Concepção e Implementação de Modelos de Machine Learning usando Árvores de Decisão

Henrique Faria and Paulo Bento

Universidade do Minho, Mestrado Integrado em Engenharia Informática

Resumo Palavras-chave: Python · MySQL · Oracle · Mongo · Neo4J

1 OracleDB

1.1 Criação da base de dados

Motivação A criação da base de dados Oracle pretende responder á necessidade de escalar horizontalmente a nossa base de dados bem com garantir maior flexibilidade e custos reduzidos na manutenção.

Criação - modificações A criação destas bases de dados passa por um script muito identico ao do MySQL.

Ao criar esta base de dados os tipos TINYINT e SMALLINT foram convertidos para NUMBER(10), visto que estes tipos não têm representação nos DataTypes Oracle. Para além disso o tipo BOOLEAN foi convertido no tipo NUMBER(1,0) visto que as bases de dados Oracle também não suportam este tipo de dados sugerindo optar por um VARCHAR2(Y,N) ou com foi impementado NUMBER(1,0). O tipo DATETIME do MySQL foi transformado no tipo correspondente na base de dados Oracle DATE e o tipo VARCHAR foi convertido em VARCHAR2. Por fim o tipo DECIMAL(5,2) presente na base de dados foi convertido para FLOAT(2).

1.2 Criação - Dificuldades na implementação

Ao implementar a mudança da base de dados em python surgiram alguns imprevistos.

Em primeiro lugar a ligação á base de dados foi difícil de conseguir visto que após instalada a base de dados tivemos de proceder á instalação do instant-client, isto suscitou problemas pois este está na versão 19.3.0.0.0 e os turturiais para a instalação deste estão atualizados para a versão 12. Isto levou-nos a procurar soluções em foruns, no git e no StackOverflow até que finalmente as variáveis com o caminho foram criadas e os ficheiros do Instant Client foram colocados na diretoria correta.

Posteriormente surgiram problemas ao correr os scripts para a criação das tabelas pois estes continham pontuação como ';' no fim dos comandos coisa que não pode ser usada nestes scripts no python, removendo os mesmos foi o suficiente para estes começarem a funcionar.

Outro problema que surgiu estava ligado á criação de duas tabelas que continham chaves estrangeiras uma da outra. Por estarmos a executar o script no python como uma das tabelas não tinha sido criada ao declarar uma chave estrangeira de uma tabela que não existe este comando não funcionava. Para resolver este problema, foram criadas primeiro as duas tabelas e a segunda com a chave estrangeira referenciando a primeira, posteriormente foram ambas preenchidas, primeiro a primeira tabela criada e depois a segunda, após serem preenchidas

procedeu-se á alteração da primeira tabela criando a restrição de chave estrangeira correspondente.

Por fim surgiu outro problema ao guardar os dados. Na tabela Film os dados referentes a special_features são devolvidos como tuplos em vez de uma string. Para isso foi necessário executar um ciclo for sobre os dados retirados da tabela MySQL e transformar esse tuplos em strings para poderem ser guardados n nossa base de dados.

1.3 Preenchimento da Base de Dados

Para preencher a base de dados foram usados os seguintes comandos (para exemplificar vão ser listados os comandos usados na tabela language):

- **language_sql = *"SELECT * FROM language"***
  Este primeiro comando serve para criar a query que servirá para ler a tabela language.

- **mycursor.execute(language_sql)**
  Este comando executa no *MySQL* a leitura da tabela language.

- **languageRows = mycursor.fetchall()**
  Este comando vai buscar todos os registos lidos previamente com a query *language_sql*.

- **oracleCursor.bindarraysize = len(languageRows)**
  Este comando serve apenas para que o python saiba quantas vezes terá de executar a query que se segue.

- **oracleCursor.executemany('insert into LANGUAGE(LANGUAGE_ID, NAME,LAST_UPDATE) values(:1,:2,:3)', languageRows)**
  Esta query será executada sobre cada linha da talela languageRows.

- **con.commit()**
  Este comando permite finalizar os processos realizados adicionando as modificações realizadas por estes permanentemente á nossa base de dados Oracle.

Nota: Estes comandos foram usados em cada tabela copiada de MySQL para OracleDB.

1.4 Querys a Base de Dados

Em seguida listam-se algumas das querys feitas á base de dados oracle e as respetivas respostas.

- **res = oracleCursor.execute("SELECT * FROM FILM WHERE FILM_ID <= 3").fetchall()**
  res = [(1, 'ACADEMY DINOSAUR', 'A Epic Drama of a Feminist And a Mad Scientist who must Battle a Teacher in The Canadian Rockies', 2006, 1, None, 6, 0.99, 86, 20.99, 'PG', 'Behind the Scenes Deleted Scenes', datetime.datetime(2006, 2, 15, 5, 3, 42)), (2, 'ACE GOLDFINGER', 'A Astounding Epistle of a Database Administrator And a Explorer who must Find a Car in Ancient China', 2006, 1, None, 3, 4.99, 48, 12.99, 'G', 'Trailers Deleted Scenes', datetime.datetime(2006, 2, 15, 5, 3, 42)), (3, 'ADAPTATION HOLES', 'A Astounding Reflection of a Lumberjack And a Car who must Sink a Lumberjack in A Baloon Factory', 2006, 1, None, 7, 2.99, 50, 18.99, 'NC-17', 'Trailers Deleted Scenes', datetime.datetime(2006, 2, 15, 5, 3, 42))]
-

2 MongoDB

2.1 Criação da base de dados

Motivação Por esta ser uma base de dados orientada a documentos, tem várias vantagens face ao MySQL, para além de ser escalável horizontalmente, é mais facil de manter, atualizar e é ótima para coletar informações referentes a um objeto, evitando joins de tabelas usados pelo MySQL visto este ter toda a informação de um objeto num mesmo documento.

Criação - modificações Pra a criação desta base de dados supusemos ser uma aplicação da loja na qual só se pretende obter informações sobre que utilizador consome filmes em que loja quem o atendeu, em que data e qual o preço. Para além disso é importante saber as carateristicas dos filmes presentes nas lojas, as linguas em que estão, quais os atores que participaram e as categorias em que se insere.

2.2 Criação - Estrutura da Base de Dados

Em seguida apresentam-se os documentos gerados com base nos requisitos para a base de dados MongoDB.

- Filme: "id": , "title": , "release_year": , "descrição": , "original_language": , "foreign_language": , "categorys": , "actors":
- Pagamento: "store_id": , "customer": , "date": , "amount": , "staff":

2.3 Preenchimento da Base de Dados

- **firstQuery = "SELECT f.title,f.release_year,f.description,c.name AS Category,a.first_name,a.last_name, language.name AS Foreign_language,extra.name AS Original_language,f.film_id FROM film AS f LEFT JOIN film_category AS fc ON fc.film_id = f.film_id LEFT JOIN category AS c ON c.category_id = fc.category_id LEFT JOIN film_actor AS fa ON fa.film_id = f.film_id LEFT JOIN actor AS a ON a.actor_id = fa.actor_id LEFT JOIN language ON language.language_id = f.language_id LEFT JOIN (SELECT f.film_id,l.name FROM film AS f LEFT JOIN language AS l ON l.language_id = f.original_language_id WHERE f.original_language_id is not null) extra ON f.film_id = extra.film_id"**
- **mycursor.execute(firstQuery)**
- **filmRecords = mycursor.fetchall()**
- **myFilmIt = iter(filmRecords)**

Em seguida podemos observar o loop usado para criar os documentos film.

```
while (True):
    try:
        i1 = next(myFilmIt)
    except StopIteration:
        print("first collection done")
        break
    if i1aux == None:
        i1aux = i1
    # category and actors
    if i1aux[0] == i1[0]:
        actors.append(i1[4] + " " + i1[5])
        if i1[3] not in categorys:
            categorys.append(i1[3])
    else:
        info = {"id": i1aux[8], "title": i1aux[0], "release_year": i1aux[1], "descrição": i1aux[2],
                "original_language": i1aux[7], "foreign_language": i1aux[6], "categorys": categorys, "actors": actors}
        filmsList.insert_one(info)
        actors.clear()
        categorys.clear()
    i1aux = i1
```

Figura 1. Ciclo que gera cada documento Film guardado na base de dados MongoDB

Cada objeto iterado neste ciclo while corresponde a um film, podendo haver filmes repetidos em iterações sucessivas cuja única diferença reside sempre no ator e pode ter também diferentes categorias, tendo portanto, cada iteração, um ator diferente e uma ou nenhuma categoria. Assim fazendo uso do ciclo

com a verificação do id de cada filme sendo igual permite que identifiquemos o mesmo filme em várias iterações e possamos adicionar os atores todos numa string bem como as categorias. Quando a verificação da igualdade do filme falha sabemos que podemos adicionar o filme á base de dados pois já coletámos todos os atores e categorias deste.

2.4 Querys a Base de Dados

Em seguida listam-se algumas das querys feitas á base de dados Mongo e as respetivas respostas.

- **filmsList.find(, "title": 1, "original_language": 1, "foreign_language": 1, "_id": 0)**

```
{'title': 'ACADEMY DINOSAUR', 'original_language': None, 'foreign_language': 'English'}
{'title': 'ACE GOLDFINGER', 'original_language': None, 'foreign_language': 'English'}
{'title': 'ADAPTATION HOLES', 'original_language': None, 'foreign_language': 'English'}
{'title': 'AFFAIR PREJUDICE', 'original_language': None, 'foreign_language': 'English'}
{'title': 'AFRICAN EGG', 'original_language': None, 'foreign_language': 'English'}
{'title': 'AGENT TRUMAN', 'original_language': None, 'foreign_language': 'English'}
{'title': 'AIRPLANE SIERRA', 'original_language': None, 'foreign_language': 'English'}
{'title': 'AIRPORT POLLOCK', 'original_language': None, 'foreign_language': 'English'}
{'title': 'ALABAMA DEVIL', 'original_language': None, 'foreign_language': 'English'}
{'title': 'ALADDIN CALENDAR', 'original_language': None, 'foreign_language': 'English'}
{'title': 'ALAMO VIDEOTAPE', 'original_language': None, 'foreign_language': 'English'}
{'title': 'ALASKA PHANTOM', 'original_language': None, 'foreign_language': 'English'}
{'title': 'ALI FOREVER', 'original_language': None, 'foreign_language': 'English'}
{'title': 'ALICE FANTASIA', 'original_language': None, 'foreign_language': 'English'}
{'title': 'ALIEN CENTER', 'original_language': None, 'foreign_language': 'English'}
{'title': 'ALLEY EVOLUTION', 'original_language': None, 'foreign_language': 'English'}
{'title': 'ALONE TRIP', 'original_language': None, 'foreign_language': 'English'}
{'title': 'ALTER VICTORY', 'original_language': None, 'foreign_language': 'English'}
{'title': 'AMADEUS HOLY', 'original_language': None, 'foreign_language': 'English'}
{'title': 'AMELIE HELLFIGHTERS', 'original_language': None, 'foreign_language': 'English'}
{'title': 'AMERICAN CIRCUS', 'original_language': None, 'foreign_language': 'English'}
{'title': 'AMISTAD MIDSUMMER', 'original_language': None, 'foreign_language': 'English'}
{'title': 'ANACONDA CONFESSIONS', 'original_language': None, 'foreign_language': 'English'}
{'title': 'ANALYZE HOOSIERS', 'original_language': None, 'foreign_language': 'English'}
{'title': 'ANGELS LIFE', 'original_language': None, 'foreign_language': 'English'}
{'title': 'ANNIE IDENTITY', 'original_language': None, 'foreign_language': 'English'}
{'title': 'ANONYMOUS HUMAN', 'original_language': None, 'foreign_language': 'English'}
```

Figura 2. Excerto do resultado da primeira query feita á base de dados MongoDB

-

- **query2Mongo = filmsList.find("title": 'ACADEMY DINOSAUR', "title": 1, "actors": 1, "_id": 0)**

```
{'title': 'ACADEMY DINOSAUR', 'actors': ['PENELOPE GUINESS', 'CHRISTIAN GABLE', 'LUCILLE TRACY', 'SANDRA PECK'
, 'JOHNNY CAGE', 'MENA TEMPLE', 'WARREN NOLTE', 'OPRAH KILMER', 'ROCK DUKAKIS', 'MARY KEITEL']}
```

Figura 3. Resultado da segunda query feita á base de dados MongoDB

- **query3Mongo = paymentList.find('customer': 'AUSTIN CINTRON', "_id": 0)**

```
{'store_id': 1, 'customer': 'AUSTIN CINTRON', 'date': datetime.datetime(2005, 6, 18, 6, 29, 53), 'amount': 1.99, 'staff': 'Mike Hillyer'}
{'store_id': 1, 'customer': 'AUSTIN CINTRON', 'date': datetime.datetime(2005, 6, 21, 18, 46, 5), 'amount': 6.99, 'staff': 'Mike Hillyer'}
{'store_id': 1, 'customer': 'AUSTIN CINTRON', 'date': datetime.datetime(2005, 7, 9, 4, 42), 'amount': 0.99, 'staff': 'Mike Hillyer'}
{'store_id': 1, 'customer': 'AUSTIN CINTRON', 'date': datetime.datetime(2005, 7, 10, 17, 14, 27), 'amount': 2.99, 'staff': 'Mike Hillyer'}
{'store_id': 1, 'customer': 'AUSTIN CINTRON', 'date': datetime.datetime(2005, 7, 30, 3, 52, 37), 'amount': 6.99, 'staff': 'Mike Hillyer'}
{'store_id': 1, 'customer': 'AUSTIN CINTRON', 'date': datetime.datetime(2005, 8, 21, 5, 7, 8), 'amount': 1.99, 'staff': 'Mike Hillyer'}
{'store_id': 1, 'customer': 'AUSTIN CINTRON', 'date': datetime.datetime(2005, 8, 21, 17, 43, 42), 'amount': 4.99, 'staff': 'Mike Hillyer'}
{'store_id': 1, 'customer': 'AUSTIN CINTRON', 'date': datetime.datetime(2005, 8, 21, 21, 41, 57), 'amount': 1.99, 'staff': 'Mike Hillyer'}
{'store_id': 2, 'customer': 'AUSTIN CINTRON', 'date': datetime.datetime(2005, 6, 18, 6, 29, 53), 'amount': 1.99, 'staff': 'Mike Hillyer'}
{'store_id': 2, 'customer': 'AUSTIN CINTRON', 'date': datetime.datetime(2005, 6, 21, 18, 46, 5), 'amount': 6.99, 'staff': 'Mike Hillyer'}
{'store_id': 2, 'customer': 'AUSTIN CINTRON', 'date': datetime.datetime(2005, 7, 9, 4, 42), 'amount': 0.99, 'staff': 'Mike Hillyer'}
{'store_id': 2, 'customer': 'AUSTIN CINTRON', 'date': datetime.datetime(2005, 7, 10, 17, 14, 27), 'amount': 2.99, 'staff': 'Mike Hillyer'}
{'store_id': 2, 'customer': 'AUSTIN CINTRON', 'date': datetime.datetime(2005, 7, 30, 3, 52, 37), 'amount': 6.99, 'staff': 'Mike Hillyer'}
{'store_id': 2, 'customer': 'AUSTIN CINTRON', 'date': datetime.datetime(2005, 8, 21, 5, 7, 8), 'amount': 1.99, 'staff': 'Mike Hillyer'}
{'store_id': 2, 'customer': 'AUSTIN CINTRON', 'date': datetime.datetime(2005, 8, 21, 17, 43, 42), 'amount': 4.99, 'staff': 'Mike Hillyer'}
{'store_id': 2, 'customer': 'AUSTIN CINTRON', 'date': datetime.datetime(2005, 8, 21, 21, 41, 57), 'amount': 1.99, 'staff': 'Mike Hillyer'}
{'store_id': 1, 'customer': 'AUSTIN CINTRON', 'date': datetime.datetime(2005, 5, 31, 1, 18, 56), 'amount': 4.99, 'staff': 'Jon Stephens'}
{'store_id': 1, 'customer': 'AUSTIN CINTRON', 'date': datetime.datetime(2005, 6, 20, 12, 38, 35), 'amount': 6.99, 'staff': 'Jon Stephens'}
{'store_id': 1, 'customer': 'AUSTIN CINTRON', 'date': datetime.datetime(2005, 6, 21, 15, 34, 38), 'amount': 4.99, 'staff': 'Jon Stephens'}
{'store_id': 1, 'customer': 'AUSTIN CINTRON', 'date': datetime.datetime(2005, 7, 12, 17, 3, 56), 'amount': 9.99, 'staff': 'Jon Stephens'}
{'store_id': 1, 'customer': 'AUSTIN CINTRON', 'date': datetime.datetime(2005, 7, 12, 21, 23, 59), 'amount': 2.99, 'staff': 'Jon Stephens'}
{'store_id': 1, 'customer': 'AUSTIN CINTRON', 'date': datetime.datetime(2005, 7, 31, 4, 57, 7), 'amount': 2.99, 'staff': 'Jon Stephens'}
{'store_id': 1, 'customer': 'AUSTIN CINTRON', 'date': datetime.datetime(2005, 7, 31, 6, 41, 19), 'amount': 2.99, 'staff': 'Jon Stephens'}
{'store_id': 1, 'customer': 'AUSTIN CINTRON', 'date': datetime.datetime(2005, 8, 17, 0, 5, 5), 'amount': 3.99, 'staff': 'Jon Stephens'}
{'store_id': 1, 'customer': 'AUSTIN CINTRON', 'date': datetime.datetime(2005, 8, 23, 6, 9, 44), 'amount': 8.99, 'staff': 'Jon Stephens'}
{'store_id': 1, 'customer': 'AUSTIN CINTRON', 'date': datetime.datetime(2005, 8, 23, 11, 8, 46), 'amount': 2.99, 'staff': 'Jon Stephens'}
{'store_id': 1, 'customer': 'AUSTIN CINTRON', 'date': datetime.datetime(2005, 8, 23, 11, 25), 'amount': 2.99, 'staff': 'Jon Stephens'}
{'store_id': 2, 'customer': 'AUSTIN CINTRON', 'date': datetime.datetime(2005, 5, 31, 1, 18, 56), 'amount': 4.99, 'staff': 'Jon Stephens'}
{'store_id': 2, 'customer': 'AUSTIN CINTRON', 'date': datetime.datetime(2005, 6, 20, 12, 38, 35), 'amount': 6.99, 'staff': 'Jon Stephens'}
{'store_id': 2, 'customer': 'AUSTIN CINTRON', 'date': datetime.datetime(2005, 6, 21, 15, 34, 38), 'amount': 4.99, 'staff': 'Jon Stephens'}
{'store_id': 2, 'customer': 'AUSTIN CINTRON', 'date': datetime.datetime(2005, 7, 12, 17, 3, 56), 'amount': 9.99, 'staff': 'Jon Stephens'}
{'store_id': 2, 'customer': 'AUSTIN CINTRON', 'date': datetime.datetime(2005, 7, 12, 21, 23, 59), 'amount': 2.99, 'staff': 'Jon Stephens'}
{'store_id': 2, 'customer': 'AUSTIN CINTRON', 'date': datetime.datetime(2005, 7, 31, 4, 57, 7), 'amount': 2.99, 'staff': 'Jon Stephens'}
{'store_id': 2, 'customer': 'AUSTIN CINTRON', 'date': datetime.datetime(2005, 7, 31, 6, 41, 19), 'amount': 2.99, 'staff': 'Jon Stephens'}
{'store_id': 2, 'customer': 'AUSTIN CINTRON', 'date': datetime.datetime(2005, 8, 17, 0, 5, 5), 'amount': 3.99, 'staff': 'Jon Stephens'}
{'store_id': 2, 'customer': 'AUSTIN CINTRON', 'date': datetime.datetime(2005, 8, 23, 6, 9, 44), 'amount': 8.99, 'staff': 'Jon Stephens'}
{'store_id': 2, 'customer': 'AUSTIN CINTRON', 'date': datetime.datetime(2005, 8, 23, 11, 8, 46), 'amount': 2.99, 'staff': 'Jon Stephens'}
{'store_id': 2, 'customer': 'AUSTIN CINTRON', 'date': datetime.datetime(2005, 8, 23, 11, 25), 'amount': 2.99, 'staff': 'Jon Stephens'}
```

Figura 4. Resultado da terceira query feita á base de dados MongoDB