAO, W.; SHI, F.; YUAN, H.; JIANG, H.; WU, D.; WEI, Y.; GAO, Y.; GAO, W.; SUI, H.; ZHANG, D.; SHEN, D. **Adaptive feature selection guided deep forest for covid-19 classification with chest ct**, 2020.

[58] TRAN, D. T.; IOSIFIDIS, A. **Learning to rank: A progressive neural network learning approach**. In: *ICASSP 2019 - 2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, p. 8355–8359, 2019.

[59] UTKIN, L. V. **An imprecise deep forest for classification**. *Expert Systems with Applications*, 141:112978, 2020.

[60] UTKIN, L. V.; KONSTANTINOV, A. V.; CHUKANOV, V. S.; KOTS, M. V.; MELDO, A. A. **An adaptive weighted deep forest classifier**, 2019.

[61] UTKIN, L. V.; KOVALEV, M. S.; MELDO, A. A. **A deep forest classifier with weights of class probability distribution subsets**. *Knowledge-Based Systems*, 173:15 – 27, 2019.

[62] UTKIN, L. V.; RYABININ, M. A. **Discriminative metric learning with deep forest**, 2017.

[63] UTKIN, L. V.; RYABININ, M. A. **A siamese deep forest**. *Knowledge-Based Systems*, 139:13 – 22, 2018.

[64] WANG, H.; TANG, Y.; JIA, Z.; YE, F. **Dense adaptive cascade forest: a self-adaptive deep ensemble for classification problems**. *Soft Computing*, 24(4):2955–2968, May 2019.

[65] WANG, X.; LI, C.; GOLBANDI, N.; BENDERSKY, M.; NAJORK, M. **The lambdaloss framework for ranking metric optimization**. In: *Proceedings of the 27th ACM International Conference on Information and Knowledge Management*, CIKM '18, p. 1313–1322, New York, NY, USA, 2018. Association for Computing Machinery.

[66] WOLPERT, D. H. **Stacked generalization**. *Neural Networks*, 5(2):241 – 259, 1992.

[67] WU, Q.; BURGES, C. J. C.; SVORE, K. M.; GAO, J. **Adapting boosting for information retrieval measures**. *Information Retrieval*, 13(3):254–270, Sept. 2009.

[68] XIA, L.; SHENHAO, T.; RUN, Z.; QIAN, W.; YUANTAO, S. **Research on potential damage estimation of household appliances based on gcforest model**. In: *Proceedings of the 2017 International Conference on Software and E-Business*, ICSEB 2017, p. 97–101, New York, NY, USA, 2017. Association for Computing Machinery.

[69] YIN, L.; SUN, Z.; GAO, F.; LIU, H. **Deep forest regression for short-term load forecasting of power systems**. *IEEE Access*, 8:49090–49099, 2020.

[70] ZHANG, M.; KUANG, D.; HUA, G.; LIU, Y.; MA, S. **Is learning to rank effective for web search?** In: *SIGIR 2009 workshop: Learning to Rank for Information Retrieval*, p. 641–647, 2009.

[71] ZHANG, Y.-L.; ZHOU, J.; ZHENG, W.; FENG, J.; LI, L.; LIU, Z.; LI, M.; ZHANG, Z.; CHEN, C.; LI, X.; QI, Y. A.; ZHOU, Z.-H. **Distributed deep forest and its application to automatic detection of cash-out fraud**. *ACM Trans. Intell. Syst. Technol.*, 10(5), sep 2019.

[72] ZHANG, Z.; KANG, C.; XIONG, G.; LI, Z. **Deep forest with lrrs feature for fine-grained website fingerprinting with encrypted ssl/tls**. In: *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*, CIKM '19, p. 851–860, New York, NY, USA, 2019. Association for Computing Machinery.

[73] ZHOU, M.; ZENG, X.; CHEN, A. **Deep forest hashing for image retrieval**. *Pattern Recognition*, 95:114–127, 2019.

[74] ZHOU, T.; SUN, X.; XIA, X.; LI, B.; CHEN, X. **Improving defect prediction with deep forest**. *Information and Software Technology*, 114:204 – 216, 2019.

[75] ZHOU, Z.-H. **Ensemble Methods: Foundations and Algorithms**. Chapman and Hall/CRC, 1st edition, 2012.

[76] ZHOU, Z.; FENG, J. **Deep forest: Towards an alternative to deep neural networks**. *CoRR*, abs/1702.08835, 2017.

[77] ŁUKASZ KORYCKI.; KRAWCZYK, B. **Adaptive deep forest for online learning from drifting data streams**, 2020.