Universidade do Minho

Universidade do Minho

Sistemas baseados em Similaridade

Trabalho Prático Nº2

Autores:
Luís Freitas, PG38347
Daniel Pereira, PG42821
João Queirós, A82422
Pedro Queirós, A84220

Grupo:
7

25 de junho de 2021

Conteúdo

1 Introdução

Este trabalho foi realizado no âmbito da cadeira de *Sistemas Baseados em Similaridade*, que se enquadra no 1º semestre do 1º ano do Mestrado em Engenharia Informática da Universidade do Minho, sendo que o seu objetivo visa a construção de algoritmos de recomendação, assim como a sua implementação.

1.1 Sistemas de Recomendação

Os sistemas de recomendação auxiliam o processo de decisão baseado na experiência dos outros utilizadores ou intervenientes do sistema.

Visam acima de tudo melhorar a interface e experiência do utilizador, aumentando a sua satisfação. Faz com que a aplicação seja mais eficiente, apresentando apenas itens e sugestões úteis ao utilizador, melhorando a eficiência do ponto de vista da aplicação, criando uma experiência personalizada para cada utilizador.

Há vários tipos de sistemas de recomendação:

- Sistemas de Recomendação baseados em Conteúdo;
- Sistemas de Recomendação de filtragem colaborativa.

Neste trabalho, demos mais ênfase a sistemas de recomendação de filtragem colaborativa tendo em conta as características do *dataset* sugerido.

Um sistema de recomendação baseado em conteúdo baseia-se na similaridade entre produtos para fazer novas recomendações. Isto é, é sugerido ao utilizador conteúdos parecidos com o conteúdo que consome.

A vantagem deste sistema é conseguir recomendar items sem um histórico grande de recomendações, conseguindo sugerir apenas pelo conteúdo do item. Por outro lado, em alguns casos fica complicado perceber por que características agrupar o conteúdo-

Os sistemas de recomendação de filtragem colaborativa sugere conteúdo aos utilizadores com base no conteúdo consumido por utilizadores. Compara as avaliações de cada utilizador a cada conteúdo de forma a perceber quais é que têm o mesmo gosto.

A vantagem deste sistema é que a escolha das características que fundamentam a recomendação não é subjectiva, é com base nos gostos directos dos utilizadores. O problema deste sistema são os usuários sem histórico, que não têm conteúdo suficiente para terem recomendações fidedignas.

2 Caso de Estudo

2.1 Contexto do Problema

O problema apresentado foi baseado num *dataset* proveniente da *Amazon*, recolhido pelo ***IIF - Institut für Informatik, Universität Freiburg***, recolhido entre Agosto e Setembro de *2004*.

O *dataset* incide sobre uma comunidade de *Book-Crossing*; contém tabelas de *Users*, *Books* e *Book Ratings*, onde, distribuídas pelas três tabelas, contêm informação sobre os livros, informação sobre cada utilizador e o *rating* que cada utilizador deu a cada livro.

É necessário um sistema de recomendações funcional adaptado a cada utilizador de forma a conseguirem uma melhor experiência na comunidade tendo disponíveis os livros mais indicados a cada um. Além disso, é necessário que o utilizador consiga interagir com o sistema de forma a que seja possível a avaliação posterior de livros.

2.2 Primeiros Passos

2.2.1 Análise do *dataset*

O *dataset*, como foi referido anteriormente, é composto por três tabelas: *Users*, *Books* e *Book-Ratings*.
A tabela *Book-Ratings* é composta por:

- ***UserID***: O *UserID* do *user* que deu o *rating*;
- ***ISBN***: O *ISBN* do livro que foi *rated*;
- ***Rate***: O *rating* dado ao livro pelo *user*.

A tabela *Users* é composta por:

- ***UserID***: a chave única de cada utilizador: serve para identificar o utilizador;
- ***Location***: a localização do *user*, sendo um *input* do próprio *user*;
- ***Age***: a idade do *user*.

A tabela *Books* é composta por:

- ***ISBN***: a chave única do livro, de índole geral;
- ***Title***: o título do livro;
- ***Author***: o autor do livro;
- ***Yearpub***: o ano da publicação do livro;
- ***Publisher***: a distribuidora do livro.

Além destas, tendo em conta um melhor dataset, foram adicionados campos à tabela *Users* e à tabela *Books*.

Na *Users*, dividimos a *Location* em *Place*, *City* e *Country* e na *Books* procuramos arranjar mais informação de cada livro: primeiramente tentando um modelo já treinado de *Previsão de Categorias a partir do título do livro* e depois, utilizando o *ISBN*, através de *requests* à *API* da *Google Books*.

Dessas duas formas, nenhuma acabou por ser utilizada na análise e no modelo.

Primeiramente, não utilizamos as previsões do modelo, por um lado por problemas de implementação (devido a versões não atualizadas do *Python*) e por outro porque não eram fiáveis, sendo bastante limitadas em termos de tipos de categoria.

Os *request* dos campos *Categoria*, *Língua* e *MatureRating* (livro destinado a utilizadores adultos ou não) à *API* da *Google* não foram úteis: para a nossa surpresa, tendo em conta a dimensão do serviço, para esses ISBN não estavam disponíveis essas informações: Apenas *5%* dos *ISBN* do *dataset* continha essas informações disponíveis na *API* da *Google Books*. Estudamos a *API* da *Amazon*, tendo em conta a origem do *dataset*, mas, à data deste trabalho, estava indisponível.

2.2.2 Plano de Trabalhos do Projeto

O projecto, tendo em conta os objectivos e contexto já definidos, dividiu-se em três partes:

- Tratar e analisar o *dataset*;
- Criar um sistema de recomendações;
- Criar uma aplicação e integrar o sistema de recomendações de forma a que o utilizador consiga interagir com as recomendações geradas.

2.3 Arquitectura da Solução

A solução final, tendo em conta o que foi já explicado nos capítulos anteriores, passa então por uma aplicação pela qual o utilizador possa receber recomendações de livros, interagir e avaliar essas mesmas recomendações.

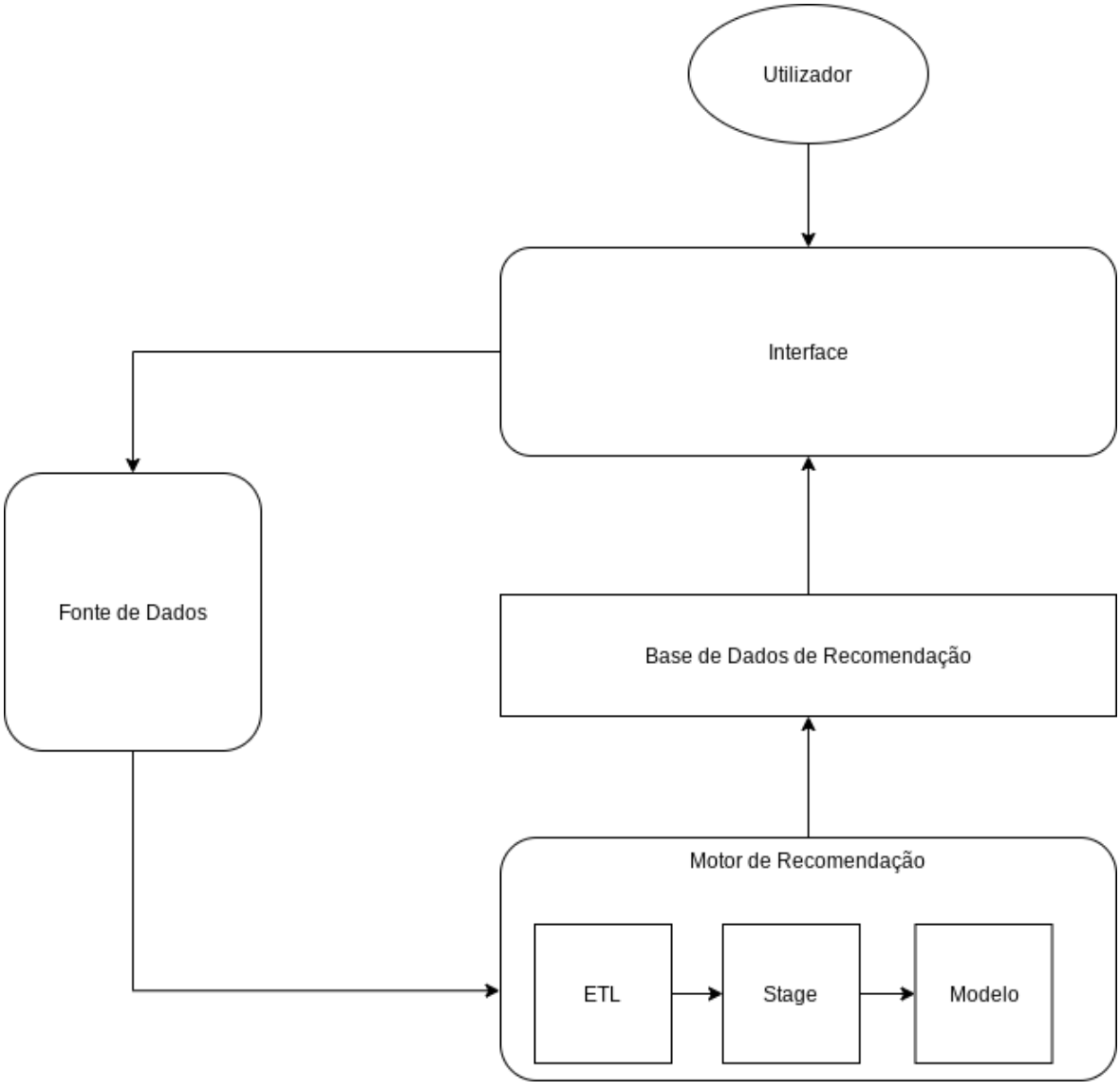

Figura 1: Diagrama a mostrar a arquitetura do trabalho

Inicialmente, os dados presentes na Fonte de Dados encontram-se em formato '.csv'. Este são importados para um workflow Knime para serem tratados, e posteriormente colocados numa nova base de dados mongoDB. Estes dados, por sua vez, são carregados para os vários algoritmos de recomendação, criados em python, que geram as várias variantes de recomendações e as inserem numa outra base de dados, esta em mySQL. Esta última base de dados é aquela que a interface utiliza para mostrar as recomendações aos utilizadores do serviço, sendo que estes mesmos podem realizar ações que levam à inserção de dados na fonte de dados original.

2.4 Análise de Dados

Para a análise dos dados, foi criada uma plataforma e *workflow* próprio e eficaz: Foi criada uma base de dados relacional em *mariadb* que por sua vez está ligada ao *Knime*. Assim, além de ser possível fazer *queries* externamente ao *Knime*, carregamos apenas certas *views* e *queries* de forma a conseguir utilizar todas as ferramentas disponíveis, quer de análise de dados, quer de visualização, sem sobrecarregar o *Knime* com o *dataset* completo.

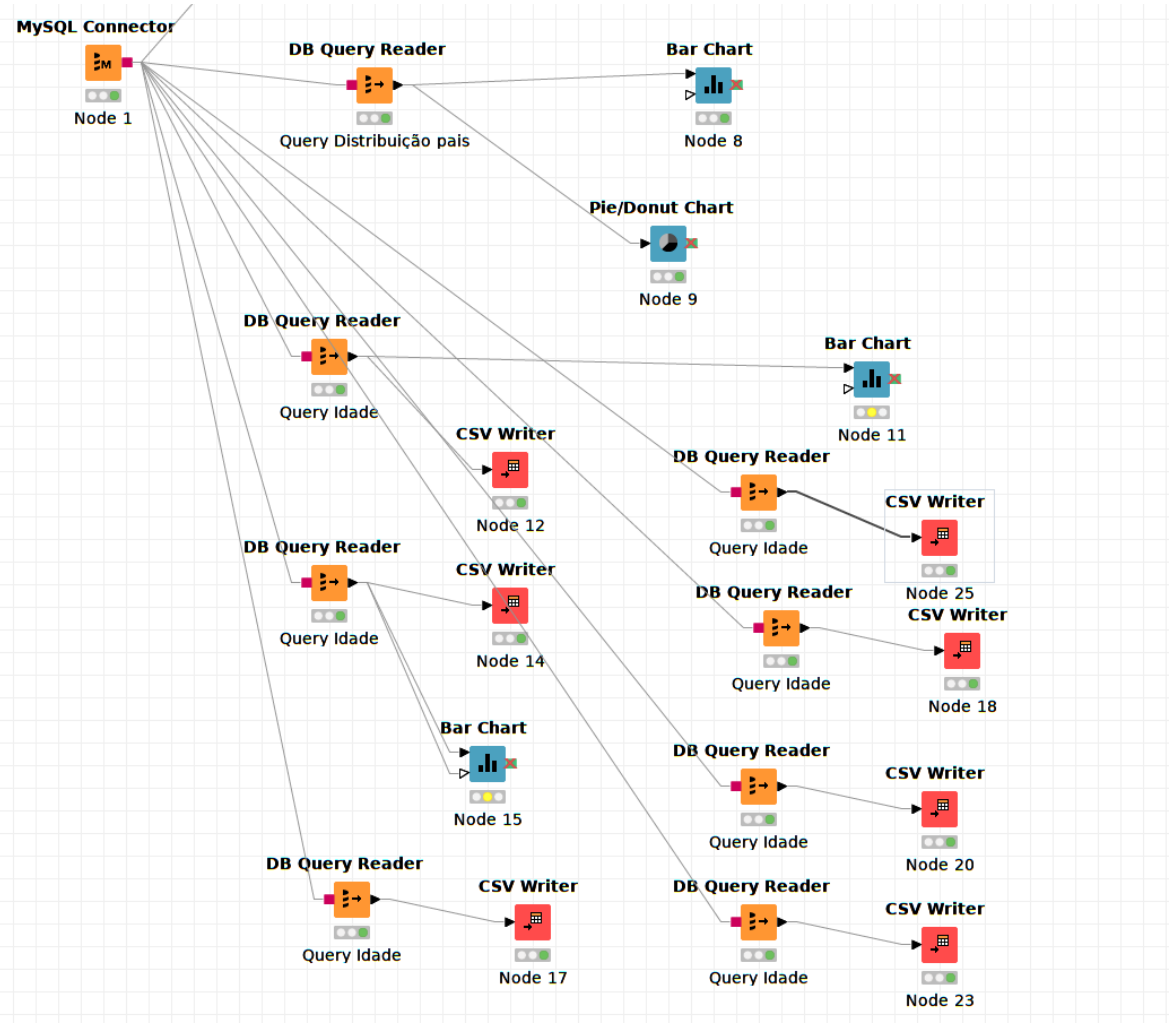

Figura 2: Workspace no Knime para análise de dados

Tendo em conta a extensão da análise, foi dividida em vários pontos:

- **Análise Demagógica**, onde analisamos e percebemos, a demagogia dos nossos utilizadores;
- **Análise de Conteúdo**, onde analisamos os livros que são disponibilizados;
- **Análise aos *Ratings***, onde analisamos as avaliações feitas pelos utilizadores.

2.4.1 Análise Demagógica

Numa primeira olhada, percebemos que o *dataset* tem 278857 **utilizadores**.

Como foi dito antes, os utilizadores vieram caracterizados com a *Location* e *Age*. Antes demais, como também já foi mencionado, dividimos a *Location* em *Country,City* e *Place*.

stockton, california, usa $\rightarrow$ *usa* / *california* / *stockton*

Figura 3: Transformação de *Location* em *Country,City, Place*

Houve apenas um problema com esta divisão: visto o campo ser de preenchimento manual, há muitos erros associados. Por exemplo, houve **1022** países diferentes no input. Quanto às cidades há demasiados erros: Confunde-se os campos de *Place* e *City*; nos estados unidos, tendo em conta que a cidade indica os estados, o erro já não é tão grande.

Visto isto, tendo em conta a qualidade da análise apenas tomamos em conta o *Country*, depois de serem tomadas as devidas precauções: Foram seleccionados 45 países, um *share* de 96% do total de observações. O restante foi agrupado num valor, para não ser perdido, denominado *inválido*.

Assim, olhando à distribuição por país:

50% dos utilizadores são dos Estados Unidos, depois disso 8% são do Canada e 7% dos U.K.

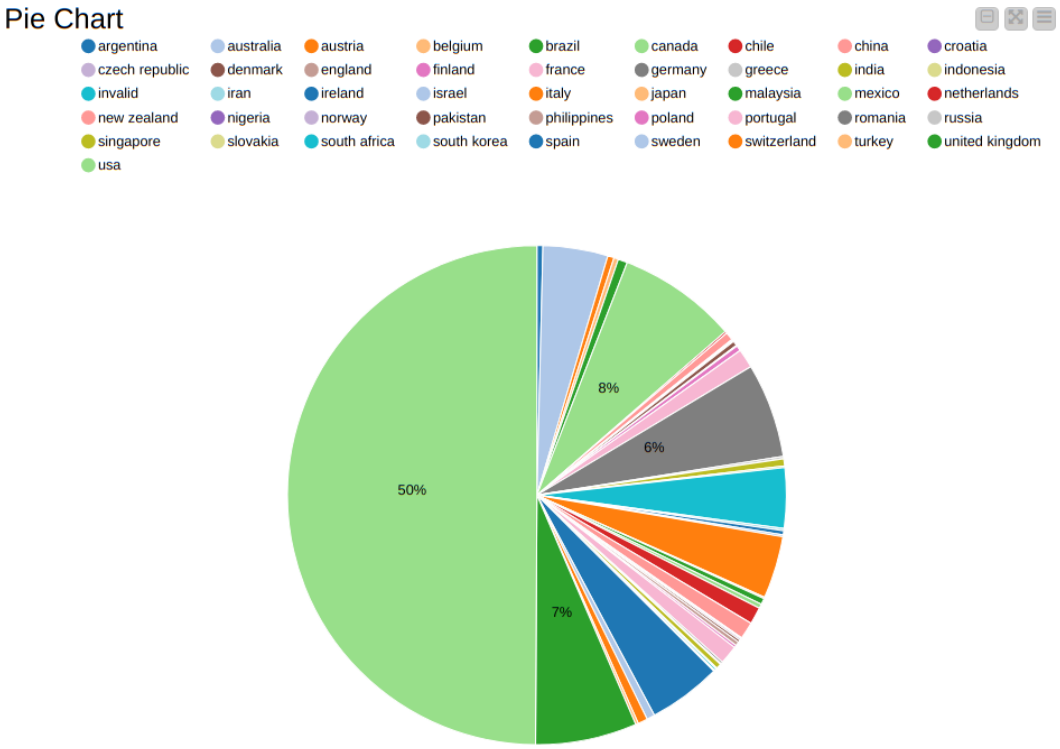

Figura 4: Distribuição por país dos utilizadores

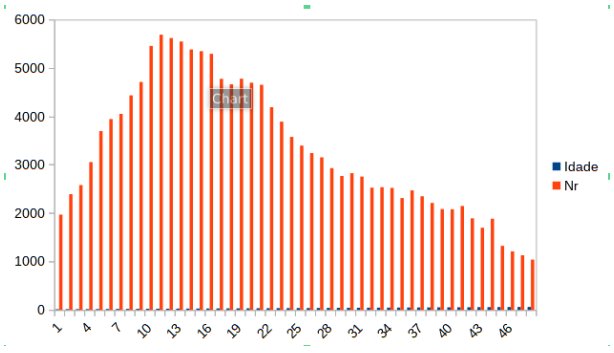

Figura 5: Distribuição das Idades

Em termos de idade, o *dataset*, como também é um campo de preenchimento manual, temos muitos dados esquisitos: primeiro, 39% dos utilizadores não tem idade associada. Além disso, por exemplo, também tem preenchimentos duvidoses, por exemplo, idades acima de 150 anos. Mesmo assim 57% das observações têm idades normais, isto é, entre os 14 (antes de isso é difícil o utilizador fazer uma compra) e os 61.

2.4.2 Análise de Conteúdo

Em termos de conteúdo, percebemos que temos 271399 livros diferentes.

Vendo a distribuição por autor, percebemos que temos 99345 autores diferentes, uma média de 271399/99345 livros por autor.

Em termos de distribuição por *publisher*, percebemos que temos 16575 *pusblisher* diferentes, uma média de 271399/16575 livros por *publisher*.

Temos, em média, 6 autores por *publisher*.

Em termos de distribuição por ano de publicação, percebemos que também temos dados inválidos. Para esta análise consideramos apenas as datas entre 1949 e 2004 válidas. 1.8% das datas são inválidas.

Além disso, percebemos que entre 1995 e 2004 são os anos com mais livros publicados.

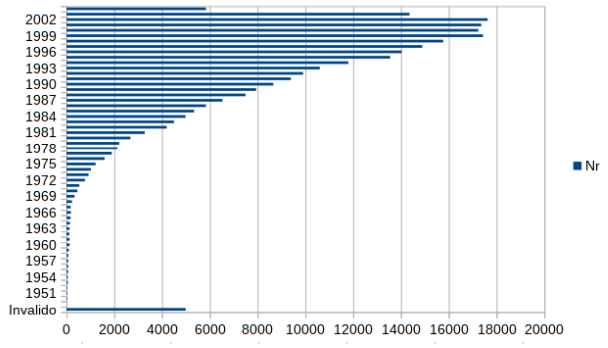

Figura 6: Distribuição do ano de publicação dos livros

2.4.3 Análise aos *Ratings*

No geral, temos 821207 avaliações por user

Olhando às avaliações por idade e localização, percebemos que os países com um racio maior de avaliações por user são a Indonésia e Paquistão . Em termos de idade, percebemos que não influencia muito o facto do utilizador avaliar ou não, mas, entre a idade de 14 e 20. os utilizadores avaliam 40% das vezes.

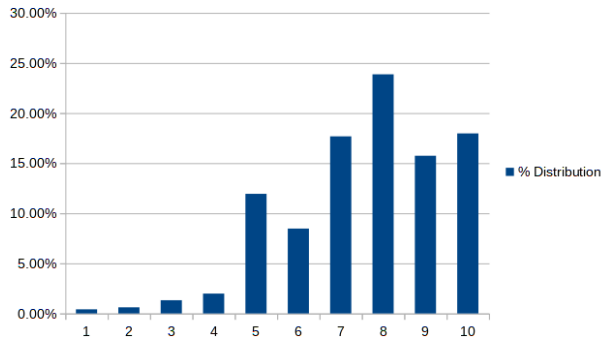

Figura 7: Distribuição do número de avaliações por user

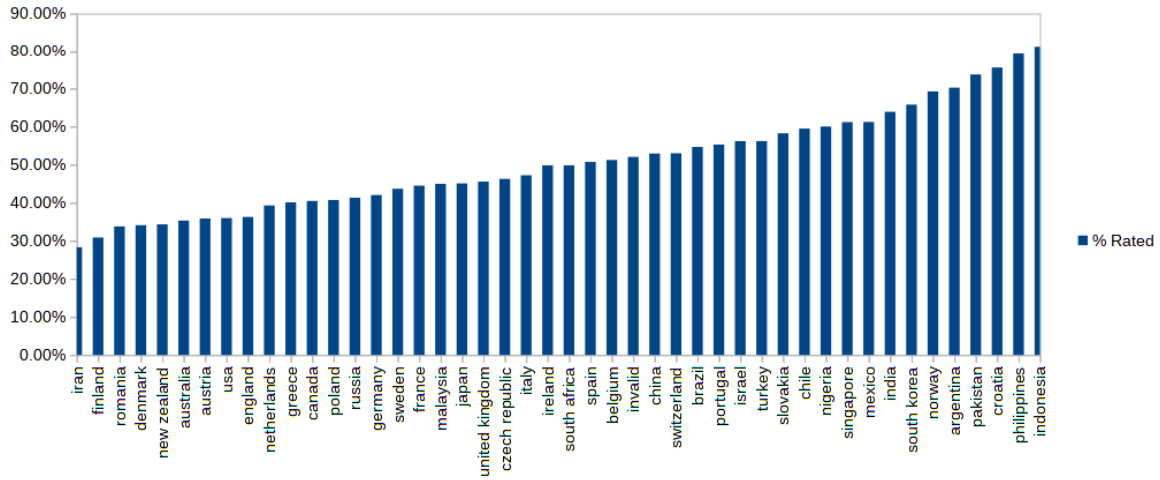

Figura 8: Distribuição do número de avaliações por Localização

Analisamos também a demagogia dos utilizadores e as características dos produtos sujeitos a avaliações melhores.

Nos utilizadores, percebemos que as melhores avaliações são dadas por utilizadores entre os 14 e 24 anos.

Em termos de país, percebemos que as melhores avaliações são dadas nas Filipinas.

3 Desenvolvimento

3.1 Filtragem Colaborativa

Foi implementada a componente de recomendações por filtragem colaborativa, onde foram utilizados e testados dois modelos de *machine learning*.

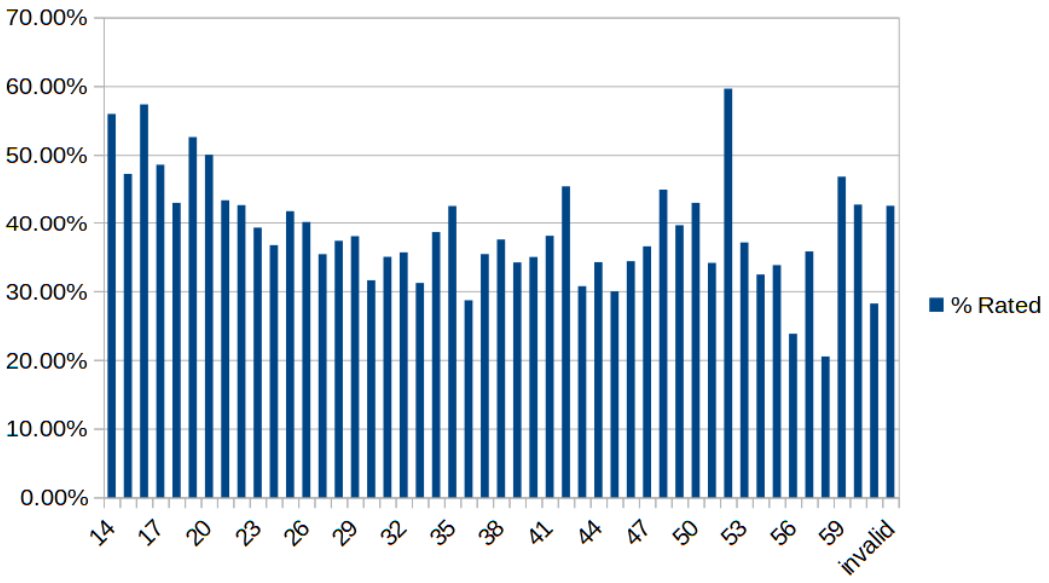

Figura 9: Distribuição do número de avaliações por Idade

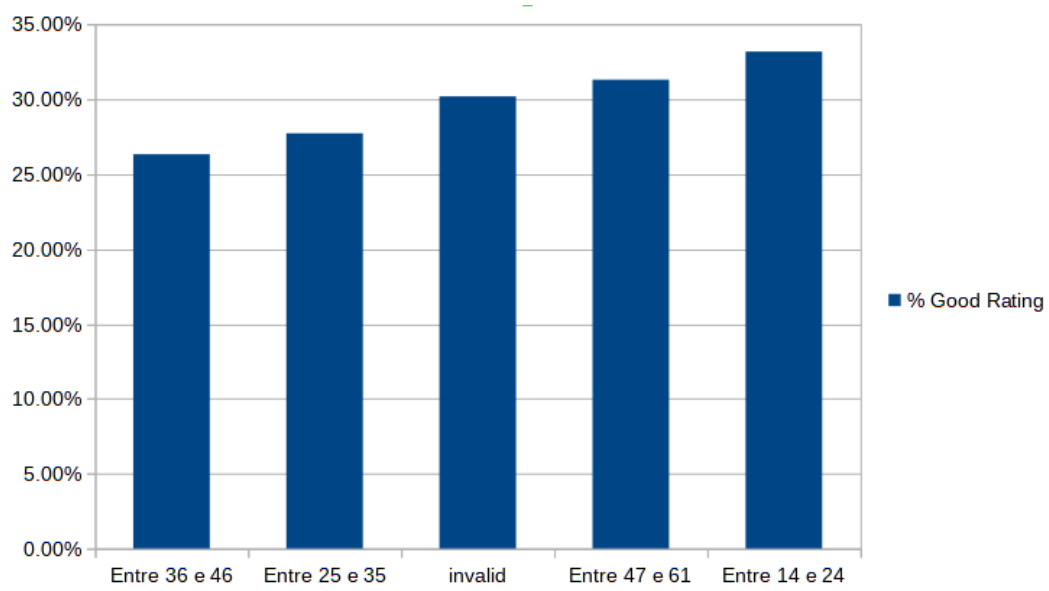

Figura 10: Distribuição do número de avaliações altas por Idade

3.1.1 Preparação dos dados

Na primeira fase de preparação dos dados para este algoritmo foi usada a plataforma *Knime*. O grande objetivo da parte de tratamento de dados foi criar uma componente de modo a que fosse possível guardar os dados tratados numa base de dados *MongoDB*, sendo que esta é a base de dados entre a fase de tratamento de dados e dos modelos de *machine learning*.

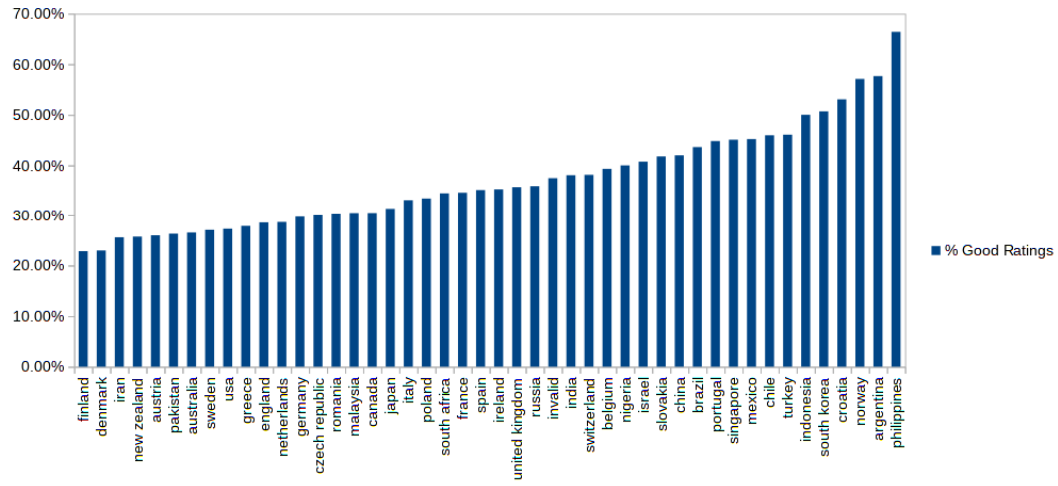

Figura 11: Distribuição do número de avaliações altas por País

A parte de tratamento de dados gera duas collections, uma com o histórico de recomendações dos utilizadores, que será utilizada para testar e desenvolver o modelo e outra com todas as possíveis combinações de utilizadores e livros, em que as combinações não pertencem ao ficheiro de *ratings*, ou seja, o utilizador ainda não leu o livro, que será usado para previsões.

Na parte de tratamento de dados ficou configurado que os utilizadores que iriam receber este tipo de recomendações têm de ter avaliado pelo menos 10 livros. Os livros para poderem ser recomendados através deste algoritmo têm de ter pelo menos 10 visualizações. Em baixo está representada esta componente do workflow no Knime.

No quadrado verde é configurado a filtragem de todos utilizadores e livros que tem mais de 10 avaliações.

Na parte a azul são tratados os dados do histórico de *ratings* e importados para a base de dados de stage em MongoDB.

Por fim, no quadrado a vermelho é feito o cross join entre todos os *users* e livros e retirados todas as combinações que têm *ratings*.

3.1.2 Desenvolvimento do Modelo

A fase de desenvolvimento do modelo foi elaborada na linguagem python, no IDE pycharm com a extensão *surprise*, utilizada no desenvolvimento de sistemas de recomendação.

Um dos modelos desenvolvidos foi o KNN (K-Nearest Neighbour) que é um dos algoritmos básicos de filtragem colaborativa, tendo por base a previsão do valor da classificação que o utilizador irá dar no livro através do cálculo das distancias aos vizinhos mais próximos.

Também são configuradas as medidas de similaridade para estimar a classificação.

O algoritmo foca-se na seguinte fórmula:

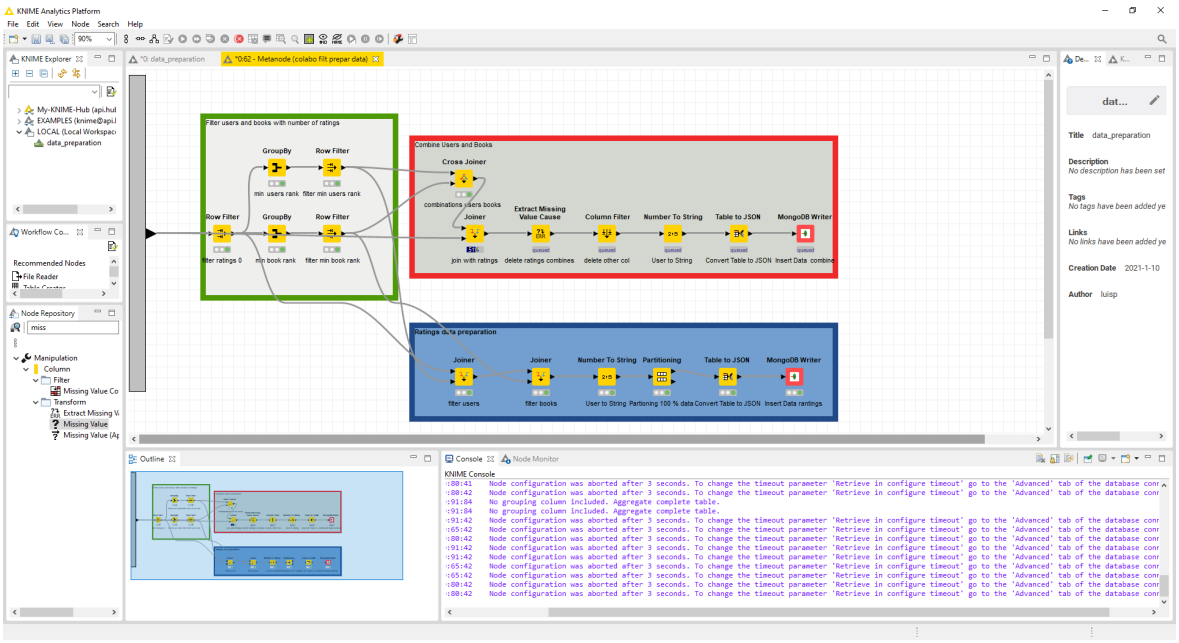

Figura 12: Data Preparation for Collaborative Filtering

$$\hat{r}_{ui} = \frac{\sum\limits_{v \in N_i^k(u)} sim(u,v) \cdot r_{vi}}{\sum\limits_{v \in N_i^k(u)} sim(u,v)}$$

Em que:

- $\hat{r}_{ui}$ é a avaliação estimada do utilizador u para o item i.
- k é o número (max) de vizinhos a serem considerados para agregação
- $sim(u, v)$ é a similaridade do utilizador u com o utilizador v.

Em relação à configuração das similaridades, foi estabelecida a similaridade da diferença média quadrática entre todos os pares de itens. A formula da medida de similaridade é representada de seguida.

$$msd(u, v) = \frac{1}{|I_{uv}|} \cdot \sum_{i \in I_{uv}} (r_{ui} - r_{vi})^2$$

Outro algoritmo utilizado é baseado na fatorização de matrizes, o SVD. A predição $\hat{r}_{ui}$ é definida como:

$$\hat{r}_{ui} = \mu + b_u + b_i + q_i^T p_u$$

Se o utilizador u é desconhecido, então o viés b_u e os fatores pu são considerados zero. O mesmo se aplica ao item i com b_i e q_i.

Para estimar todo o desconhecido, minimizamos o seguinte erro quadrático regularizado:

$$\sum_{r_{ui} \in R_{train}} (r_{ui} - \hat{r}_{ui})^2 + \lambda(b_i^2 + b_u^2 + \parallel q_i \parallel^2 + \parallel p_u \parallel^2)$$

A minimização é realizada por uma descida gradiente estocástica muito direta:

$$b_u \leftarrow b_u + \gamma(e_{ui} - \lambda b_u)$$
$$b_i \leftarrow b_i + \gamma(e_{ui} - \lambda b_i)$$
$$p_u \leftarrow p_u + \gamma(e_{ui} \cdot q_i - \lambda p_u)$$
$$q_i \leftarrow q_i + \gamma(e_{ui} \cdot p_u - \lambda q_i)$$

Em que:

- $\hat{r}_{ui}$ é a avaliação estimada do utilizador u para o item i.
- γ é o termo de regularização
- λ é a taxa de aprendizagem
- p_u é o fator do utilizador
- q_i é o fator dos itens
- μ a média de classificações
- b_u é o viés do utilizador u em relação a média
- b_i viés do artigo i em relação a média
- R_train é o tamanho do conjunto de treino

Para o desenvolvimento destes algoritmos foram criadas funções no python para suportar a implementação.

É desenvolvida a função para exportar os dados da *stage* e converter os dados para *dataframe*:

```
def mong_bx(collections) -> object:
    uri = "mongodb://127.0.0.1:27017"
    client = pymongo.MongoClient(uri)
    database = client['bxdata']
    collection = database[collections]
    cursor = collection.find({})
    return pandas.DataFrame(cursor)
```

Em baixo, é apresentada a função que otimiza os parâmetros nos dois modelos, e cria os modelos com os parâmetros que tem melhor resultado.

```
# parameter otimization
def optgrid(dataset, algorithm):
    # cross validation
    cva = 2
    # measures
    meas = ['rmse', 'mae']

    if algorithm == KNNBasic:
        # var k
        k = list(range(1, 300, 50))
        print("iniciate otimizaion " + str(algorithm) + " :")
        param_grid = {'k': k,
                      'sim_options': {'name': ['msd'],
                                      'user_based': [False]}
                      }

    elif algorithm == SVD:
        # var n_factores
        nf = list(range(1, 397, 99))
        ia = [0.010, 0.020, 0.030]
        ne = [10, 20]
        ra = [0.04, 0.05, 0.06, 0.07]
        print("iniciate otimizaion " + str(algorithm) + " :")
        param_grid = {'n_factors': nf, 'n_epochs': ne, 'init_mean':
    [0], 'init_std_dev': [0],
                      'lr_all': ia, 'reg_all': ra
                      }

    gs = GridSearchCV(algorithm, param_grid, measures=meas, cv=cva)
    gs.fit(dataset)
    # best RMSE score
    print("The best score : ")
    print(gs.best_score['rmse'])
    print("\n")
    print("The best parameters : ")
    print(gs.best_params['rmse'])
    print("\n")
    print("Results : \n")
    print(pd.DataFrame.from_dict(gs.cv_results))
    print("\n")
```

```
    return gs.best_params['rmse']
```

Depois, como só um dos modelos poder ser utilizado, foi desenvolvida uma função que permite aos modelos competirem entre si para definir qual é o melhor através da comparação das médias da *accuracy* num *cross validation*:

```
# Cross Validation
def cross_valid(algori, db):
    return cross_validate(algo=algori, data=db, measures=['RMSE', 'MAE'], cv=2, verbose=False)


def best_model(model1, model2, db):
    first_test = cross_valid(model1, db)
    second_test = cross_valid(model2, db)
    if np.mean(first_test['test_rmse']) > np.mean(second_test['test_rmse']):
        print('the best model is: ' + str(model1))
        return model1
    else:
        print('the best model is: ' + str(model2))
        return model2
```

Finalmente a função que gera as recomendações com input do modelo e do *dataset* que queremos prever, e devolve as 15 melhores previsões para cada utilizador:

```
def rec_gen(model, data_pred):
    col_recfc = [
        'User',
        'Book',
        'Rating'
    ]
    recfc = pd.DataFrame(columns=col_recfc)
    for i in range(len(data_pred)):
        uid = str(data_pred.loc[i, 'User-ID'])
        iid = str(data_pred.loc[i, 'ISBN'])
        predicton = model.predict(uid, iid, verbose=False)
        recfc.loc[-1] = [predicton[0], predicton[1], round(predicton[3], 2)]
        recfc.index = recfc.index + 1
        recfc = recfc.sort_index()
        print(i)
    recfc["rank"] = recfc.groupby("User")["Rating"].rank("first", ascending=False)
    is_rank = recfc["rank"] <= 15
    recfc = recfc[is_rank]
    recfc = recfc.reset_index(drop=True)
    return recfc
```

Depois da implementação das funções é desenvolvido o código principal, que cria o modelo e fornece as previsões desejadas.

```
# DataBase ratings access
print("\ntry access collection ratings...")
try:
    ratings = ma.mong_bx('ratings')
    ratings = ratings.drop(['_id'], axis=1)
except Exception as e:
    print("error accessing database: \n" + str(e) + "\n")
else:
    print("successfully accessed\n")


# Prepare Data
print("\nPrepare Data...")
try:
    reader = Reader(rating_scale=(1, 10))
    data = Dataset.load_from_df(ratings[['User-ID', 'ISBN', 'Book-
   Rating']],
                                reader)
except Exception as e:
    print("error to prepare data : \n" + str(e) + "\n")
else:
    print("successfully prepared data\n")


# Machine Learning Models
try:
    knn = sop.optimization(data, KNNBasic)
    svd = sop.optimization(data, SVD)
except Exception as e:
    print("error in creation models : \n" + str(e) + "\n")
else:
    print("successfully created models\n")


# Decide Model with cross validation
try:
    print('Choose Best Model...')
    model = sop.best_model(knn, svd, data)
except Exception as e:
    print("error in choose models : \n" + str(e) + "\n")
else:
    print("successfully chose the best model\n")

# DataBase datapred access
print("\ntry access collection datapred...")
try:
    datapred = ma.mong_bx('datapred')
    datapred = datapred.drop(['_id'], axis=1)
    datapred.set_index(['ISBN'], inplace=True)
    datapred.reset_index(level=['ISBN'], inplace=True)

```

```
except Exception as e:
    print("error accessing database: \n" + str(e) + "\n")
else:
    print("successfully accessed\n")

# Generate Recomendations
print("\ngenerate recoemndations of collab filt ...")
try:
    recfc = rg.rec_gen(model, datapred)
except Exception as e:
    print("error to generate recomendations: \n" + str(e) + "\n")
else:
    print("successfully recomendations generated\n")

print(recfc)
```

3.2 Filtragem Baseada em Conteúdo

3.2.1 Preparação Dos Dados

3.3 Recomendações *Top World*

As recomendação do *Top World* são as mais simples desenvolvidas, onde é recomendado a todos os utilizadores os melhores *best sellers*.

Esta componente foi inteiramente desenvolvida no Knime e inserida diretamente na base de dados de recomendações. De seguida é apresentado o workflow:

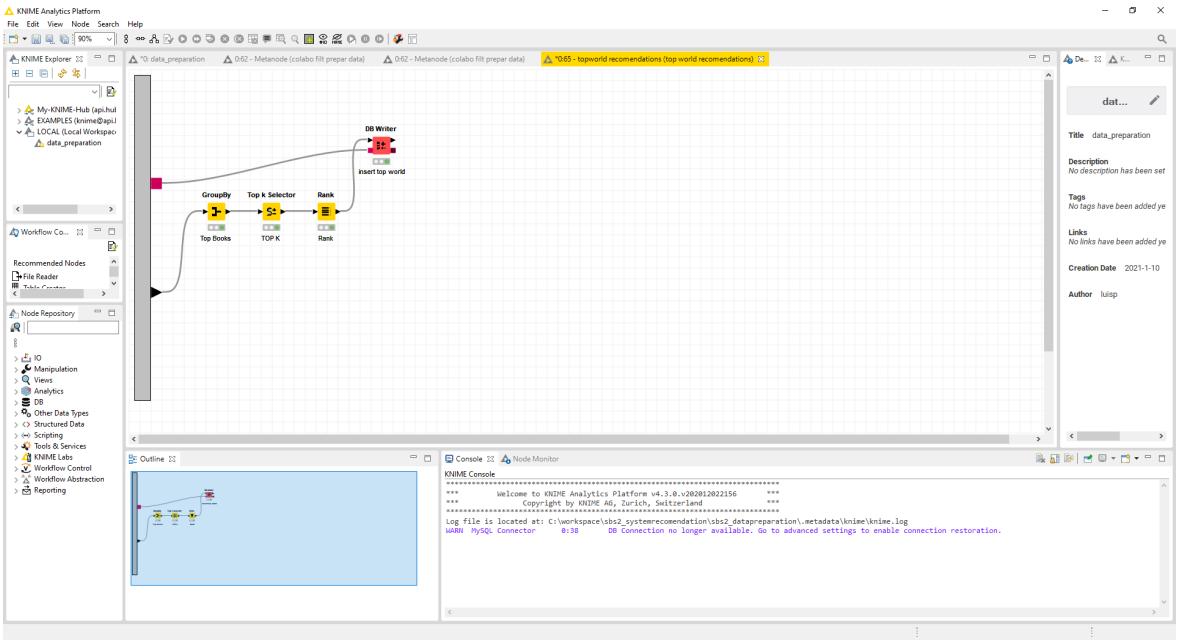

Figura 13: Recomendations Top N - Best Sellers

3.4 Recomendações *Top Country*

Esta componente foi desenvolvida na ferramenta *knime* e na ferramenta *pycharm* através de linguagem *python*. No *knime*, os dados são tratados de forma a criar um *dataset* com o livro, o país e o número de vendas nesse país. Depois, é enviado para a *stage* no *mongoDB* e são geradas as recomendações através do *pycharm*.

De seguida podemos ver o *workflow* no *knime*:

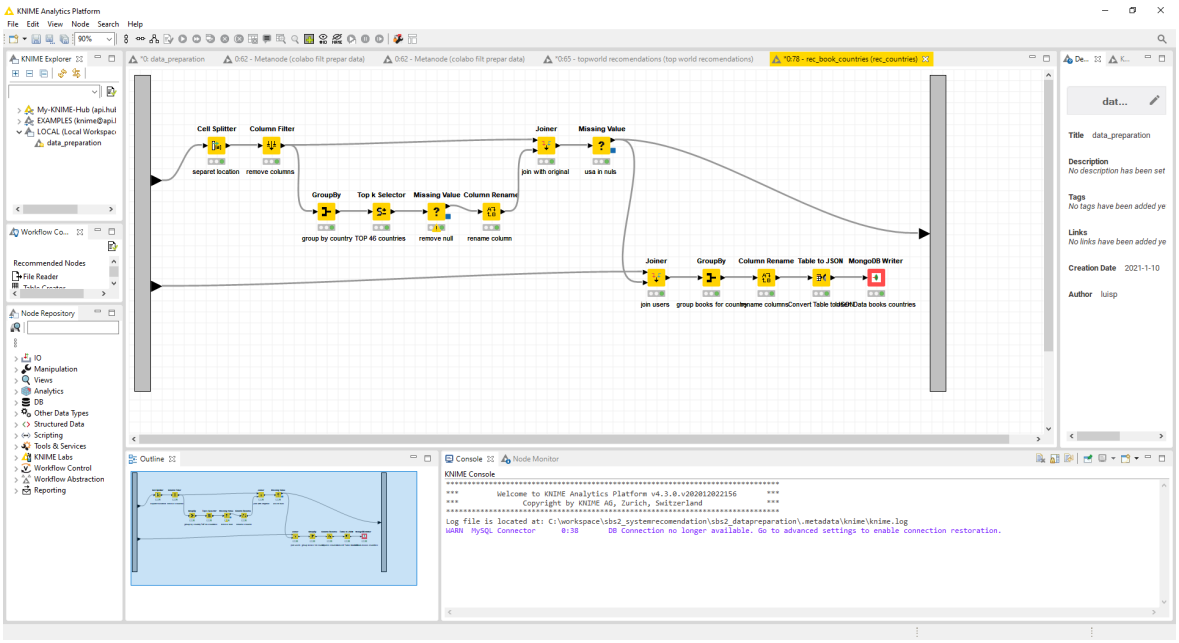

Figura 14: Recomendations Top N - Country

As recomendações são o top 15 de itens em cada país através das vendas, o python executa o processo de definir o top 15 para cada país Em baixo, podemos ver a função desenvolvida e a sua implementação.

```
def coutry_order(data):
    data["rank"] = data.groupby("Country")["Count"].rank("first",
   ascending=False)
    is_rank = data["rank"] <= 15
    data = data[is_rank]
    data = data.reset_index(drop=True)
    return data
```

```
#Top Country
#DataBase isbn_countriesviews access
print("\ntry access collection isbn_countrieviews...")
try:
    isbn_countryviews = ma.mong_bx('isbn_countryviews')
    isbn_countryviews = isbn_countryviews.drop(['_id'], axis=1)
except Exception as e:
    print("error accessing database: \n" + str(e) + "\n")
else:
    print("successfully accessed\n")

isbn_countryviews = rg.coutry_order(isbn_countryviews)
```

3.5 Recomendações *Top Recents*

Neste componente, são selecionados os livros lançados do ano atual e é feito um ranking dos best sellers do atual ano. Serve também para combater o Cold Start e dar oportunidade aos novos livros serem recomendados ao sutilizadores. Esta componente foi desenvolvida inteiramente em Knime, pode ver-se o workflow em baixo :

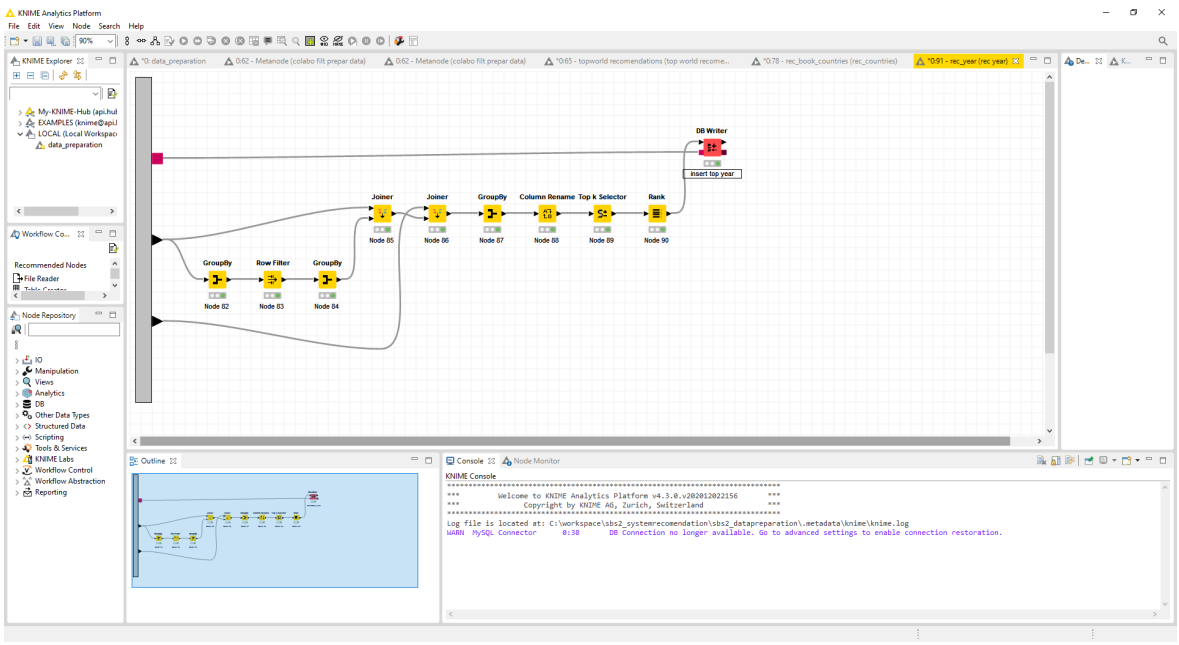

Figura 15: Recomendations Top N - Recents

3.6 Recomendações *Top Authors*

Esta componente tem o objetivo de fazer recomendações das obras do autor favorito do utilizador, a ferramenta knime é utilizada para tratar os dados de forma a inserir na collection stage, os dados das vendas dos livros de cada autor para depois, com python fazer o ranking de livors de cada autor.

Entretanto, o knime também procura a informação do melhor autor para cada utilizador, sendo definido como o autor em que o utilizador compra mais livros.

Na base de dados de recomendações são incluídas duas informações importantes sobre os utilizadores para se poderem gerar as recomendações, a informação do seu país (para a recomendação dos top countries) e a informação do seu autor favorito (para esta recomendação).

Agora é mostrado o workflow que retrata a preparação dos dados para a stage no mongoDB:

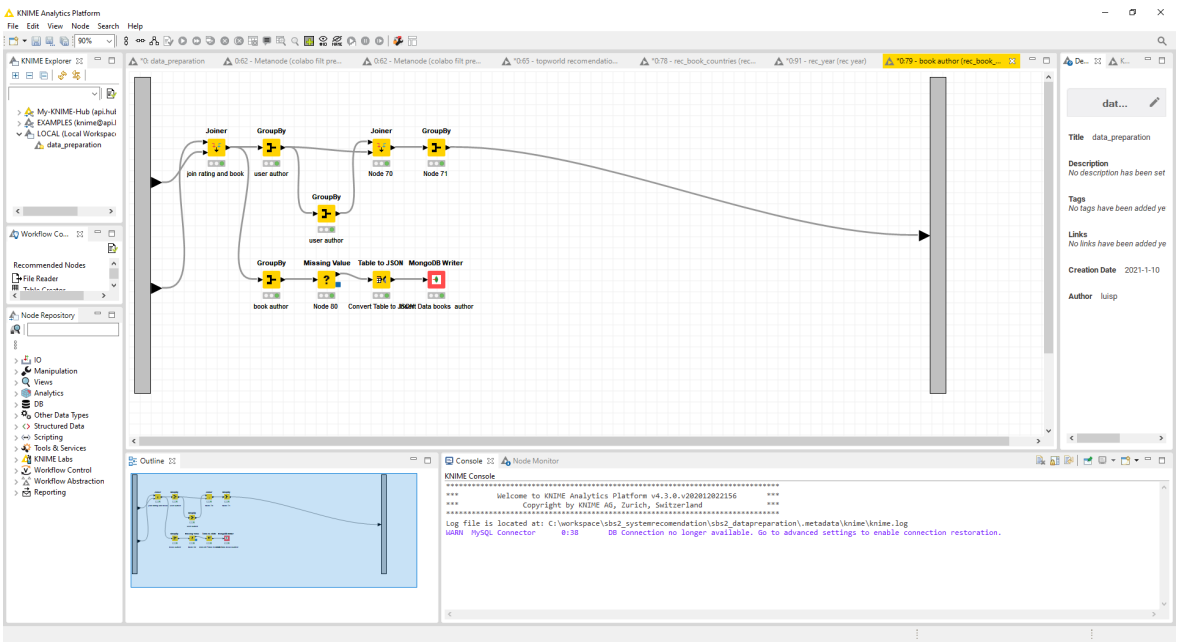

Figura 16: Data Preparation - Top Authors

No python é desenvolvido o código que devolve o rank de cada autor. Podemos ver em baixo:

```
def ab_order(data):
    data["rank"] = data.groupby("Book-Author")["Count*(Book-Rating)"].
   rank("first", ascending=False)
    is_rank = data["rank"] <= 15
    is_book = data["Count*(Book-Rating)"] >= 5
    data = data[is_rank]
    data = data[is_book]
```

```
    data = data.reset_index(drop=True)
    return data

print("\ntry access collection author_books...")
try:
    author_books = ma.mong_bx('author_books')
    author_books = author_books.drop(['_id'], axis=1)
except Exception as e:
    print("error accessing database: \n" + str(e) + "\n")
else:
    print("successfully accessed\n")



author_books = rg.ab_order(author_books)
```

Depois de todo o mecanismo de recomendações elaborado, avançamos para a parte de *front-end* onde será apresentada a base de dados de recomendações e a forma de como os utilizadores interagem com as recomendações.

Front-end

4 Preparação da Interface

A interface do projeto foi desenvolvida em asp.net core Web Pages.

4.1 Página Principal

Página com as recomendações globais, contendo os melhores livros a nível mundial, assim como os melhores livros do ano presente.

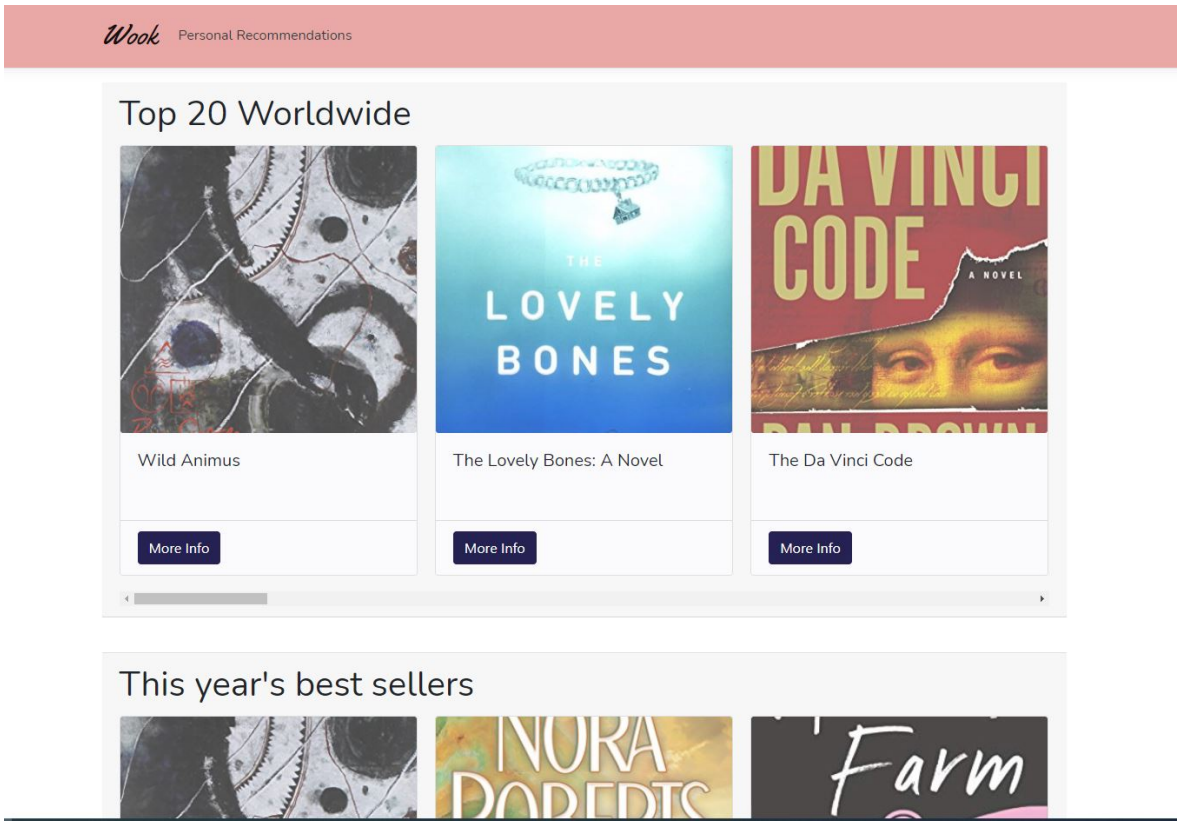

Figura 17: Main page

4.1.1 Página com as recomendações pessoais

Página com as recomendações pessoais, contendo recomendações por filtragem colaborativa, por pais do utilizador e finalmente por autor favorito.

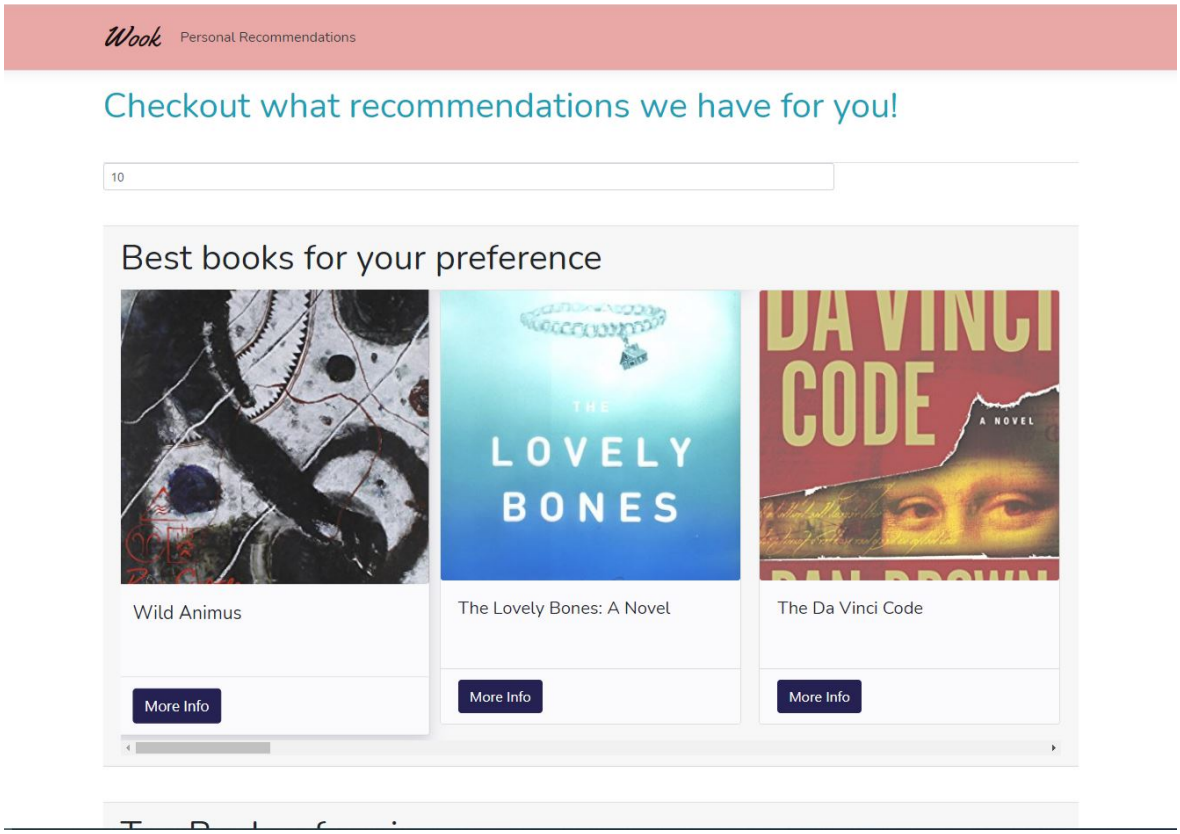

Figura 18: Personal Recomendations page

4.1.2 Pagina com a informação total do livro

Página que disponibiliza mais informação sobre o livro escolhido e que também permite ao utilizador dar um rating ao mesmo.

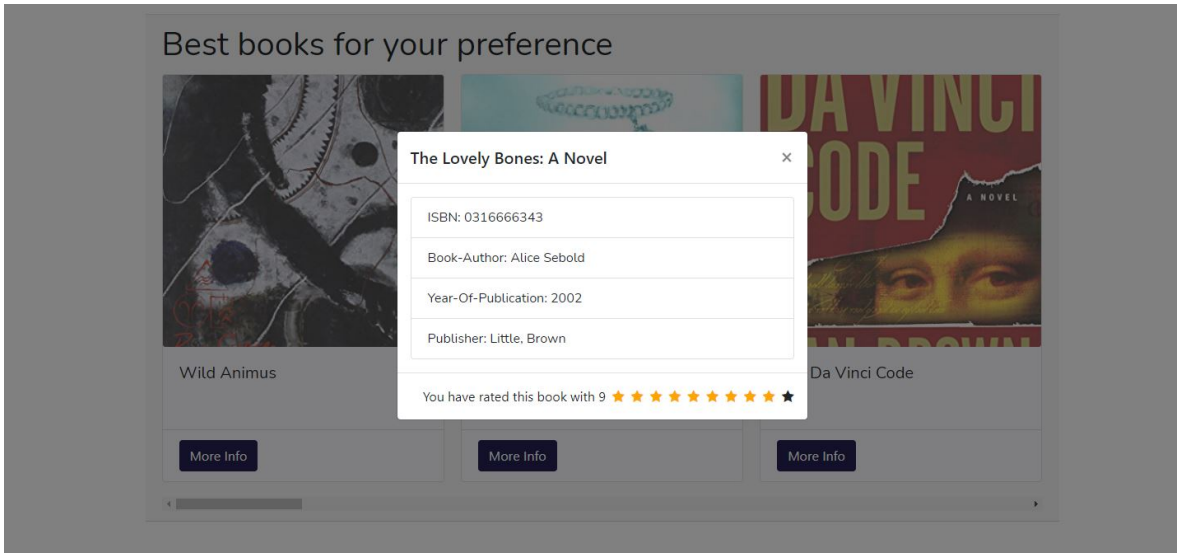

Figura 19: Rating User's Book

5 Conclusão e trabalho futuro

Em suma, achamos que o trabalho realizado foi bastante satisfatório, quer em termos da sua execução, quer em termos dos resultados obtidos, sendo que os vários algoritmos de recomendação geraram resultados distintos entre si.

Para além dos sistemas apresentados, a equipa de trabalho decidiu desenvolver um algoritmo de filtragem baseada em conteúdo para ser implementado no futuro. Neste caso, não foi viável passar à fase de implementação porque não existia informação suficiente sobre os itens. De qualquer forma, a equipa de trabalho testou com toda a informação disponível e desenvolveu um modelo de baseado em conteúdo com técnicas de processamento de linguagem natural. Inicialmente, no Python, foram reunidas todas as palavras associadas aos livros, retiradas das features, através desta query:

```
def import_mtsql(table):
    db_connection_str = 'mysql+pymysql://root:1234567Ll.@localhost/recomendations'
    db_connection = create_engine(db_connection_str)
    df = pandas.read_sql("SELECT  ISBN, lower(concat(`Book-Title`,' ', replace(`Book-Author`, ' ', ''),' ',"
                         "`Year-Of-Publication`, ' ',replace(`Publisher`,' ','''))) as words "
                         "FROM " + str(table) + " limit 3000;", con=db_connection)

    return df
```

Na parte de modelação, para detetar as semelhanças dos livros foi preciso vetorizar e aplicada a similaridade do cosseno para calcular a matriz de similaridade. Depois, foi criada a função que gera recomendações, ou seja, quando se introduz um input(livro), esta devolve 10 livros semelhantes.

```
#Content-Based
#Insert Data

book_inf = ma.import_mtsql('book')
print(book_inf)

count=CountVectorizer()
count_matrix=count.fit_transform(book_inf['words'])

cosine_sim=cosine_similarity(count_matrix ,count_matrix)

print(cosine_sim)

indices = pd.Series(book_inf.index)


def recommendations(title, cosine_sim=cosine_sim):
    recommended_movies = []
```

```
19
20      idx = indices[indices==title].index[0]
21
22      score_series = pd.Series(cosine_sim[idx]).sort_values(ascending=
        False)
23
24      top_10_indexes = list(score_series.iloc[1:11].index)
25
26      for i in top_10_indexes:
27          recommended_movies.append(list(book_inf.index)[i])
28
29      return recommended_movies
```

No entanto, apesar de que testes realizados em pequena escala tivessem funcionado, não foi possível implementá-lo visto que não era possível aplicá-lo à base de dados completa devido à falta de recursos computacionais.

Em termos de trabalho futuro, seria interessante tentar expandir a base de dados que contém a informação dos livros, de modo a permitir aprimorar os sistemas de recomendação, assim como poder criar novos sistemas com critérios diferentes.