FACULDADE DE TECNOLOGIA DE AMERICANA "MINISTRO RALPH BIASI"

Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas

Arthur Andreazza Marinho

Sistemas de recomendação de sessão utilizando NLP aplicados à Indústria 4.0

Americana, SP

2023

FACULDADE DE TECNOLOGIA DE AMERICANA "MINISTRO RALPH BIASI"

Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas

Arthur Andreazza Marinho

Sistemas de recomendação de sessão utilizando NLP aplicados à Indústria 4.0

Trabalho de Conclusão de Curso desenvolvido em cumprimento à exigência curricular do Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas, sob a orientação do Prof. Me. Jonas Bodê.

Área de concentração: Tecnologia da Informação

Americana, SP.

2023

FICHA CATALOGRÁFICA — Biblioteca Fatec Americana Ministro Ralph Biasi- CEETEPS
Dados Internacionais de Catalogação-na-fonte

MARINHO, Arthur Andreazza

Sistemas de recomendação de sessão utilizando NLP aplicados à Indústria 4.0. / Arthur Andreazza Marinho — Americana, 2023.

27f.

Monografia (Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas) - - Faculdade de Tecnologia de Americana Ministro Ralph Biasi — Centro Estadual de Educação Tecnológica Paula Souza

Orientador: Prof. Ms. Jonas Bodê

1 . Inteligência artificial. I. MARINHO, Arthur Andreazza II.

BODÊ, Jonas III. Centro Estadual de Educação Tecnológica Paula Souza — Faculdade de Tecnologia de Americana Ministro Ralph Biasi

CDU: 007.52

Elaborada pelo autor por meio de sistema automático gerador de ficha catalográfica da Fatec de Americana Ministro Ralph Biasi.

Arthur Andreazza Marinho

Sistemas de recomendação baseados em sessão utilizando NLP aplicados à Indústria 4.0

Trabalho de graduação apresentado como exigência parcial para obtenção do título de Tecnólogo em Análise e Desenvolvimento de Sistemas pelo CEETEPS/Faculdade de Tecnologia – FATEC/ Americana.

Área de concentração: Tecnologia da Informação

Americana, 29 de novembro de 2023

Prof. Me. Jonas Bodê (Presidente)
FATEC - Americana

Prof. Esp. Antonio Alfredo Lacerda (Membro
FATEC – Americana

Prof. Dr. Cleberson Eugenio Forte (Membro)
FATEC - Americana

AGRADECIMENTOS

Este Trabalho de Conclusão de Curso teve o apoio de muitas pessoas, e, primeiramente, gostaria de agradecer ao meu orientador, o Professor Jonas Bodê, pela atenção, confiança e por aceitar esta empreitada.

Agradeço também ao Professor Marcos de Carvalho Dias pelas sugestões sobre a Indústria 4.0.

Desejo igualmente agradecer aos professores da FATEC-Americana pela paciência, apoio, correções e ensinamentos.

Agradeço também aos funcionários da FATEC-Americana pela paciência, apoio e carinho.

Agradeço aos amigos que de alguma forma fizeram parte desta jornada.

E, por fim, agradeço aos meus familiares, pela ajuda, incentivo e apoio incondicional.

RESUMO

O presente trabalho tem como objetivo identificar e explorar o uso da tecnologia de processamento de linguagem natural (NLP) em sistemas de recomendação baseadas em sessão e sua aplicabilidade à Indústria 4.0. Neste sentido, foi implementado o modelo *Transformers4rec*, baseado em *transformers*, utilizando a plataforma Vertex *AI* da *Google* para realizar os testes. Inicialmente, foi criado um projeto na plataforma e o *kernel Merlin PyTorch* foi utilizado para desenvolver os notebooks de teste. Foi selecionado um subconjunto do conjunto de dados de comércios eletrônicos referente ao mês de outubro de 2019. Foi utilizada uma *GPU NVIDIA Tesla A100* disponibilizada gratuitamente pela plataforma *Vertex AI*. Foram exploradas duas abordagens diferentes: o uso do modelo *GRU* e o uso do modelo *XLNet* com e sem informações adicionais, bem como as métricas *NDCG@10*, *NDCG@20*, *Recall10* e Recall20. Modelos mais avançados, como o *XLNet* com informações adicionais, incluindo metadados de itens e contexto do usuário, pode levar a resultados de recomendação mais precisos e personalizados. Esses modelos podem contribuir para a eficiência e otimização dos processos industriais na Indústria 4.0 e podem nortear futuras pesquisas nessa área, impulsionando ainda mais a eficiência e a otimização dos referidos processos. No entanto, também identificamos desafios associados a essas abordagens, como a utilização de *GPUs* mais potentes para treinamento de modelos de recomendação baseados em sessão que requerem um maior consumo de energia, o que pode ter um impacto negativo na sustentabilidade ambiental, além dos elevados recursos financeiros e infraestrutura. Tais desafios podem representar um obstáculo para a implementação generalizada dessas técnicas, especialmente para empresas com recursos financeiros limitados.

Palavras Chave: Sistemas de recomendação, processamento de linguagem natural, Indústria 4.0

ABSTRACT

The present work aims to identify and explore the use of natural language processing technology (NLP) in session-based recommendation systems and its applicability to Industry 4.0. In this regard, the Transformers4rec model, based on transformers, was implemented using Google's Vertex AI platform to perform the tests. Initially, a project was created on the platform and the Merlin PyTorch kernel was used to develop the test notebook. A subset of the e-commerce data set from the month of October 2019 was selected. It was used an NVIDIA Tesla A100 GPU available for free on Vertex AI platform. Two different approaches were explored: using the GRU model and using the XLNet model with and without additional information, as well as the metrics NDCG@10, NDC G@20, Recall10 and Recall20. More advanced models such as XLNet with additional information, such as item metadata and user context, can lead to more accurate and personalized recommendation results, which can contribute to the efficiency and optimization of industrial processes in Industry 4.0, and can guide future research in this area, further boosting the effectiveness and optimization of Industrial Processes in Industries 4.0. However, we also identified challenges associated with these approaches, such as using more powerful GPUs for training session-based recommendation models that require higher energy consumption, which can have a negative impact on environmental sustainability, in addition to high financial resources and infrastructure. Such challenges may represent an obstacle to the widespread implementation of these techniques, especially for companies with limited financial resources.

Keywords: *Recommendation systems, natural language processing, Industry 4.0*

SUMÁRIO

1 INTRODUÇÃO

A Indústria 4.0 também chamada de Quarta Revolução Industrial emprega um amplo sistema de inovações e tecnologias avançadas como inteligência artificial, robótica, aprendizado de máquina, computação em nuvem, internet das coisas, dentre outros que está mudando as políticas de produção e os negócios industriais no mundo. Nesse cenário, observam-se avanços significativos nas tecnologias de recomendação, impulsionados pelo crescente volume de dados gerados pelos sistemas de informação e pelos avanços em algoritmos de aprendizado de máquina. Dentre esses algoritmos, o modelo *Transformers4rec* tem emergido como uma abordagem promissora para recomendações baseadas em sessão. No entanto, a eficácia e o impacto do uso de conjuntos de dados atualizados e abrangentes no modelo Transformers4rec para recomendações baseadas em sessão e suas possíveis implicações para a Indústria 4.0 ainda precisam ser investigados.

Para atingir o objetivo proposto, este estudo adotou como estratégia metodológica a revisão bibliográfica e a implementação de algoritmos de processamento de linguagem natural em sistemas de recomendação baseados em sessão com o intuito de explorar os possíveis impactos na Indústria 4.0. É necessário compreender os princípios e conceitos-chave da tecnologia de processamento de linguagem natural, bem como as características específicas da Indústria 4.0 que são importantes na implementação de sistemas de recomendação baseados em sessão.

Para tanto, foi utilizada a plataforma da *Google Colab* e *Vertex AI* para reproduzir os resultados da pesquisa apresentada por Moreira *et al.* (2021) que utiliza NLP (“*Natural Language Processing*”) para sistemas de recomendação baseados em sessão que possibilita identificar os desafios, eficácia e limitações na implantação dessa tecnologia, levando em consideração a infraestrutura computacional que tal tecnologia exige. São analisados aspectos como a qualidade e a disponibilidade dos dados, interpretação e compreensão das informações textuais, bem como a capacidade de lidar com a dinamicidade e a complexidade dos ambientes industriais com o intuito de aprimorar as práticas de recomendação na Indústria 4.0 e como elas podem contribuir para a eficiência e a otimização dos processos industriais.

2 FUNDAMENTAÇÃO TEÓRICA

2.1 Indústria 4.0

Apesar de sua importância mundial, o termo Indústria 4.0 ainda carece de uma definição adequada (KLINGENBERG *et al.*, 2019; apud SILTORI *et al.,* 2021). Nos estados Unidos, é comumente conhecida como "*Internet Industry of Things*", manufatura avançada ou fabricação inteligente (GIMENEZ *et al.*, 2019).

A manufatura avançada ou indústria 4.0 tem como base a utilização de tecnologias chaves, por exemplo: Internet Industrial das Coisas (*IIoT*), Realidade Aumentada, aplicativos de tecnologias de nuvem, robôs autônomos (ARBIX *et al.*, 2017). É caracterizada por uma nova era de automação, crescimento da conectividade de sistemas e redes, avanço da internet, avanços da sensorização, capacidade de acumulação e processamento de dados, bem como avanços na área de robótica (GIMENEZ *et al.,* 2019). Trata-se de um conjunto de tecnologias digitais e físicas que oferecem novos valores e serviços a clientes e organizações. Representa transformações abrangentes na produção industrial por meio de tecnologias emergentes como a introdução da internet e avanços em inteligência artificial, gerando uma capacidade sem precedentes de processar grandes quantidades de dados e acelerando inovações. Tal Indústria pode ser descrita como um movimento que digitaliza e automatiza processos na área de manufatura, de forma sequencial, conectando plataformas por meio dessa digitalização (DALENOGARE *et al*., 2018; FRAZZON *et al*., 2019; REJIKUMAR *et al*., 2019; SUNG, 2018; apud SILTORI *et al*.,2021).

Cabe ressaltar que o termo Indústria 4.0 foi introduzido pela primeira vez na Alemanha, em 2011, durante uma das mais importantes feiras de tecnologia do mundo, a Hannover Messe (SILTORI *et al*., 2021). Em um sentido amplo, a manufatura herdeira do século XX consiste na fabricação e montagem de produtos através de processos lineares. Na manufatura avançada, os novos materiais deixam de ser somente insumos e passam a ser parte integrante dos processos de fabricação porque ao serem conectados passam a ser emissores e receptores de dados. Nesse cenário, as fronteiras entre fabricação e montagem tendem a desaparecer graças aos processos de automação e os processos de reaproveitamento de rejeitos industriais, apontando para uma nova dimensão da reciclagem, prolongando o ciclo de vida dos produtos (ARBIX *et al*., 2017).

Seis princípios definem as características da Indústria 4.0: interoperabilidade, virtualização, descentralização, capacidade em tempo real, orientação a serviços e modularidade (SILTORI *et al.*, 2021). O conceito de interoperabilidade visa facilitar a comunicação entre sistemas, sendo recomendados padrões abertos para realizar essa tarefa. Por meio da virtualização, os sistemas físicos podem ser replicados na nuvem, possibilitando simulações em tempo real. A descentralização é a troca constante de informações que permite aos sistemas ciber-físicos tomar decisões em tempo real sem interferência humana. A capacidade em tempo real consiste em coletar informações instantaneamente, facilitando decisões rápidas e ágeis. A orientação a serviços consiste em serviços de customização de software de acordo com as necessidades de cada organização. Finalmente, a modularidade é caracterizada pela flexibilidade de todo o processo de produção que permite o rearranjo das linhas de produção por meio de módulos de acoplamento e desacoplamento.

Com base nesses seis princípios, são definidos os seguintes pilares como conceitos da Indústria 4.0 que podem ser adotados pelas empresas: Sistemas Ciber-Físicos (CPS), Internet das Coisas (IoT), Internet de Serviços (IoS), Veículos Autônomos, Impressoras 3D, Robôs Avançados, Inteligência Artificial, “Big Data”, “Cloud Computing”, Realidade Virtual e Aumentada, Nanomateriais e Nanosensores, entre outros (SILTORI *et al*., 2021)

Em suma, a Indústria 4.0 representa a otimização de componentes envolvidos no processo de produção, reduzindo seus custos, particularmente na área de planejamento de pessoal, permitindo à empresa um posicionamento melhor na competição internacional (WISSKIRCHEN *et al*., 2017).

2.2 Sistemas de recomendação baseados em sessões

Sistemas de recomendação baseados em sessões (*SBRS*) são um tópico emergente no campo dos sistemas de recomendar que têm atraído atenção tanto da academia quanto da indústria (WANG *et al*., 2019). São um tipo de sistema de recomendação que fornece recomendações com base nas interações de um usuário dentro de uma sessão em curso, sem depender de perfis de usuário de longo prazo ou preferências históricas (WANG *et al*., 2019). Esses sistemas ganharam popularidade nos últimos anos devido à sua capacidade de funcionar mesmo na ausência de dados históricos do usuário ou disponibilidade de conjuntos de “*datasets*”

centrados em sessões, bem como devido às técnicas de "*deep learning*" adequadas para sequências.

O objetivo principal dos *SBRSs* é capturar preferências de usuário dinâmicas a curto prazo e fornece recomendações oportunas e precisas que sejam sensíveis à evolução de seus contextos de sessão (WANG *et al*. 2019). Isto é particularmente útil em situações em que as informações de longo prazo não estão disponíveis, principalmente quando os usuários não estão conectados ou são usuários pela primeira vez (LUDEWING *et al*., 2021).

Ao contrário dos sistemas de recomendação baseados em conteúdo e de filtragem colaborativa, que geralmente modelam preferências de usuário de longo prazo, porém estáticas, os *SBRSs* visam capturar preferências do usuário a curto prazo. São dinâmicas para fornecer recomendações mais oportunas e precisas, as quais são sensíveis à evolução de seus contextos de sessão (WANG *et al.*, 2019).

Uma variedade de algoritmos foi proposta para problemas de recomendação baseados em sessão, incluindo esquemas de vizinhos mais próximos, fatoração de matriz e métodos baseados em "*deep learning*" (WANG *et al*. 2019).

Existem vários desafios associados ao *SBRS*, como a heterogeneidade do item, as relações de acoplamento de características dentro e entre itens, e a modelagem de dependências de nível de item para tarefas de recomendação. A pesquisa nessa área está em desenvolvimento com o intuito de enfrentar esses desafios e melhorar o desempenho dos sistemas de recomendação baseados em sessões.

2.3 *NLP*

O "*Natural Language Processing*" (*NLP*) é um subcampo da inteligência artificial que se concentra em permitir que computadores percebam, interpretem e gerem linguagem humana. Suas técnicas permitem que os computadores processem a entrada de texto e áudio, avaliem sua estrutura e significado e gerem respostas linguísticas semelhantes às humanas. Devido às suas inúmeras aplicações, incluindo tradução automática, análise de sentimentos, *chatbots* e extração de informações, entre outros, a *NLP* atraiu muita atenção nos últimos anos (KHURANA *et al.,* 2022)

Os avanços no processamento de linguagem natural (*NLP*) impulsionados por técnicas de "*deep learning*" tiveram um impacto significativo nos "*Recommender Systems* (*RecSys*)", especialmente nas recomendações sequenciais e baseadas em

sessões. Os primeiros modelos de linguagem neural se concentraram em agrupar palavras com sintaxe e significado semelhantes no espaço vetorial. Esses métodos foram posteriormente estendidos para representar frases e parágrafos. Em *RecSys*, essas abordagens neurais foram adaptadas para aprender "*embeddings*" para itens, usuários ou contextos, considerando sua concorrência nas interações do usuário.

Por exemplo, *Prod2Vec* e *Meta Prod2Veci* usaram o modelo de "*skip-gram*" do *Word2VeC* para aprender representações de produtos, incorporando metadados de itens no último. Essas inserções pré-treinadas foram empregadas para recomendar produtos similares. *RecSys* também utilizou Redes Neurais Recorrentes (*RNNs*) para aproveitar as interações de itens sequenciais em sessões. A GRU4REC usou um modelo *GRU* para prever o próximo clique, abordando a escalabilidade através de diferentes funções de perda em pares de classificação.

Após o *GRU4REC*, os pesquisadores exploraram a incorporação de recursos adicionais além de *IDs* de itens em modelos baseados em *RNN*, empregando a ampliação de dados e contabilização para mudanças de distribuição de dados. Os mecanismos de atenção, introduzidos em 2016, reformularam as arquiteturas de NLP e foram adotados no *RecSys*. Modelos como a "*Neural Attentive Recommendation Machine* (*NARM*)" adicionaram camadas de atenção aos *RNNs* para capturar o comportamento do usuário e o propósito da sessão, enquanto a "*Attentional FM*" usou redes de atenção para a importância da interação de recursos em modelos de máquinas fatorizadas não sequenciais.

2.4 "*Hugging face transformers*"

"*Hugging face transformers*" ou somente *Transformer* é uma biblioteca de código aberto que fornece modelos de processamento de linguagem natural de última geração, incluindo modelos pré-treinados (aprendizagem de máquina) e capacidades de ajuste fino, para uma ampla gama de tarefas. A biblioteca é construída sobre o *PyTorch* e o *TensorFlow*. O principal objetivo do "*Hugging face transformers*" é tornar a pesquisa de ponta em *NLP* acessível e fácil de usar pela comunidade (WOLF *et al*., 2019).

A biblioteca de código aberto é composta por arquiteturas *Transformer* de última geração cuidadosamente projetadas sob uma API unificada. O suporte para esta biblioteca é uma coleção acurada de modelos pré-treinados e disponíveis para a

comunidade. A arquitetura *Transformer* é uma arquitetura de rede simples, inicialmente proposta para tarefas de tradução automática, revolucionou o campo da aprendizagem profunda, introduzindo o conceito de mecanismos de auto-atenção, dispensando recorrência e convoluções internas. Essa arquitetura facilita a construção de modelos de maior capacidade e o pré-treinamento, permitindo utilizar eficazmente essa capacidade para uma ampla variedade de tarefas. Em suma, Transformers é projetado para ser utilizado por pesquisadores, simples para praticantes e rápido e robusto em implantações industriais (WOLF *et al.*, 2019).

O *XLNet* é um modelo de linguagem neural que foi proposto em 2019 por (Yang *et al*., 2019). Ele é baseado em uma arquitetura de *Transformer* e usa uma técnica de permutação de palavras para lidar com a limitação de independência de palavras em modelos de linguagem anteriores, como o *BERT*. O *XLNet* com informações adicionais é uma versão do *XLNet* que incorpora informações adicionais, como imagens e áudio, para melhorar o desempenho do modelo em tarefas de processamento de linguagem natural.

2.5 *Transformer4rec*

Transformers4rec é um modelo de sistema de recomendação baseado na arquitetura *Transformer*, originalmente introduzido para tarefas de processamento de linguagem natural. Ele é projetado especificamente para lidar com os desafios de recomendações baseadas em sessão, onde o comportamento e as preferências do usuário mudam ao longo do tempo dentro de uma sessão ou sequência de interações. Ao contrário dos modelos de recomendação tradicionais que dependem de modelos sequenciais, como redes neurais recorrentes (RNNs) ou cadeias de Markov, *transformers4rec* centra-se no mecanismo de auto-atenção, que permite ao modelo verificar a importância de diferentes itens dentro de uma sessão de modo dinâmico. Este mecanismo permite que o modelo considere todo o contexto da sessão em vez de depender exclusivamente das janelas do passado imediato ou de comprimento fixo. Ao participar de itens relevantes na sessão, *transformers4rec* pode capturar dependências de longo prazo e modelar efetivamente as preferências do usuário (MOREIRA *et al*., 2021).

O modelo *transformers4rec* consiste em um codificador e um decodificador. O codificador toma a sequência de sessão como entrada e a processa através de várias

camadas de auto-atenção, permitindo que o modelo aprenda o contexto da sessão. O decodificador então gera recomendações com base na representação da sessão aprendida.

Para treinar o modelo *transformers4rec*, várias técnicas de otimização podem ser empregadas, como modelagem de linguagem mascarada, onde um subconjunto de itens na sessão é mascarado, e o modelo é treinado para prever os itens que faltam. Além disso, a "amostragem negativa" pode ser usada para criar exemplos negativos para treinamento para abordar a natureza desequilibrada dos conjuntos de dados de recomendação (MOREIRA *et al.*, 2021).

As vantagens do transformers4rec estão na sua capacidade de capturar padrões complexos e dependências em dados de sessão, lidar com dependências de longo prazo de forma eficaz e gerar recomendações precisas e diversificadas. Seu mecanismo de atenção permite a interpretabilidade, permitindo que os pesquisadores analisem quais itens são mais influentes no processo de recomendação (MOREIRA *et al*., 2021).

Gerenciar problemas de "*cold-start*" é um desafio bem conhecido em sistemas de recomendação. "*Cold-start*" refere-se à questão de fornecer recomendações para novos usuários ou itens com pouco ou nenhum dado de interação. Portanto, o Transformers4Rec não é inerentemente ruim em lidar com "*Cold-start*", mas seu desempenho depende da arquitetura do modelo especificado e das características e estratégias de treinamento empregadas. Ao incorporar recursos de usuário/item, usando modelos híbridos, pré-treinamento e aprendizagem ativa, pode-se melhorar o desempenho do modelo (e.g. https://github.com/NVIDIA-Merlin/Transformers4Rec).

Em suma, transformers4rec é um modelo de sistema de recomendação de última geração baseado na arquitetura *Transformer*. Ele aproveita os mecanismos de auto-atenção para capturar o contexto da sessão, permitindo-lhe gerar recomendações precisas e diversas para cenários baseados em sessão. Seu impacto no campo dos sistemas de recomendação, incluindo suas implicações à Indústria 4.0, é uma área importante de pesquisa.

3 IMPLEMENTAÇÕES

Neste trabalho foi implementado o modelo *Transformers4rec*, baseado em *transformers*, utilizando a plataforma *Vertex AI* da *Google* para realizar os testes. Para

guiar a implementação, foram utilizados os tutoriais disponíveis no GitHub do Transformers4Rec (Anexo).

Primeiramente, um projeto foi criado na plataforma e o *kernel Merlin PyTorch* foi utilizado para desenvolver os *notebooks* de teste. Foi selecionado um subconjunto do conjunto de dados de comércio eletrônico referente ao mês de outubro de 2019 de vários locais, os quais podem ser encontrados em https://www.kaggle.com/mkechinov/ecommerce-behavior-data-from-multi-category-store. O conjunto de dados utilizado tem aproximadamente 6 GB de tamanho, sendo assim, foi necessário utilizar uma GPU NVIDIA Tesla A100 disponibilizada gratuitamente pela plataforma *Vertex AI*.

Foram desenvolvidos três notebooks com diferentes propósitos: O primeiro notebook trata do pré-processamento do conjunto de dados. O segundo notebook utiliza a biblioteca *Rapids cuDF* e a biblioteca *Merlin NVTabular* para realizar o pré-processamento e a engenharia de recursos, preparando assim o conjunto de dados para o treinamento de um modelo de recomendação baseado em sessão utilizando a abordagem de *Pipe*. O terceiro *notebook* tem como foco principal acelerar o carregamento de dados de arquivos de parquet com múltiplos recursos no *PyTorch*, empregando a biblioteca *NVTabular*. Nesse último notebook, também é abordada a formação e avaliação de um modelo de recomendação baseado em redes neurais recorrentes (RNN) "*Gated recurrent unit* (*GRU*)", bem como o treinamento e avaliação de uma arquitetura *Transformer XLNET* para um modelo de recomendação baseado em sessão. Além disso, é explorada a integração de informações laterais (funções adicionais) nas arquiteturas de transformadores com o intuito de aprimorar a precisão das recomendações. Todas usando a biblioteca do *Transformers4rec*.

3.1 Etapas

A implementação do modelo *Transformers4rec*, baseado em *transformers*, envolveu as seguintes etapas:

1. Categorização da coluna "*user_session*": Embora a coluna "*user_session*" não seja utilizada como recurso de entrada para o modelo, ela foi convertida em valores inteiros para evitar possíveis falhas ao agrupar interações pelo "*user_session*" em um notebook subsequente.

2. Remoção de interações consecutivas repetidas, por exemplo, usuário, item: Foram mantidas as interações repetidas nos mesmos itens, removendo apenas as interações consecutivas, porque isso poderia ser devido às atualizações da guia do navegador ou devido aos diferentes tipos de interação, por exemplo, clique, adicionar ao carrinho, compra.

3. Inclusão do recurso "*item first time seen*" para cálculo de recenticidade: Foi criada a coluna "*prod_first_event_time_ts*", que indica o "*timestamp*" em que um item foi visto pela primeira vez. Somente a primeira semana do conjunto de dados de outubro de 2019 foi utilizada. A coluna "*event_time_dt*" foi removida, porque não seria mais utilizada. Os dados foram salvos como um único arquivo parquet para uso posterior.

4. Leitura do arquivo parquet de entrada: Mesmo que o conjunto de dados original contenha arquivos de dados de 7 meses, apenas os primeiros sete dias do conjunto de dados de comércio eletrônico de outubro de 2019 foram utilizados. Tendo sido realizadas etapas de pré-processamento no primeiro mês (outubro de 2019) do conjunto de dados bruto no primeiro notebook. Algumas das etapas realizadas incluem a criação da coluna "*event_time_ts*" a partir da coluna "*event_time*", mostrando o momento em que o evento ocorreu em UTC, criação da coluna "*prod_first_event_time_ts*" que indica o *timestamp* em que um item foi visto pela primeira vez, remoção das linhas em que "*user_session*" é nulo, categorização da coluna "*user_session*" e remoção de interações repetidas consecutivamente.

5. Extração de recursos temporais: Nesta etapa, foram criados recursos relacionados ao tempo, como o dia da semana da coluna "*session_time*", utilizando as funções seno e cosseno para representar o dia da semana como um recurso cíclico contínuo. Também foi criado o recurso de *recência* do produto, utilizando uma operação personalizada para calcular essa *recência* dos itens em dias.

6. Normalização de recursos contínuos: Nesta etapa, os recursos contínuos foram normalizados, utilizando uma operação de suavização para a distribuição de preço e aplicando uma padronização.

7. Agrupamento de interações em sessões: As interações são agrupadas por sessão usando a coluna *'user_session*' como chave de agrupamento. Várias informações sobre as sessões são agregadas, incluindo a lista de *IDs* (*ID* por extenso) de produtos (*'product_id-list'*), lista de categorias (*'category_code-list'*), lista de marcas (*'brand-list'*), lista de *IDs* de categoria (*'category_id-list'*) e outros recursos relacionados

ao tempo e preço. O objetivo é criar um formato sequencial para cada sessão, onde as interações são ordenadas pelo tempo.

8. Seleção de colunas para treinamento do modelo: As colunas selecionadas para treinamento do modelo incluem o ID da sessão (*'user_session'*), o número de interações na sessão (*'product_id-count'*) e os recursos sequenciais das interações (*'product_id-list', 'category_code-list', 'brand-list'*, etc.). Sessões com menos de 2 interações são removidas.

9. Inicialização do objeto *Dataset* e *Workflow* do *NVTabular*: O objeto *Dataset* do *NVTabular* é inicializado com os dados filtrados e o *Workflow* do *NVTabular* é criado para realizar o pré-processamento e a engenharia de recursos. O *Workflow* é um grafo direcionado de operadores que define a sequência de transformações a serem aplicadas aos dados.

10. Treinamento do modelo: Nessa etapa, são treinados diferentes modelos de recomendação baseados em sessões, com o objetivo de comparar seu desempenho. Os modelos utilizados são um modelo baseado em RNN (*GRU*) e uma arquitetura *Transformer* (*XLNET*) para recomendação baseada em sessões. Além disso, também é considerada a integração de informações adicionais ("*side information*") nas arquiteturas *Transformer* para melhorar a precisão da recomendação.

11. Avaliação do modelo: Após cada treinamento, os modelos são avaliados em relação a métricas de avaliação, como *NDCG@20* e *Recall*. Essas métricas são usadas para medir a qualidade e a relevância das recomendações geradas pelos modelos.

12. Comparação dos modelos: Ao final do treinamento e avaliação de ambos os modelos (*GRU* e *XLNet*), os resultados das métricas de avaliação (*NDCG@20* e *Recall*) são registrados em um arquivo de texto chamado "*results.txt*". Esses resultados permitem comparar o desempenho dos dois modelos e identificar qual deles obteve melhor resultados na tarefa de recomendação baseada em sessões.

3.2 Métricas

As métricas utilizadas foram o *NDCG@10*, *NDCG@20*, *Recall 10* e *Recall 20*.

NDCG@10 ("*Normalized Discounted Cumulative Gain at 10*") e *NDCG@20* ("*Normalized Discounted Cumulative Gain at 20*") são métricas de avaliação usadas

para medir a qualidade de um sistema de recomendação em relação aos primeiros 10 e 20 itens recomendados, respectivamente.

O *NDCG@10* e o *NDCG@20* levam em consideração a relevância dos itens recomendados, bem como a posição em que eles são apresentados na lista de recomendações. A métrica normaliza o ganho cumulativo dos itens, com um desconto aplicado a itens menos relevantes e um aumento de peso para itens mais relevantes.

O cálculo do NDCG envolve os seguintes passos:

- Para cada usuário, o sistema de recomendação gera uma lista de itens recomendados, respectivamente, classificados em ordem de relevância estimada
- A relevância de cada item é atribuída com base em algum critério pré-definido, como avaliações de usuários ou dados de interação do usuário.
- O ganho acumulado de cada item é calculado levando em consideração sua posição na lista de recomendações e sua relevância. Itens mais relevantes em posições superiores têm um ganho acumulado maior.
- O ganho acumulado é então normalizado dividindo-o pelo ganho acumulado ideal, que é calculado ordenando todos os itens de acordo com sua relevância real e calculando o ganho acumulado nessa ordem.
- As métricas *NDCG@10* e *NDCG@20* são representadas pela média dos NDCGs individuais de todos os usuários.

Uma pontuação de *NDCG@10* e *NDCG@20* mais alta indica que o sistema de recomendação está fornecendo recomendações mais relevantes e úteis nos primeiros 10 e 20 itens, respectivamente. São medidas amplamente usadas para avaliar e comparar sistemas de recomendação em diferentes domínios e cenários de aplicação.

Recall10 e *Recall20* são métricas de avaliação comumente usada para medir a capacidade de um sistema de recomendação em recuperar corretamente itens relevantes em relação aos primeiros 10 e 20 itens recomendados, respectivamente.

Em um sistema de recomendação, o *Recall* é calculado considerando os itens relevantes em um conjunto de dados de referência. Geralmente, esses itens relevantes são determinados com base em "*feedbacks*" dos usuários, como avaliações ou interações passadas.

O cálculo do *Recall* envolve os seguintes passos:

- Para cada usuário, o sistema de recomendação gera uma lista de itens recomendados.

- A lista de itens recomendados é comparada com o conjunto de itens relevantes que é conhecido a partir do conjunto de dados de referência.
- O *Recall* é calculado como a proporção entre o número de itens relevantes que foram recomendados corretamente e o número total de itens relevantes existentes.
- Por exemplo, se existem 100 itens relevantes no conjunto de dados de referência e se o sistema de recomendação recomendar 50 desses itens, o *Recall* será de 50%.

4 RESULTADOS

Neste trabalho foram implementados e realizados testes de treinamento de modelos de recomendação baseados em sessão utilizando a arquitetura do *Transformers4Rec*. Para efetuar a implementação, foram utilizados os tutoriais disponíveis no GitHub do *Transformers4Rec* (Anexo).

Para realizar o teste, foi selecionado um conjunto de dados de comércio eletrônico referente ao mês de outubro de 2019 de vários locais, os quais podem ser encontrados em (https://www.kaggle.com/mkechinov/ecommerce-behavior-data-from-multi-category-store). O conjunto de dados selecionado tem aproximadamente 6 GB de tamanho, desse modo, foi necessário utilizar uma *GPU NVIDIA Tesla A100* disponibilizada gratuitamente pela plataforma *Vertex AI* da *Google*. Ainda, foi realizado o treinamento e a avaliação do modelo ao longo de várias janelas de tempo.

Primeiramente, um projeto foi criado na plataforma e o *kernel Merlin PyTorch* foi utilizado para desenvolver os notebooks de teste. Foram exploradas duas abordagens diferentes: o uso do modelo *GRU* e do modelo *XLNet* com e sem informações adicionais.

No caso do modelo GRU, foi utilizada a biblioteca *transformers4rec*.torch para definir a arquitetura do modelo e realizar o treinamento. Foi demonstrado nas seções anteriores como conectar os blocos de construção, como a entrada de dados, a camada *GRU* e a camada de previsão de itens futuros.

Foi também explorada a arquitetura do modelo *XLNet* para realizar a tarefa de previsão do próximo item com a técnica de máscara de linguagem (MLM). Novamente, foi usada a biblioteca transformers4rec.torch para definir a arquitetura do modelo, conectar os blocos de construção e realizar o treinamento e a avaliação. Também foi

mostrado como acrescentar informações adicionais, como metadados de itens e contexto do usuário, ao modelo *XLNet* para melhorar a qualidade das recomendações.

As métricas obtidas a partir do modelo GRU e do modelo *XLNet* com e sem informações adicionais podem ser encontrados na Tabela 1.

Tabela 1 – Precisões referentes às métricas dos modelos GRU e XLNet-MLM

Métricas	Precisão *GRU*	Precisão *XLNet-MLM*	Precisão do *XLNet-MLM* com informações adicionais
NDCG@10	0,0674505457	0,074229903	0,0830973908
NDCG@20	0,0817371383	0,090177714	0,1009939163
Recall@10	0,1259984523	0,140685394	0,1542772501
Recall@20	0,1826204657	0,203877866	0,2251996994

Fonte: Próprio autor

Para os modelos testados, pode-se observar que a métrica *NDCG@20* usada para medir a qualidade de um sistema de recomendação em relação aos primeiros 20 itens recomendados é mais precisa que a *NDCG@10*, ou seja, em relação aos primeiros 10 itens recomendados. A mesma tendência é observada entre as métricas *Recall@10* e *Recall@20*. Porém, a métrica *Recall20* foi a que apresentou maior precisão para todos os modelos.

Quanto aos modelos testados, é possível observar que o modelo *XLNet-MLM* com informações adicionais foi o mais preciso. Ainda, pode-se observar que, em todas as métricas avaliadas, o modelo *XLNet-MLM* apresentou um desempenho superior em comparação ao modelo *GRU* que utiliza redes neurais recorrentes (RNN). Isso indica que a técnica de máscara de linguagem (MLM) utilizada pelo XLNet proporcionou resultados mais precisos e personalizados nas recomendações.

Ao adicionar informações adicionais ao modelo *XLNet-MLM*, observa-se um aumento geral na acurácia em relação ao modelo sem informações adicionais. Nesse caso, a inclusão de metadados de itens e contexto do usuário pode melhorar significativamente a qualidade das recomendações, tornando-as mais relevantes e úteis para os usuários.

Cabe ressaltar que resultados similares foram obtidos por Moreira *et al.* (2021).

5 CONSIDERAÇÕES FINAIS

Neste trabalho, foram realizados testes de treinamento de modelos de recomendação baseados em sessão utilizando a arquitetura do *Transformers4Rec*, com o objetivo de identificar e explorar o uso da tecnologia de processamento de linguagem natural (*NLP*) em sistemas de recomendação baseadas em sessão na Indústria 4.0. Ao longo dos testes, foram exploradas duas abordagens diferentes: o uso do modelo *GRU* e o uso do modelo *XLNet* com e sem informações adicionais, bem como as métricas *NDCG@10*, *NDCG@20*, *Recall10* e *Recall20*.

Foi observado que o uso de modelos mais avançados, como o *XLNet*, pode levar a resultados de recomendação mais precisos e personalizados, o que pode contribuir para a eficiência e otimização dos processos industriais na Indústria 4.0. No entanto, também identificamos desafios associados a essas abordagens. A necessidade de *GPUs* mais poderosas e caras para treinar esses modelos pode representar um obstáculo para a implementação generalizada dessas técnicas, especialmente para empresas com recursos financeiros limitados. Além disso, a utilização de *GPUs* mais potentes para treinamento de modelos de recomendação baseados em sessão requer um maior consumo de energia, o que pode ter um impacto negativo na sustentabilidade ambiental. Essa demanda energética pode aumentar, ainda mais, o monopólio das grandes empresas, como o Google, que possuem os recursos financeiros e infraestrutura necessários para suportar essa demanda, ampliando a integração de informações e *IoT* em seus sistemas de recomendação.

Portanto, ao considerar a implementação dessas abordagens avançadas de recomendação baseada em sessão na Indústria 4.0, é importante ponderar cuidadosamente os custos e benefícios, levando em conta questões financeiras, acessibilidade, sustentabilidade e equidade. A inclusão de informações adicionais, como metadados de itens e contexto do usuário, pode melhorar significativamente a qualidade das recomendações, mas também requer considerações adicionais em termos de privacidade e ética na Indústria 4.0.

No geral, os testes realizados forneceram “insights” valiosos sobre as técnicas de recomendação baseada em sessão com o uso de modelos avançados, bem como os desafios e impactos sociais associados a essas abordagens na Indústria 4.0. Essas informações podem orientar a tomada de decisões futuras ao implementar sistemas

de recomendação mais sofisticados, garantindo que sejam inclusivos, sustentáveis e éticos. Além disso, contribuem para a compreensão e aprimoramento das práticas de recomendação aplicadas à Indústria 4.0 e podem nortear futuras pesquisas nessa área, impulsionando ainda mais a eficiência e a otimização dos processos industriais.

REFERÊNCIAS BIBLIOGRÁFICAS

ARBIX G., SALERNO M. S., ZANCUL E., AMARAL G., LINS L., **O Brasil e a nova onda de manufatura avançada:** o que aprender com a Alemanha, China e Estados Unidos. NOVOS ESTUDOS – CEBRAP., São Paulo, v. 36.03, p. 29-49, 2017. Disponível em: https://doi.org/10.25091/S0101-3300201700030003

GIMENEZ D. M., dos SANTOS, A. L**., Indústria 4.0, manufatura avançada e seus impactos sobre o trabalho**, Instituto de Economia – Unicamp, 2019. Disponível em: https://www.eco.unicamp.br/images/arquivos/artigos/TD/TD371.pdf.

KHURANA D., KOLI A., KHATTER K., SINGH S., ***Natural Language Processing:*** *State of The Art, Current Trends and Challenges*. Multimedia Tools and Applications, v. 82, p. 3713–3744, 2023. Disponível em: https://doi.org/10.1007/s11042-022-13428-4.

LUDEWING M., *et al.* ***Empirical analysis of session-based recommendation algorithms***. User Model User-Adap Inter, v. 31, p. 149–181, 2021.

MOREIRA G. P. S., RABHI S., LEE J. M., AK R., OLDRIDGE E., ***Transformers4Rec:*** *Bridging the Gap between NLP and Sequential/Session-Based Recommendation*. In Proceedings of the 15th ACM Conference on Recommender Systems (RecSys '21). Association for Computing Machinery, New York, NY, USA, p. 143–153, 2021. Disponível em: https://doi.org/10.1145/3460231.3474255

SILTORI P. F. S., ANHOLON R., RAMPASSO I. S., QUELHAS O. L. G., SANTA-EULALIA L., A., LEAL FILHO W., ***Industry 4.0 corparate sustaninability an exploratory analysis of possible impacts in the Brazilian context***, Technological Forecasting and Social Change, 167. p. 120741, 2021. Disponível em: https://doi.org/10.1016/j.techfore.2021.120741

WANG S., CAO L., WANG Y., SHENG Y. W., SHENG Q. Z., ORGUN M., LIAN D., ***Survey on Session-based Recommender Systems***, ACM Computing Surveys, 9, 1-38, 2021. Disponível em: <arXiv:1902.04864>.

WISSKIRCHEN G., BIACABE B. T., BORMANN U., MUTZ A., NIEHAUS G., SOLER G. J., VON BRAUCHITSC B., ***Artificial Intelligence and Robotics and Their Impact on the Workplace***, IBA Global Employment Institute, 2017. Disponível em: http://www.innovation4.cn/library/r15268

WOLF T., *et al*., ***Hugging Face's Transformers: State-of-the-art Natural Language Processing***, 2020. Disponível em: ArXiv abs/1910.03771

Yang, Z., *et al*. ***XLNet: Generalized Autoregressive Pretraining for Language Understanding***. (2019). Disponível em: ArXiv:1906.08237.

ANEXO

Código modificado para uso nos notebooks na plataforma, https://github.com/NVIDIA-Merlin/Transformers4Rec

```
import os
import glob
import torch
import transformers4rec.torch as tr

from transformers4rec.torch.ranking_metric import NDCGAt, RecallAt
from transformers4rec.torch.utils.examples_utils import wipe_memory

from merlin.schema import Schema
from merlin.io import Dataset

INPUT_DATA_DIR = os.environ.get("INPUT_DATA_DIR", "/workspace/data")

train = Dataset(os.path.join(INPUT_DATA_DIR, "processed_nvt/part_0.parquet"))
schema = train.schema
schema = schema.select_by_name(['product_id-list'])

sequence_length = 20
inputs = tr.TabularSequenceFeatures.from_schema(
        schema,
        max_sequence_length= sequence_length,
        masking = 'causal',
    )

d_model = 128
body = tr.SequentialBlock(
        inputs,
        tr.MLPBlock([d_model]),
```

```
        tr.Block(torch.nn.GRU(input_size=d_model,                hidden_size=d_model,
num_layers=1), [None, 20, d_model])
)



head = tr.Head(
    body,
    tr.NextItemPredictionTask(weight_tying=True,
                              metrics=[NDCGAt(top_ks=[10, 20], labels_onehot=True),
                                       RecallAt(top_ks=[10, 20], labels_onehot=True)]),
)
model = tr.Model(head)

from transformers4rec.torch.utils.data_utils import MerlinDataLoader
x_cat_names, x_cont_names = ['product_id-list_seq'], []

sparse_features_max = {
    fname: sequence_length
    for fname in x_cat_names + x_cont_names
}

def get_dataloader(data_path, batch_size=128):
        loader = MerlinDataLoader.from_schema(
            schema,
            data_path,
            batch_size,
            max_sequence_length=sequence_length,
            shuffle=False,
        )
        return loader
```

```
Set training arguments
from transformers4rec.config.trainer import T4RecTrainingArguments
from transformers4rec.torch import Trainer

train_args = T4RecTrainingArguments(local_rank = -1,
                                    dataloader_drop_last = False,
                                    report_to = [],    #set empty list to avoid logging metrics to Weights&Biases
                                    gradient_accumulation_steps = 1,
                                    per_device_train_batch_size = 256,
                                    per_device_eval_batch_size = 32,
                                    output_dir = "./tmp",
                                    max_sequence_length=sequence_length,
                                    learning_rate=0.00071,
                                    num_train_epochs=3,
                                    logging_steps=200,
                                   )

trainer = Trainer(
    model=model,
    args=train_args,
    schema=schema,
    compute_metrics=True,
)


OUTPUT_DIR = os.environ.get("OUTPUT_DIR", "/workspace/data/sessions_by_day")




%%time
```

```
start_time_window_index = 1
final_time_window_index = 4
for time_index in range(start_time_window_index, final_time_window_index):

    time_index_train = time_index
    time_index_eval = time_index + 1
    train_paths = glob.glob(os.path.join(OUTPUT_DIR,
f"{time_index_train}/train.parquet"))
    eval_paths = glob.glob(os.path.join(OUTPUT_DIR,
f"{time_index_eval}/valid.parquet"))


    trainer.train_dataloader = get_dataloader(train_paths,
train_args.per_device_train_batch_size)
    trainer.eval_dataloader = get_dataloader(eval_paths,
train_args.per_device_eval_batch_size)


    print('*'*20)
    print("Launch training for day %s are:" %time_index)
    print('*'*20 + '\n')
    trainer.reset_lr_scheduler()
    trainer.train()
    trainer.state.global_step +=1


    train_metrics = trainer.evaluate(metric_key_prefix='eval')
    print('*'*20)
    print("Eval results for day %s are:\t" %time_index_eval)
    print('\n' + '*'*20 + '\n')
    for key in sorted(train_metrics.keys()):
        print(" %s = %s" % (key, str(train_metrics[key])))
    wipe_memory()
```

```
with open("results.txt", 'w') as f:
    f.write('GRU accuracy results:')
    f.write('\n')
    for key, value in  model.compute_metrics().items():
        f.write('%s:%s\n' % (key, value.item()))
import os
import glob

import torch
import transformers4rec.torch as tr
from transformers4rec.torch.ranking_metric import NDCGAt, RecallAt
from transformers4rec.torch.utils.examples_utils import wipe_memory

from merlin.schema import Schema
from merlin.io import Dataset

INPUT_DATA_DIR = os.environ.get("INPUT_DATA_DIR", "/workspace/data")

train = Dataset(os.path.join(INPUT_DATA_DIR, "processed_nvt/part_0.parquet"))
schema = train.schema
schema = schema.select_by_name(['product_id-list'])

sequence_length, d_model = 20, 192
inputs= tr.TabularSequenceFeatures.from_schema(
    schema,
    max_sequence_length=sequence_length,
    d_output=d_model,
    masking="mlm",
)

transformer_config = tr.XLNetConfig.build(
    d_model=d_model, n_head=4, n_layer=2, total_seq_length=sequence_length
)
```

```
body = tr.SequentialBlock(
   inputs,          tr.MLPBlock([192]),          tr.TransformerBlock(transformer_config,
masking=inputs.masking)
)

head = tr.Head(
   body,
   tr.NextItemPredictionTask(weight_tying=True,
                        metrics=[NDCGAt(top_ks=[10, 20], labels_onehot=True),
                              RecallAt(top_ks=[10, 20], labels_onehot=True)]),
)

model = tr.Model(head)



from transformers4rec.config.trainer import T4RecTrainingArguments
from transformers4rec.torch import Trainer

training_args = T4RecTrainingArguments(
          output_dir="./tmp",
          max_sequence_length=20,
          data_loader_engine='merlin',
          num_train_epochs=3,
          dataloader_drop_last=False,
          per_device_train_batch_size = 256,
          per_device_eval_batch_size = 32,
          gradient_accumulation_steps = 1,
          learning_rate=0.000666,
          report_to = [],
          logging_steps=200,
       )

trainer = Trainer(
```

```
    model=model,
    args=training_args,
    schema=schema,
    compute_metrics=True,
)

OUTPUT_DIR = os.environ.get("OUTPUT_DIR", "/workspace/data/sessions_by_day")

%%time
start_time_window_index = 1
final_time_window_index = 4
for time_index in range(start_time_window_index, final_time_window_index):
    time_index_train = time_index
    time_index_eval = time_index + 1
    train_paths = glob.glob(os.path.join(OUTPUT_DIR,
f"{time_index_train}/train.parquet"))
    eval_paths = glob.glob(os.path.join(OUTPUT_DIR,
f"{time_index_eval}/valid.parquet"))
    print('*'*20)
    print("Launch training for day %s are:" %time_index)
    print('*'*20 + '\n')
    trainer.train_dataset_or_path = train_paths
    trainer.reset_lr_scheduler()
    trainer.train()
    trainer.state.global_step +=1

    trainer.eval_dataset_or_path = eval_paths
    train_metrics = trainer.evaluate(metric_key_prefix='eval')
    print('*'*20)
    print("Eval results for day %s are:\t" %time_index_eval)
    print('\n' + '*'*20 + '\n')
    for key in sorted(train_metrics.keys()):
        print(" %s = %s" % (key, str(train_metrics[key])))
    wipe_memory()
```

```
with open("results.txt", 'a') as f:
    f.write('\n')
    f.write('XLNet-MLM accuracy results:')
    f.write('\n')
    for key, value in  model.compute_metrics().items():
        f.write('%s:%s\n' % (key, value.item()))


import IPython
app = IPython.Application.instance()
app.kernel.do_shutdown(True)

import os
import glob
import nvtabular as nvt

import torch
import transformers4rec.torch as tr
from transformers4rec.torch.ranking_metric import NDCGAt, RecallAt

from merlin.schema import Schema
from merlin.io import Dataset

x_cat_names = ['product_id-list', 'category_id-list', 'brand-list']
x_cont_names    =    ['product_recency_days_log_norm-list',    'et_dayofweek_sin-list',
'et_dayofweek_cos-list',
                'price_log_norm-list', 'relative_price_to_avg_categ_id-list']


INPUT_DATA_DIR = os.environ.get("INPUT_DATA_DIR", "/workspace/data")

train = Dataset(os.path.join(INPUT_DATA_DIR, "processed_nvt/part_0.parquet"))
```

```
schema = train.schema
schema = schema.select_by_name(x_cat_names + x_cont_names)



sequence_length, d_model = 20, 192
inputs= tr.TabularSequenceFeatures.from_schema(
    schema,
    max_sequence_length=sequence_length,
    aggregation="concat",
    d_output=d_model,
    masking="mlm",
)

transformer_config = tr.XLNetConfig.build(
    d_model=d_model, n_head=4, n_layer=2, total_seq_length=sequence_length
)
body = tr.SequentialBlock(
    inputs,          tr.MLPBlock([192]),          tr.TransformerBlock(transformer_config,
masking=inputs.masking)
)

head = tr.Head(
    body,
    tr.NextItemPredictionTask(weight_tying=True,
                              metrics=[NDCGAt(top_ks=[10, 20], labels_onehot=True),
                                       RecallAt(top_ks=[10, 20], labels_onehot=True)]),
)

model = tr.Model(head)

from transformers4rec.config.trainer import T4RecTrainingArguments
from transformers4rec.torch import Trainer
from transformers4rec.torch.utils.examples_utils import wipe_memory
```

```
training_args = T4RecTrainingArguments(
        output_dir="./tmp",
        max_sequence_length=20,
        data_loader_engine='merlin',
        num_train_epochs=3,
        dataloader_drop_last=False,
        per_device_train_batch_size = 256,
        per_device_eval_batch_size = 32,
        gradient_accumulation_steps = 1,
        learning_rate=0.000666,
        report_to = [],
        logging_steps=200,
)

trainer = Trainer(
    model=model,
    args=training_args,
    schema=schema,
    compute_metrics=True,
)

OUTPUT_DIR = os.environ.get("OUTPUT_DIR", "/workspace/data/sessions_by_day")

%%time
start_time_window_index = 1
final_time_window_index = 4
for time_index in range(start_time_window_index, final_time_window_index):
    # Set data
    time_index_train = time_index
    time_index_eval = time_index + 1
    train_paths = glob.glob(os.path.join(OUTPUT_DIR,
f"{time_index_train}/train.parquet"))
```

```
    eval_paths = glob.glob(os.path.join(OUTPUT_DIR,
f"{time_index_eval}/valid.parquet"))
    # Train on day related to time_index
    print('*'*20)
    print("Launch training for day %s are:" %time_index)
    print('*'*20 + '\n')
    trainer.train_dataset_or_path = train_paths
    trainer.reset_lr_scheduler()
    trainer.train()
    trainer.state.global_step +=1
    # Evaluate on the following day
    trainer.eval_dataset_or_path = eval_paths
    train_metrics = trainer.evaluate(metric_key_prefix='eval')
    print('*'*20)
    print("Eval results for day %s are:\t" %time_index_eval)
    print('\n' + '*'*20 + '\n')
    for key in sorted(train_metrics.keys()):
        print(" %s = %s" % (key, str(train_metrics[key])))
    wipe_memory()

with open("results.txt", 'a') as f:
    f.write('\n')
    f.write('XLNet-MLM with side information accuracy results:')
    f.write('\n')
    for key, value in  model.compute_metrics().items():
        f.write('%s:%s\n' % (key, value.item()))
```