Alvaro Austin - 2106752180

Laporan IR - TP 3

Step 1: Preprocessing

a. Mendapatkan dataset queries, qrels, dan juga corpus dokumen dari drive ini: bertujuan untuk nantinya melatih model reranker kita berdasarkan model yang dikonsepkan nanti.
b. Mengganti format dataset yang ada:
    i. Format corpus (collections): mengubah kolom “id” menjadi “docno” karena itu merupakan kebutuhan pyterrier untuk melakukan indexing terhadap dokumen kita.
    ii. Format queries (queries): mengubah kolom “id” menjadi “qid” dan “text” menjadi “query”, karena juga alasan yang sama, untuk melakukan sebuah pyterrier experiment, diperlukan queries dengan format qid dan text.
    iii. Format train/test/dev: mengubah kolom “query-id” menjadi “qid” dan “corpus-id” menjadi “docno”, dan “score” menjadi “label”, hal ini berguna untuk melakukan validasi terhadap experiment yang dilakukan pada queries yang ktia miliki.
c. Melakukan pembuangan non-alphanumeric text pada collections kita
    i. Sebagai contoh, pada collections (corpus) dan query agar query yang dihasilkan dapat semirip mungkin dengan collection yang sudah kita proses. Tidak hanya itu, untuk semua tulisan pada dokumen dan juga query, saya melakukan proses lower case terhadap setiap text tersebut. Alasan hal ini agar mencegah terjadinya “black box” *implementation* dimana kita tidak mengetahui bagaimana processing pada pyterrier dilakukan. Oleh karena itu proses lowercasing ini baik untuk dilakukan untuk mencegah perbedaan antara similarity dokumen dengan query karena perbedaan huruf kapital.

```python
def index(self, text, *args, **kwargs):
    """
    Index the specified

    Args:
        text(pd.Series): A pandas.Series(a column) where each row is the body of text for each document
        *args: Either a pandas.Dataframe or pandas.Series.
            If a Dataframe: All columns(including text) will be passed as metadata
            If a Series: The Series name will be the name of the metadata field and the body will be the metadata content
        **kwargs: Either a list, a tuple or a pandas.Series
            The name of the keyword argument will be the name of the metadata field and the keyword argument contents will be the
            metadata content
    """
    self.checkIndexExists()
    # we need to prevent collectionIterator from being GCd, so assign to a variable that outlives the indexer
    collectionIterator, meta_lengths = DFIndexUtils.create_javaDocIterator(text, *args, **kwargs)

    # record the metadata key names and the length of the values
    self.meta = meta_lengths

    # make a Collection class for Terrier
    javaDocCollection = pt.terrier.J.CollectionFromDocumentIterator(collectionIterator)
    if self.verbose:
        javaDocCollection = TQDMSizeCollection(javaDocCollection, len(text))
    index = self.createIndexer()
    index.index(pt.terrier.J.PTUtils.makeCollection(javaDocCollection))
    global lastdoc
    lastdoc = None
    javaDocCollection.close()
    self.index_called = True
    collectionIterator = None
```

d. Membuat sampling queries (sebanyak 122 queries dimana terdiri atas 42 testing data + 30 dev/validation data + 50 sampling random data).
    i. Hal ini dilakukan dengan melakukan join terhadap kolom pada qrels (train, test, dan juga dev qrels) menggunakan kolom “qid”.
    ii. Semua hasil join tersebut di concat menggunakan method pd.Concat dan melakukan proses *remove_duplicate*, apabila ada data yang duplikasi.
e. Mendapatkan train, test, dan dev queries
    i. Untuk train queries, saya menggunakan inner join terhadap kolom “q_id” pada train_qrels.
    ii. Untuk test queries, saya menggunakan inner join terhadap kolom “q_id” pada test_qrels.
    iii. Untuk dev queries, saya menggunakan inner join terhadap kolom “q_id” pada dev_qrels.
f. Mengapa saya tidak melakukan proses stemming dan stopwords removal
    i. Alasan tidak menggunakan stemming: Dengan melakukan stemming, hal ini dapat membuang konteks yang penting dalam kalimat kita. Dengan melakukan stemming, melalui pipeline yang telah saya buat sendiri yang memanfaatkan bm25, tf-idf dan juga transformer, stemming ini dapat saja membuat inkonsistensi terhadap makna dari query dan juga dokumen teks kita.
    ii. Alasan tidak menggunakan stopwords removal: Hal ini didasari dari pipeline yang saya buat, pipeline tersebut terdiri atas jaccard, bm25, tfidf, normalization, dan sentence embedding, dimana metode seperti bm25, tf-idf dan sentence embedding (transformer) sebenarnya sudah memiliki mekanisme untuk mencegah stopwords menjadi terlalu relevan karena efek dari *inverse document frequency* (IDF).
g. Pengubahan format ID pada query dan corpus
    i. Pada query, setiap ID saya perlu untuk menambahkan prefiks Q pada ID tersebut, contoh untuk ID 12532, maka data pada kolom “q_id” akan bernilai “Q12532”
    ii. Pada dokumen, setiap ID saya perlu untuk menambahkan prefiks D pada ID tersebut, contoh untuk ID 12532, maka data pada kolom “docno” akan bernilai “D12532”

Step 2: Indexing & Baseline Evaluation

a. Proses indeks dilakukan seperti biasa, yaitu dengan menggunakan method DFIndexer lalu memanggil method *.index* pada class yang dibuat oleh DFIndexer,

```
index_ref = pd_indexer.index(collections["text"], collections)
```

b. Selanjutnya kita evaluasi baseline dengan menggunakan model BM25
c. Selanjutnya kita melakukan eksperimen dengan models BM25, lalu evaluation metrics NDCG@10
   Hasil yang diperoleh adalah 0.586706

	name	P@10	map	recip_rank	nDCG@10
0	BM25	0.437209	0.492925	0.684582	0.586706

NDCG sendiri memberikan insights untuk memperoleh ranking yang ideal dari hasil yang di retrieve. nDCG mengukur kualitas ranking dengan mempertimbangkan posisi dokumen relevan dalam daftar yang diambil. nDCG@10 mengevaluasi metrik ini hingga 10 hasil teratas dan menormalkan skor antara 0 dan 1. Pada hal ini, maksudnya nDCG@10 adalah Skor 0,586706 menunjukkan relevansi dan urutan hasil adalah 58,67% optimal dibandingkan dengan peringkat ideal.

Step 3: Feature Engineering

Terdapat 5 fitur yang digunakan:

1. Jaccard Similarity
    a. Deskripsi: Jaccard similarity mengukur kesamaan antara dua set kata, yaitu kata-kata dalam dokumen dan query. Skor dihitung sebagai rasio antara jumlah kata yang sama (intersection) dengan jumlah total kata unik (union).
    b. Tujuan: Fitur ini relevan untuk menangkap keterkaitan kata-kata kunci spesifik, terutama dalam skenario di mana kata-kata dalam query dianggap penting secara eksplisit untuk relevansi dokumen.
2. BM25 Score:
    a. Deskripsi: BM25 adalah skor relevansi berbasis probabilistik yang menghitung frekuensi kata dalam dokumen (term frequency), kelangkaan kata dalam collection (inverse document frequency), dan panjang dokumen.
    b. Tujuan: Fitur ini memberikan *baseline* dalam menilai relevansi dokumen berdasarkan distribusi kata-kata yang relevan dengan query.
3. TF-IDF Cosine Similarity
    a. Deskripsi: TF-IDF cosine similarity mengukur kesamaan antara dokumen dan query dalam ruang vektor menggunakan representasi TF-IDF yang mencerminkan pentingnya istilah dalam dokumen dibandingkan seluruh collection.
    b. Tujuan: Fitur ini membantu mengidentifikasi relevansi berdasarkan bobot kata penting yang tidak hanya sering muncul di dokumen tetapi juga unik dalam collection.
4. Query Length Normalized Overlap
    a. Deskripsi: Mengukur rasio kata dalam query yang ditemukan di dokumen, dinormalisasi terhadap panjang query. Fitur ini menghindari bias terhadap query panjang yang mungkin mengandung banyak kata tak relevan.

   b. Tujuan: Fitur ini menjamin bahwa seluruh kata dalam query (atau mayoritasnya) muncul di dokumen, sehingga mengurangi risiko dokumen dengan relevansi parsial mendapat skor tinggi.
5. Cosine Similarity of Embeddings
   a. Deskripsi: Menggunakan model embedding (SentenceTransformer) untuk menghasilkan representasi semantik dari dokumen dan query, lalu menghitung kesamaan kosinus antar representasi tersebut.
   b. Fitur ini sangat berguna untuk menangani query yang bersifat abstrak atau membutuhkan pemahaman konteks semantik.

Keseluruhan Pipeline Reranker:

- Pertama, BM25 digunakan untuk memberikan skor awal terhadap dokumen berdasarkan query. Ini memberi ranking awal dokumen berdasarkan frekuensi kata dan distribusi kata dalam koleksi. Disini saya mengambil Top 30 menggunakan BM25.
- Setelah skor BM25 dihitung, dokumen yang relevan diambil menggunakan fungsi get_text, yang mengakses kolom "text" untuk setiap dokumen yang ter-ranking.
- Selanjutnya, fitur tambahan dihitung untuk dokumen-dokumen yang dipilih berdasarkan skor BM25. Fitur-fitur ini dapat mencakup berbagai metrik seperti Jaccard, TF-IDF, Query Length Normalized Overlap, dan Sentence Embedding (Transformer) yang melengkapi BM25.

Step 4: Hyperparameter Tuning

Saya melakukan proses tuning dengan membuat sebuah model reranker LambdaMart dengan menggunakan *training set* dan ditune menggunakan dev/validation set untuk mengvalidasi (proses fitting) yang dilakukan:

Berikut adalah parameter yang saya gunakan untuk XGBRanker (LambdaMart):

```
param_grid = {
    "learning_rate": [0.05, 0.1, 0.2],
    "gamma": [0, 1, 2, 5],
    "min_child_weight": [0.1, 1, 3],
    "max_depth": [3, 6],
    "n_estimators": [100, 300],
}
```

Alasan pemilihan:

- Learning Rate:
  Rentang ini dipilih karena nilai dibawah 0.05 membuat training terlalu lambat dan model bisa terjebak di solusi tidak optimal, sedangkan nilai diatas 0.2 menyebabkan penyesuaian skor yang terlalu agresif sehingga ranking tidak stabil dan sulit konvergen.

- Gamma
  Dimulai dari 0 untuk memberikan fleksibilitas maksimal dalam pemisahan tree, hingga 5 sebagai batas atas yang ideal.

- Min Child Weight
  Rentang ini memungkinkan pembentukan node dari yang sangat detail (0.1) hingga cukup general (3).

- Max Depth
  Depth yang dipilih adalah 3,6. Alasannya adalah tree dengan kedalaman kurang dari 3 terlalu simpel untuk menangkap pola ranking yang kompleks, selanjutnya diatas 6 bisa membuat proses tuning menjadi sangat lama.

- N Estimators:
  Untuk estimators, awalnya digunakan 100 tree diperlukan untuk menangkap pola ranking yang kompleks lalu tidak lebih dari 300 karena dapat memberikan peningkatan performa yang minimal namun signifikan meningkatkan waktu komputasi dan risiko overfitting.

Best hyperparameters:
Parameter terbaik adalah parameter dengan spesifikasi:
- Gamma: 0
- Learning Rate: 0.05
- Max Depth: 6
- Min Child Weight: 1
- N Estimators: 100

Dengan score terhadap dev set sebesar **0.937**, artinya ranking optimal dan relevan sebanyak 94%.

Alasan pemilihan hyperparameter yang tidak banyak (random):
- Pemilihan ini didasari karena proses yang sangat lama untuk melakukan tuning. Disaat sekarang untuk parameter tersebut dengan parent queries yang hanya berjumlah 122 queries, hal ini saja memakan waktu **sebanyak 8 jam**.
- Alasan saya juga menggunakan hanya 122 samples, dibandingkan 1000 sample query yang bisa didapatkan, maka **1 iterasi parameter, akan memakan waktu sebanyak 3 jam.** Hal ini berarti untuk setiap parameter (apabila kita iterasi seluruh parameter, dimana terdapat 144 parameter), maka akan memakan waktu selama **18 hari** untuk hyperparameter tuning tersebut selesai.
- Tentu saja ada *drawback* yang terjadi, yaitu hasil reranker tidak sebaik yang diharapkan, dengan reranker menggunakan best parameter, bisa mencapai nDCG@10 sebesar 0.7 untuk **122 parent queries** yang digunakan. Akan tetapi untuk **query sebanyak 1000 parent query,** proses fitting terhadap test_query dan test_qrels

```
Generate  + Code  + Markdown                                                    Python

lmart_pipe = pipeline >> pt.ltr.apply_learned_model(final_model, form = "ltr")
lmart_pipe.fit(train_queries, train_qrels, test_queries, test_qrels)
[99]  ✓ 122m 32.0s                                                            Python

...  0%|          | 0/6992 [2:19:26<?, ?documents/s]
     0%|          | 0/6992 [23:27<?, ?documents/s]
     0%|          | 0/6992 [23:28<?, ?documents/s]
d:\IR\TP3\env\Lib\site-packages\xgboost\core.py:158: UserWarning: [17:43:53] WARNING: C:\buildkite-agent\builds\buildkite-windows-cpu-autoscaling-gro

    E.g. tree_method = "hist", device = "cuda"

  warnings.warn(smsg, UserWarning)
d:\IR\TP3\env\Lib\site-packages\xgboost\core.py:158: UserWarning: [17:43:53] WARNING: C:\buildkite-agent\builds\buildkite-windows-cpu-autoscaling-gro
Parameters: { "verbose" } are not used.

  warnings.warn(smsg, UserWarning)
```

mencapai 2 jam, dan untuk **parameter yang random (bukan terbaik)** memberikan performa nDCG sebesar 0.72

	name	map	recip_rank	P@10	nDCG@10	map +	map -	map p-value	recip_rank +	recip_rank -	recip_rank p-value	P@10 +	P@10 -	P@10 p-value	nDCG@10 +	nDCG@10 -
0	BM25	0.492329	0.684582	0.432558	0.583628	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
1	L2R	0.579679	0.852713	0.523256	0.727161	28.0	11.0	0.002698	16.0	4.0	0.004178	25.0	1.0	0.000002	30.0	6.0

Hal ini berarti, performa model saya **bisa menjadi lebih baik**, apabila terdapat **computing resources** yang lebih banyak dan sample query yang lebih banyak.

Step 5: Evaluasi

Seperti yang disinggung sebelumnya, evaluasi re-ranking testing set saya dengan menggunakan parameter terbaik yang diperoleh dari hyperparameter tuning memberikan performa sebesar 0.7 untuk nilai nDCG@10 nya.

	name	map	recip_rank	P@10	nDCG@10	map +	map -	map p-value	recip_rank +	recip_rank -	recip_rank p-value	P@10 +	P@10 -	P@10 p-value	nDCG@10 +	nDCG@10 -	nDCG@10 p-value
0	BM25	0.492925	0.684582	0.437209	0.586706	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
1	L2R	0.566752	0.837016	0.502326	0.708312	26.0	11.0	0.002424	16.0	2.0	0.00375	22.0	5.0	0.000498	28.0	7.0	0.000058

Berdasarkan hasil tersebut:

- Hasil statistical significance test menunjukkan p-value < 0.05 untuk semua metrik (MAP, Reciprocal Rank, P@10, nDCG@10), mengkonfirmasi bahwa peningkatan performa adalah signifikan secara statistik
- Peningkatan nDCG@10 sebesar 0.11 poin (dari 0.59 ke 0.70) menunjukkan bahwa model berhasil melakukan re-ranking yang lebih baik, terutama untuk dokumen-dokumen di posisi atas.
- Hasil ini menunjukkan bahwa LambdaMART dapat secara efektif memperbaiki ranking BM25 dengan memanfaatkan fitur-fitur tambahan dan learning dari data.

Permasalahan:

- Performa ini **sebenarnya bukan performa maksimal** karena terdapat sampling yang dilakukan yang menyebabkan performa menurun. Menurut saya, performa maksimal yang dapat diperoleh **apabila kita menggunakan seluruh data** adalah sebesar 0.76 pada nDCG@10, hal ini karena **saya mencoba megevaluasi nilai nDCG@10** untuk

parameter random dibandingkan parameter terbaik pada LambdaMart terdapat perbedaan sebesar 0.02 (dari 0.68 sampai 0.70). Hal ini dilakukan hanya pada 112 samples query.
- Sebelumnya, apabila kita menggunakan **semua** query, parameter random pada LambdaMart dapat menghasilkan performa sebesar 0.72, oleh karena itu saya berasumsi bahwa 0.76 merupakan batas atas dari performa Re-Ranker saya.

Jawaban yang lebih komprehensif terhadap perbandingan terhadap baseline:

1. **Performa Metrik Dasar:**
    - P@10:
        - BM25: 0.437209
        - L2R: 0.502326
        - Peningkatan: +15%
    - nDCG@10:
        - BM25: 0.586706
        - L2R: 0.708312
        - Peningkatan: +20.7%
2. **Signifikansi Statistik:**
    - L2R menunjukkan p-value yang sangat signifikan untuk semua metrik:
        - map p-value: 0.002424 (< 0.05)
        - reciprocal rank p-value: 0.00375 (< 0.05)
        - P@10 p-value: 0.000498 (< 0.05)
        - nDCG@10 p-value: 0.000058 (< 0.05)

3. Analisis Komprehensif

a. **Efektivitas Re-ranking**
    - Model L2R secara konsisten mengungguli baseline BM25 di semua metrik evaluasi
    - Peningkatan terbesar terlihat pada nDCG@10, menunjukkan bahwa L2R sangat baik dalam mengurutkan dokumen yang relevan di posisi atas

b. **Signifikansi Statistik:**
    - Semua p-value < 0.05, yang berarti peningkatan performa adalah statistik signifikan
    - P-value terkecil ada pada nDCG@10 (0.000058), menunjukkan bahwa peningkatan pada metrik ini sangat meyakinkan secara statistik

c. **Performa Delta**
    - map+/map-: 26.0/11.0
    - reciprocal_rank+/reciprocal_rank-: 16.0/2.0
    - P@10+/P@10-: 22.0/5.0
    - nDCG@10+/nDCG@10-: 28.0/7.0

    - Kesimpulan: Rasio positif/negatif yang tinggi ini menunjukkan bahwa L2R lebih sering meningkatkan performa daripada menurunkannya.

Kesimpulannya:

- L2R terbukti menjadi metode re-ranking yang efektif
- Peningkatan performa signifikan secara statistik di semua metrik
- Khususnya baik dalam meningkatkan relevansi dokumen di posisi atas (ditunjukkan oleh peningkatan nDCG@10)
- Model ini konsisten dalam meningkatkan performa, dengan lebih banyak kasus peningkatan daripada penurunan

Rekomendasi:

- L2R layak diimplementasikan sebagai pengganti atau pelengkap BM25
- Fokus khusus bisa diberikan pada optimasi nDCG@10 karena menunjukkan peningkatan terbesar