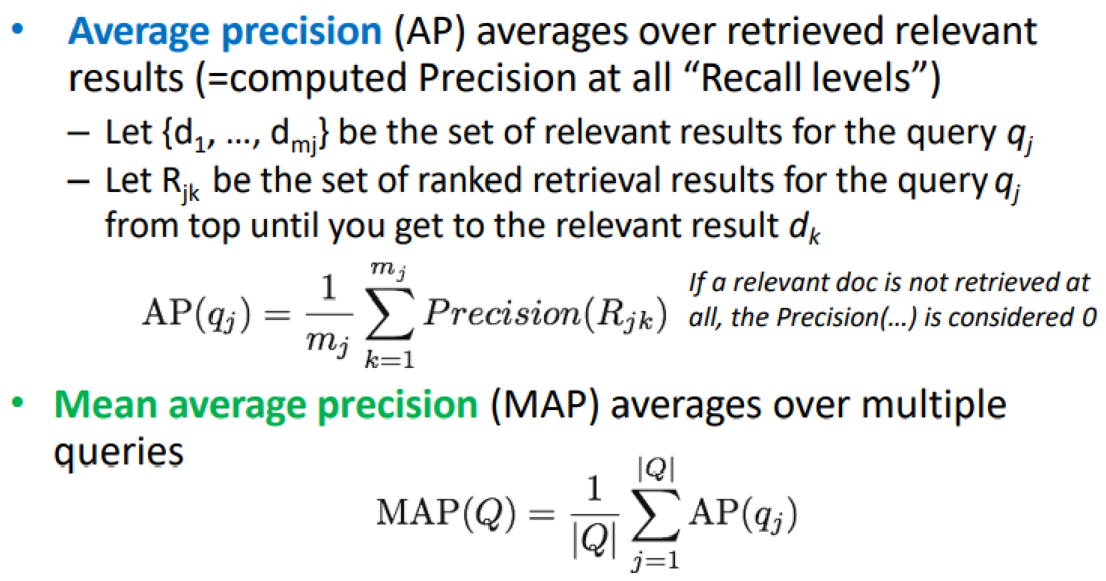
Experiment3: IR Evaluation

Mean Average Precision (MAP)
计算 p@k,
计算 p@k 均值（AP）
计算 AP 均值（MAP）



- **Average precision** (AP) averages over retrieved relevant results (=computed Precision at all "Recall levels")
  - Let $\{d_1, \ldots, d_{mj}\}$ be the set of relevant results for the query q_j
  - Let R_{jk} be the set of ranked retrieval results for the query q_j from top until you get to the relevant result d_k

$$\mathrm{AP}(q_j) = \frac{1}{m_j}\sum_{k=1}^{m_j} Precision(R_{jk})$$

If a relevant doc is not retrieved at all, the Precision(...) is considered 0

- **Mean average precision** (MAP) averages over multiple queries

$$\mathrm{MAP}(Q) = \frac{1}{|Q|}\sum_{j=1}^{|Q|} \mathrm{AP}(q_j)$$

```
for doc_id in test_result[0: length_use]:
            #计算 P@K i_retrieval_true 为当前正确的文档数目  i 为当前文档数
            i += 1
            if doc_id in true_list:
                i_retrieval_true += 1
                P_result.append(i_retrieval_true / i)
                #print(i_retrieval_true / i)
        if P_result:
            AP = np.sum(P_result) / len(true_list)
            #计算 average of P@K  计算 p_result 中的 p@K/正确文档数
            print('query:', query, ',AP=', AP)
            AP_result.append(AP)
        else:
            AP_result.append(0)
```

Mean Reciprocal Rank (MRR)
计算第一个（名次位置为 K）相关文档，1/K（RR）
计算 RR 均值（MRR）

- Consider rank position, K, of first relevant doc
    - Could be – only clicked doc

- Reciprocal Rank score = $\frac{1}{K}$

- MRR is the mean RR across multiple queries

```
for doc_id in test_result[0: length_use]:
            i += 1
            if doc_id in true_list:
                #找到第一个相关文档，名次位置为 K
                i_retrieval_true = 1
                P_result.append(i_retrieval_true / i)
                #计算 1/K
                break
                #print(i_retrieval_true / i)
        if P_result:
            RR = np.sum(P_result)/1.0
            print('query:', query, ',RR=', RR)
            RR_result.append(RR)
        else:
            RR_result.append(0)
```

Normalized Discounted Cumulative Gain(NDCG)
计算 DCG 为 rel1 加上所有 reli/log2(i)的总和
用 sort 排序后的文档名次位置计算 IDCG
NDCG=DCG/IDCG
因为 log2(i)当 i=1 时取值为 0，采用将 i+1

- **Discounted Cumulative Gain:**

$$\mathrm{DCG}_n = rel_1 + \sum_{i=2}^{n} \frac{rel_i}{\log_2 i}$$

```
for doc_id in test_result[0: length_use]:
```

```
                i += 2
                rel = qrels_dict[query].get(doc_id, 0)
                DCG += rel[i] / math.log(i, 2)
                IDCG += true_list[i] / math.log(i, 2)
            DCG = DCG + rel[1]
            IDCG = IDCG +true_list[1]
            NDCG = DCG / IDCG
            print('query', query, ', NDCG:=', NDCG)
            NDCG_result.append(NDCG)
```

代码

```
import math
import numpy as np

def generate_tweetid_gain(file_name):
    qrels_dict = {}
    with open(file_name, 'r', errors='ignore') as f:
        for line in f:
            ele = line.strip().split(' ')
            if ele[0] not in qrels_dict:
                qrels_dict[ele[0]] = {}
            if int(ele[3]) > 0:
                qrels_dict[ele[0]][ele[2]] = int(ele[3])
    return qrels_dict
def read_tweetid_test(file_name):
    test_dict = {}
    with open(file_name, 'r', errors='ignore') as f:
        for line in f:
            ele = line.strip().split(' ')
            if ele[0] not in test_dict:
                test_dict[ele[0]] = []
            test_dict[ele[0]].append(ele[1])
    return test_dict
#qrels_dict 为真实的 idset,test_dict 为测试的 idset

def MAP(qrels_dict, test_dict, k = 100):
    AP_result = []
    for query in qrels_dict:
        test_result = test_dict[query]
        true_list = set(qrels_dict[query].keys())
        length_use = min(k, len(test_result))
        if length_use <= 0:
            return []
        P_result = []
```

```
            i = 0
            i_retrieval_true = 0

            for doc_id in test_result[0: length_use]:
                #计算 P@K i_retrieval_true 为当前正确的文档数目  i 为当前文档数
                i += 1
                if doc_id in true_list:
                    i_retrieval_true += 1
                    P_result.append(i_retrieval_true / i)
                    #print(i_retrieval_true / i)
            if P_result:
                AP = np.sum(P_result) / len(true_list)
                #计算 average of P@K  计算 p_result 中的 p@K/正确文档数
                print('query:', query, ',AP=', AP)
                AP_result.append(AP)
            else:
                AP_result.append(0)
        return np.mean(AP_result)

    def MRR(qrels_dict, test_dict, k = 100):
        RR_result = []
        for query in qrels_dict:
            test_result = test_dict[query]
            true_list = set(qrels_dict[query].keys())
            length_use = min(k, len(test_result))
            if length_use <= 0:
                return []
            P_result = []
            i = 0
            i_retrieval_true = 0

            for doc_id in test_result[0: length_use]:
                i += 1
                if doc_id in true_list:
                    #找到第一个相关文档，名次位置为 K
                    i_retrieval_true = 1
                    P_result.append(i_retrieval_true / i)
                    #计算 1/K
                    break
            if P_result:
                RR = np.sum(P_result)/1.0
                print('query:', query, ',RR=', RR)
                RR_result.append(RR)
            else:
                RR_result.append(0)
        return np.mean(RR_result)

    def NDCG(qrels_dict, test_dict, k = 100):
        NDCG_result = []
        for query in qrels_dict:
            test_result = test_dict[query]
            true_list = list(qrels_dict[query].values())
            true_list = sorted(true_list, reverse=True)
            i = 1
            DCG = 0.0
            IDCG = 0.0
            length_use = min(k, len(test_result), len(true_list))
            if length_use <= 0:
                return []

            for doc_id in test_result[0: length_use]:
                i += 1
                ################################
                rel = qrels_dict[query].get(doc_id, 0)
                DCG += rel[i] / math.log(i, 2)
                IDCG += true_list[i+1] / math.log(i, 2)
            DCG = DCG + rel[1]
            IDCG = IDCG +true_list[1]
            NDCG = DCG / IDCG
            print('query', query, ', NDCG: ', NDCG)
            NDCG_result.append(NDCG)
        return np.mean(NDCG_result)

    def evaluation():
        k = 100
        file_qrels_path = 'qrels.txt'
        qrels_dict = generate_tweetid_gain(file_qrels_path)
        file_test_path = 'result.txt'
        test_dict = read_tweetid_test(file_test_path)
        map = MAP(qrels_dict, test_dict, k)
        print('map', ' = ', map, sep='')
        mrr = MRR(qrels_dict, test_dict, k)
        print('mrr', ' = ', mrr, sep='')
        ndcg = NDCG(qrels_dict, test_dict, k)
        print('ndcg', ' = ', ndcg, sep='')
    if __name__ == '__main__':
        evaluation()
```