> Оценка релевантности поисковой выдачи

Основные метрики оценки эффективности моделей ранжирования выполняются путем сравнения между списком ранжирования выданным моделью **(model output)** и приведенным в качестве базовой истины **(ground truth)**.

Классические метрики:

- Точность **(precision)** - это количество релевантных документов в выдаче, деленное на общее количество документов в выдаче
- Полнота **(recall)** - это количество релевантных документов в выдаче, деленное на общее количество релевантных документов находящихся в базе поисковой системы

Однако эти параметры не учитывают ранжирование выдачи,

Одна из самых популярных метрик ранжирования - это **NDCG@K** (Normalized Discounted Cumulative Gain at K) которая учитывает порядок элементов в списке выдачи. В следующих 4-х задачах вам предстоит написать поэтапно реализовать метрики **CG, DCG, NDCG**, **average NDCG across user**.

Материал для более полного изучения материала метрик поисковой выдачи:

- https://machinelearningmedium.com/2017/07/24/discounted-cumulative-gain/
- https://www.geeksforgeeks.org/normalized-discounted-cumulative-gain-multilabel-ranking-metrics-ml/
- https://www.youtube.com/watch?v=nCtM4Xg7e4k
- https://www.youtube.com/watch?v=qm1In7NH8WE
- https://medium.com/valassis-engineering-blog/p-ndcg-a-new-learning-to-rank-metric-for-large-lists-of-imbalanced-binary-data-4bee19dc4734

> Cumulative Gain

Вы разработчик в компании **Karpov Courses**, последние 4 месяца вы занимались разработкой бота **CyberTolya** который выдает релевантные по запросу студента видео с youtube канала. Но в процессе разработки вы поддерживали две версии которые используют разные технологии **CyberTolya_v0** и **CyberTolya_v1**. Ваша задача оценить какая версия бота выдает более релевантную выдачу.

Вам необходимо реализовать метрику **Cumulative Gain**, которая была бы адекватной для данной задачи.

```python
def cumulative_gain(rel: Union[float, int], k: int = None) -> float:
```

> Распределение баллов

- **0%** – корректность формата на входе/выходе (API). Если тест не пройден, за всё решение 0 баллов.
- **60%** — проверка корректности вашей метрики
- **20%** – скорость работы (полный балл, если ваше решение выполняется быстрее чем ...)
- **20%** — качество кода (Pylint Score; соблюдение PEP8)

> Discounted Cumulative Gain [1/2]

Вы успешно реализовали **Cumulative Gain**, однако эта метрика не лучшим способом учитывает порядок элементов, она не меняется от перемены рангов. Допустим по запросу Систем Дизайн и Валерий Бабушкин **CyberTolya** выдаст одинаковые ранги в разном порядке, но чаще всего пользователи смотря на первичную выдачу. Эту проблема устраняет другая более модицифированная метрика!

Вам необходимо реализовать метрику **Discounted Cumulative Gain**, которая является модификацией метрики **CG** учитывающая порядок элементов в списке выдачи путем домножения релевантности элемента на вес равный обратному логарифму номера позиции. Существует альтернативная формулировка **DCG**, которая уделяет больше внимания поиску соответствующих документов, добавляя экспоненциальная домножение к элементу.

Метрика имеет две версии **Standart** и **Industry**, которые должны задаваться параметром method принимающим булевые значения (**standart**: method==0, **industry**: method==1).

```python
def discounted_cumulative_gain(rel: Union[float, int], k: int = None, method=0) -> float:

```

> Распределение баллов

- **0%** – корректность формата на входе/выходе (API). Если тест не пройден, за всё решение 0 баллов.
- **60%** — проверка корректности вашей метрики (20% standart, 20% industry)
- **20%** – скорость работы (полный балл, если ваше решение выполняется быстрее чем ...)
- **20%** — качество кода (Pylint Score; соблюдение PEP8)

> Discounted Cumulative Gain [2/2]

Вам удалось реализовать **Discounted Cumulative Gain**, а теперь попробуем сделать **Ideal Discounted Cumulative Gain** - это такая метрика которая идеальна для данного запроса. Мы можем ее использовать относительно **DCG** чтобы понимать насколько хорошо работает выдача по сравнению с ее идеальной версией. Она так же понадобиться нам в следующем задании!

Метрика имеет две версии **Standart** и **Industry**, которые должны задаваться параметром method принимающим булевые значения (**standart**: method==0, **industry**: method==1).

```
def ideal_discounted_cumulative_gain(rel: Union[float, int], k: int = None, method=0) -> float:
```

> Распределение баллов

- **0%** – корректность формата на входе/выходе (API). Если тест не пройден, за всё решение 0 баллов.
- **60%** — проверка корректности вашей метрики (20% standart, 20% industry)
- **20%** – скорость работы (полный балл, если ваше решение выполняется быстрее чем ...)
- **20%** — качество кода (Pylint Score; соблюдение PEP8)

> Normalized Discounted Cumulative Gain

Отлично! Теперь мы можем оценить выдачу моделей **CyberTolya...** или все таки нет? Метрика **DCG@k** плоха тем, что расположено в пределах **[0, K]**, что усложняет задачу сравнения ведь при разных **k**, мы не сможем сравнить модели. Для этого нам нужна нормировка, ранее мы уже реализовали метрики **DCG** и **IDCG**, которые помогут вам в этом задании! Данная метрика хороша тем, что она распределена от **[0,1]** и при любых **k** метрика модели будет нормирована и адекватна для сравнения.

Вам необходимо реализовать метрику **Normalized Discounted Cumulative Gain**, которая была бы адекватной для данной задачи.

```python
def normalized_discounted_cumulative_gain(rel: Union[float, int], k: int = None, method=0) -> float:
```

> Распределение баллов

- **0%** – корректность формата на входе/выходе (API). Если тест не пройден, за всё решение 0 баллов.
- **60%** — проверка корректности вашей метрики.
- **20%** – скорость работы (полный балл, если ваше решение выполняется быстрее чем ...)
- **20%** — качество кода (Pylint Score; соблюдение PEP8)

> Average Normalized Discounted Cumulative Gain across users

Замечательно! Мы уже умеем сравнивать релевантность выдачи моделей по какому то конкретному запросу, но согласитесь для разных запросов точность между **CyberTolya_v1** и **CyberTolya_v2** может разниться. Следовательно нам нужна метрика которая будет в среднем оценивать насколько хорошо модели ранжируют по различным запросам.

Вам необходимо реализовать метрику **Average Normalized Discounted Cumulative Gain across users**, которая была бы адекватной для данной задачи.

```
def average_ndcg_across_users(quary_scores: List[Union[float, int]], k: int = None, method=0) -> float:
```

> Распределение баллов

- **0%** – корректность формата на входе/выходе (API). Если тест не пройден, за всё решение 0 баллов.
- **60%** — проверка корректности вашей метрики.
- **20%** – скорость работы (полный балл, если ваше решение выполняется быстрее чем ...)
- **20%** — качество кода (Pylint Score; соблюдение PEP8)