Document-Domain Resampling Framework for Text Retrieval and Natural Language Processing

CS410 Final Project Report

Keh-Harng Feng
Department of Computer Science, University of Illinois - Urbana Champaign

December 18, 2017

1 Background and Motivation

Overfitting is perhaps the single most crucial problem for model builders. The performance of a model fitting strategy on unseen data is a direct indication of the model's predictive power and hence its usefulness in the wild. However, without an unbiased model evaluation method it is difficult to generalize the model performance obtained on existing data. To that end, resampling methods are often used to mitigate biased evaluation that can be potentially introduced by model overfitting. For a more detailed summary on resampling methods, please refer to the Technology Review.

Unfortunately, existing implementations of resampling methods in popular machine-learning toolkits such as scikit-learn and caret are often ill-suited for text retrieval and NLP data in general. This is due to NLP data organization commonly adopts the separation of labeled responses (query judgments) from data features (text corpus). This project implements a proof-of-concept framework for resampling methods suitable for data in the line-corpus format along with separate query judgments that are frequently encountered in natural language processing.

2 Implementation

As the project is a proof-of-concept to demonstrate the use of resampling methods in the context of text retrieval, the implementation aims to

1. Make use of basic, common libraries as much as possible to keep the foundation of the framework largely independent of any particular NLP toolkit.
2. Prefer generalizability over speed. Thus even if parts of the framework make use of a specific toolkit, it can still be quickly rewritten using another toolkit with minimum changes to the code flow.
3. Prioritize simplicity over safety. This means the code should be relatively easy to follow. However it is not memory safe when the input data is very large and is therefore not suitable for use in a production environment.

The framework is implemented in Python. It can be roughly split into three parts - resampling, evaluation and testing.

2.1 Resampling Layer

The resampling layer deals only with the generation of resampled data folds from existing data and is largely toolkit neutral. It uses built-in and common libraries such as numpy to provide low-level functions that serve as generic data manipulation tools, data parsers and construction blocks for different types of resampling methods. Both k-fold cross validation and bootstrap are implemented, although These functions are implemented in `base.py` and `base_helpers.py`. At its core, a resampling method can be constructed by:

1. Sample the indices of documents from a line corpus to be included in the test fold, with or without replacement.
2. Generate training fold by taking the complement of the test fold from the entire dataset.
3. Translate sampled indices into actual data partitions by generating new corpuses containing only the documents according to the sampled indices. Also generate support files such as associated configs and copy over the queries and relevance files. This makes each fold more or less a standalone dataset.

Complication arises from the fact that often the query judgments are stored in a separate file that contains document IDs/indices linked to the associated corpus. By resampling the corpus the document indices in the resampled fold will no longer correspond to the correct relevance data. The framework resolves this by generating a document ID mapping for every resampled fold. This allows user to map the new document ID in the resampled corpus back to its original ID through reverse lookup and hence still retrieve the query relevance data corresponding to it.

2.2 Evaluation Layer

The evaluation layer provides methods that generalize common text retrieval evaluation schemes to allow unbiased evaluation using train and test data split. It makes use of the *metapy* interface to the MeTA toolkit for NLP functionalities and pymp for parallel processing.

Unfortunately MeTA's built-in text retrieval functionalities do not have the facilities to directly enable train/test split and model predictions on unseen data. This mechanism is crucial for resampling methods, as each test fold acts as a set of unseen data and evaluation must be carried out **only** on the test fold. A modular evaluation framework and a customized BM25 ranker is thus written to facilitate both rank score and evaluation score computation with train/test split. These are implemented in the files `eval.py` and `eval_metapy.py`.

The standard Okapi BM25 scoring function is shown below:

$f(q,d) = \sum_{w \in q \cap d)} c(w,q) \frac{(k+1)c(w,d)}{c(w,d)+k(1-b+b\frac{|d|}{avdl})} log \frac{M+1}{df(w)}$,

$b \in [0,1]$

$k \in [0,+\infty]$

It is important to note that

1. |d|, c(w,d), c(w,q) are all attributes linked only to the document/query and independent of the training corpus.
2. avdl, M, df(w) are attributes linked to the training corpus.

It stands to reason that all attributes in 2 must be determined using a combination of the training corpus and the specific document being predicted by the ranker and nothing else. The evaluation framework and the customized BM25 accomplishes this by allowing user to specify both a test and train corpus. While all documents ranked are extracted from the test corpus one by one, all required attributes are computed using either the document/query themselves (those in 1) or in conjunction with the train corpus (those in 2). If it is a document in the latter case, modifications are made appropriately (ie: M is the size of the training corpus + 1 due to the addition of the document being predicted).

Similarly, a customized logarithmic NDCG evaluator[1] is written to generate the NDCG score of a set of retrieved document with a cutoff at 10, measured against the test fold (ie: the source corpus of the documents being ranked and retrieved). All relevant attributes are computed using only the test fold corpus when applicable to account for the test/train split.

2.3 Test Layer

The test layer provides template codes to demonstrate a common use case with cross-validation and to generate evaluation data to study the method's efficacy. More information can be found in the **Testing Methodology** section.

3 Testing Methodology

A large corpus apnews containing 164464 documents is split into two disjoint sets. One of the set (fold_1) is further split into five disjoint sets to represent five different training sets. This can be found in `test_data_prep.py`. Some modifications to the config files are necessary. As such some of the files are included in the repository for reproducibility.

A suite of different evaluations are then carried out on these five sets:

1. 5-fold cross validation.
2. Training set evaluation (ie: reported training error).
3. Test set evaluation (ie: reported test error on fold_2).

Two model fitting strategies are tested. The first one is the vanilla BM25 model with parameters k = 1 and b = 0.5 (seen in `test_cv_eval.py`, `test_training_eval.py` and `test_testing_eval.py`). The second one is a modified BM25 model that uses a 3x3 linear grid search on the range $k \in [1, 2.5]$ and $b \in [0, 0.5]$. Normally the grid search should be carried out on each training set (ie: for CV it should search each training fold as opposed to the entire training set presplit). However to minimize computation time a decision is made to use the result of the grid search on each of the independent training set. This allows a comparison of the bias of the training set evaluation and 5-fold CV evaluation using the test set evaluation as the true model performance.

4 Result and Discussion

Figure 1 shows the arithmetic mean NDCG computed with different evaluation strategies using both regular BM25 with default values (k = 1, b = 0.5) and grid-search optimized BM25. Surprisingly, the training set performances of both models are closer to the test set performance compared to the evaluation obtained through 5-fold cross-validation. In the case of the regular BM25, the training set performance is positively biased compared to the test set as expected. However, in the case of the grid-search optimized BM25, the test set performance in fact outstrips even the training set while the CV-estimated performance stays roughly identical to that of the untuned BM25. This implies two things:

1. BM25 as a model is not very prone to overfitting, even with grid-optimized model parameter tuning.
2. There is a hidden bottleneck that is causing consistent overestimation in prediction error from the resampled folds and to a much lesser extent, the complete training set.

While cross-validation and resampling methods in general indeed tend to overestimate model error[2], the bottleneck is different in the sense that it manifests itself even using reported training error. Thus it is likely not directly connected to the use of resampling methods but the side-effect of an unforeseen complication. The NDCG per query is analyzed to pin-point the culprit. A proportionality quantity is computed from the number of relevant documents contained in the test fold used for evaluation divided by 10 (to a maximum of 1). This quantity shall be refered to as Pro10 from this point on. Since the NDCG in this study is computed with a constant cutoff at 10, Pro10 roughly represents the maximum NDCG a query can possibly obtain from using the specified performance estimator (training or CV) and a perfect retrieval model. If Pro10 is low, then max NDCG is bottlenecked at a low value and the end result will likely deviate from the NDCG computed from the much larger test set due to the lack of relevant documents included in the test fold rather than actual model performance.

Figure 2, Figure 3, Figure 4 and Figure 5 show the difference between test set NDCG and training/CV NDCG per query for each of the five training set. The data points are color-coded using Pro10 for each query. It is clear that

1. The full training set contains more data points with high Pro10.
2. Data with high Pro10 values tend to be less scattered around 0.

With this in mind it is likely the culprit for the unepxectedly large negative bias from cross validation and strangely high true model performance with grid-search is the dramatic size difference between the test corpus compared to each of the training set and resampled fold. With a total of 164464 documents, the test corpus contains roughly 80000 documents. In contrast, each of the five training set only contains about 16000 documents, while each of the test folds during cross-validation contains only 3200 documents. It is therefore very likely for model performance on any test folds to become severely bottlenecked due to the lack of documents that are judged to be relevant in the test folds.

It may be possible to combat this phenomenon with the use of stratified sampling during data split and dynamic cutoff resizing. The key is to ensure that the resampled test folds contain roughly same amount of relevant documents for each query proportionally and NDCG cutoff value is shrunk accordingly to the inevitably reduced number of relevant documents after resampling.

5 Conclusion

A document-domain resampling framework is implemented in Python. While the design emphasis on flexibility and robustness allows multiple types of resampling methods to be quickly built and interfaced with different NLP toolkits, preliminary analysis indicates a large pessimistic bias due to the lack of consideration with regards to the inevitable loss of relevant documents during the resampling process. As it stands it is not a useful production tool. However it may be possible to overcome the shortcomings with additional safeguards in the data sampling and evaluation process.

6 Future Work

1. General performance optimization (Ex: minimizing disk access using memoryview or multiplicity index, more parallelism) in order to enable use of bootstrap in realistic scenarios.
2. Implementation of stratified sampling and dynamic cutoff.

7 Reference

1. Järvelin K., Kekäläinen J., Cumulated Gain-Based Evaluation of IR Techniques, ACM Transactions on Information Systems (TOIS): Volume 20 Issue 4, October 2002
2. Kohavi, R., A Study of Cross-Validation and Bootstrap for Accuracy Estimation and Model Selection, International Joint Conference on Artificial Intelligence (IJCAI), 1995

8 Appendix

This is a reproducible report. All codes for data analysis and plot production are available for audit in the repository and can be run to regenerate the report.

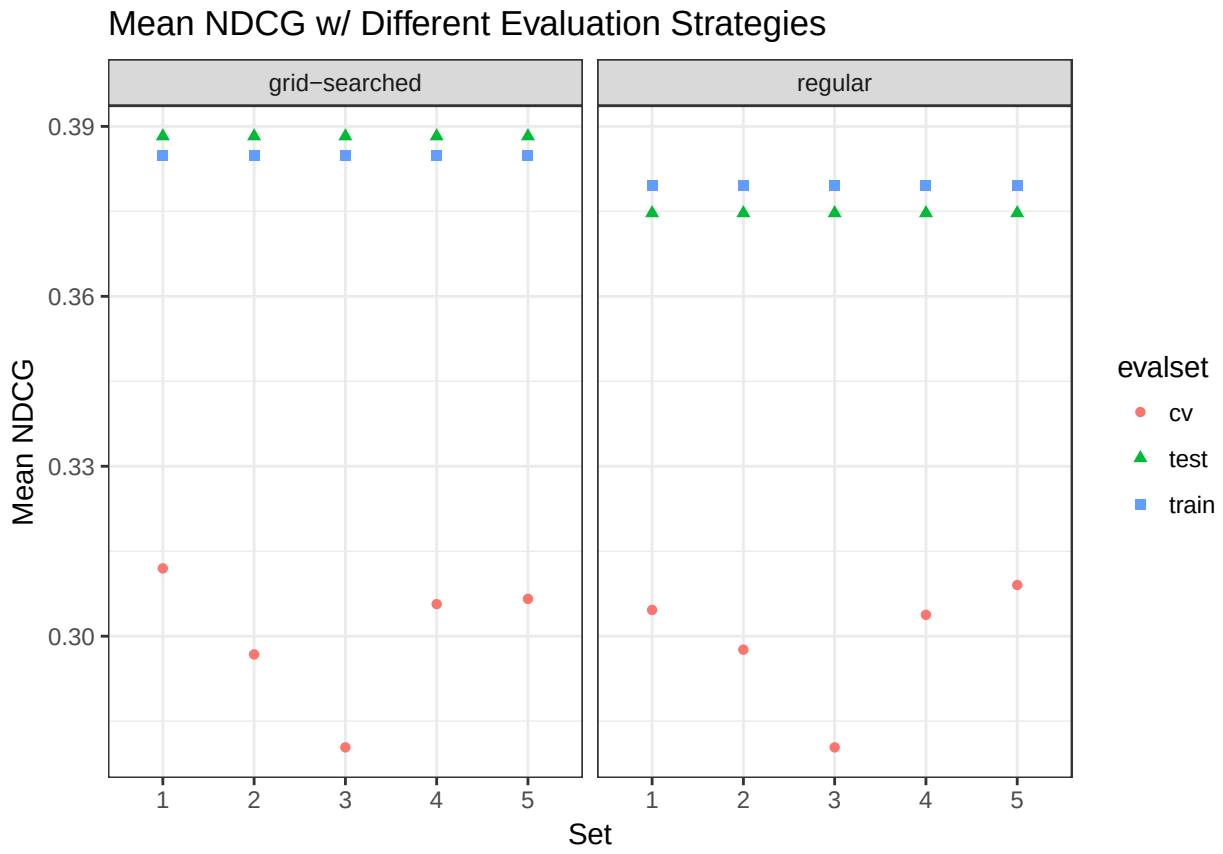

Figure 1: Mean NDCG computed using training set, 5-fold cross validation and test set evaluation.

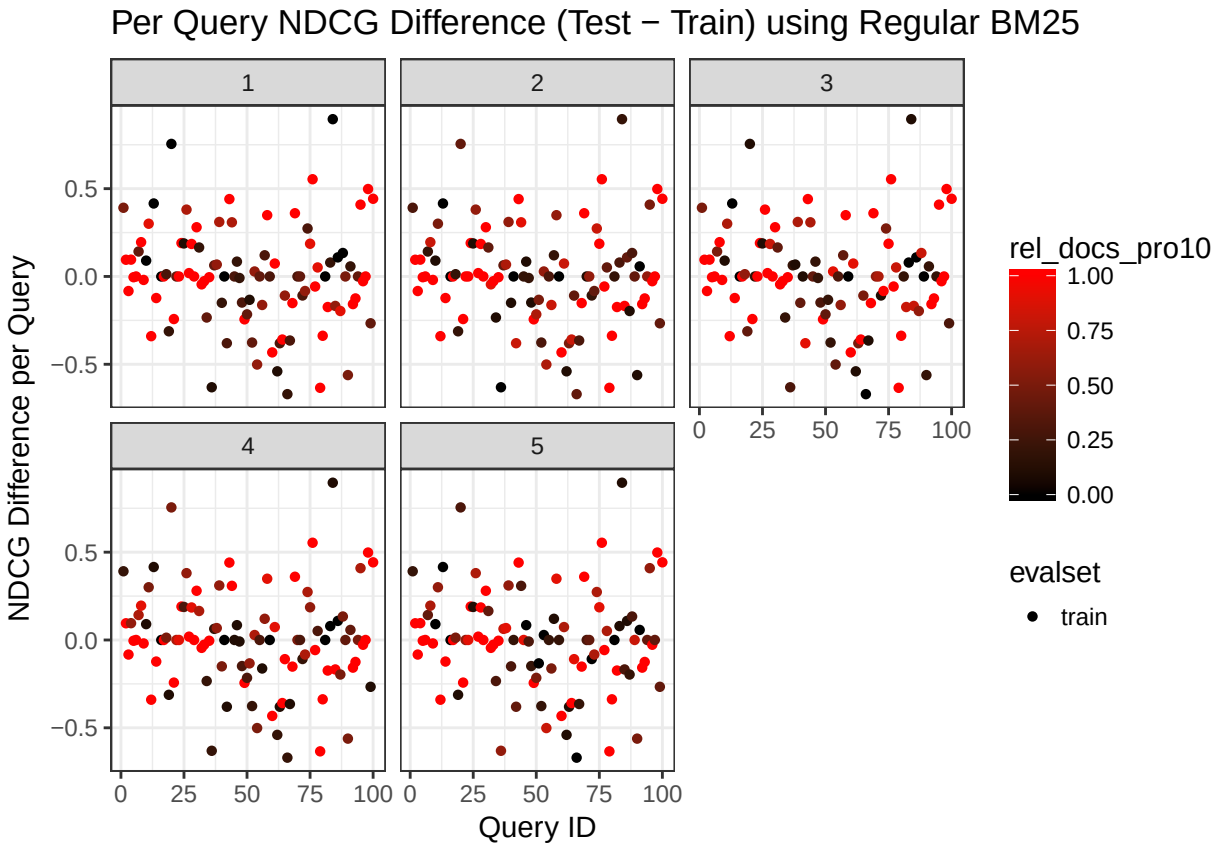

Figure 2: NDCG difference (test set - training) per query computed using regular BM25. Each facet shows result from a specific training set.

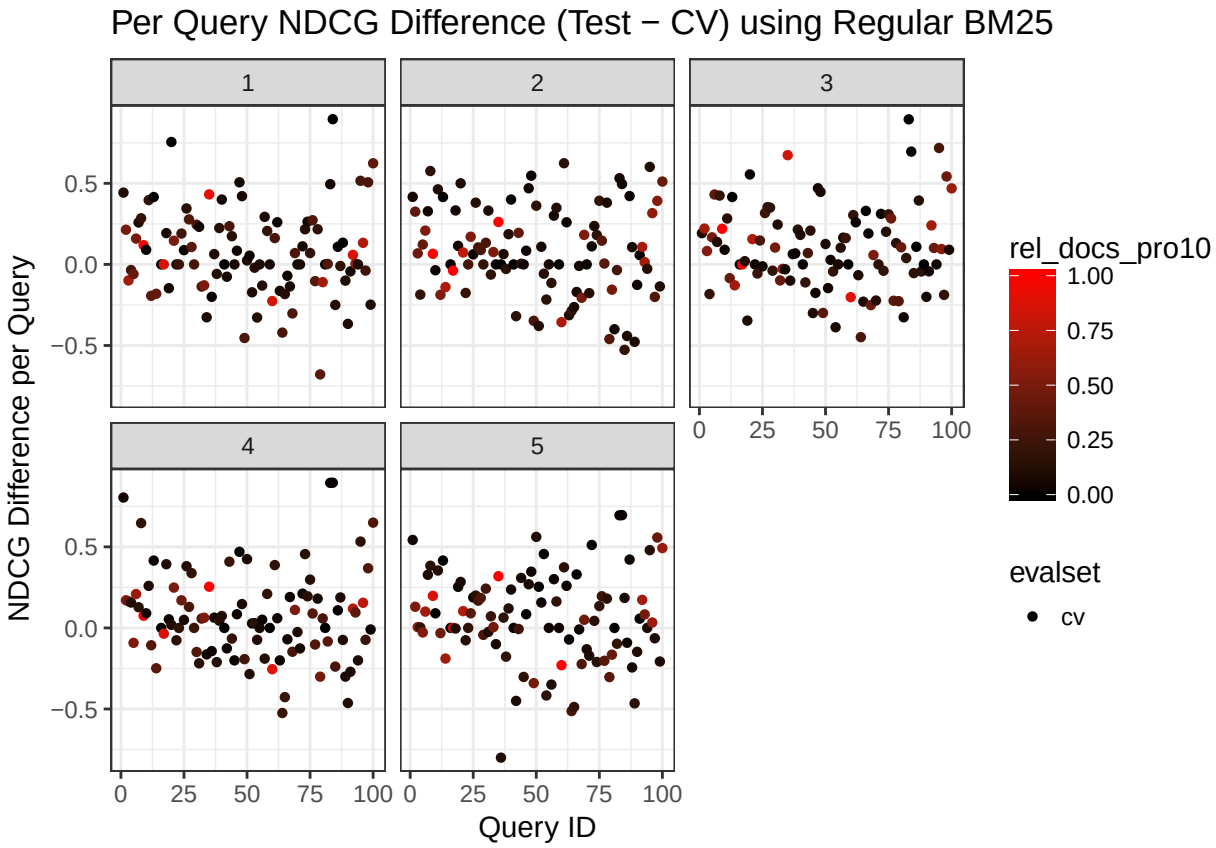

Figure 3: NDCG difference (test set - cv) per query computed using regular BM25. Each facet shows result from a specific training set.

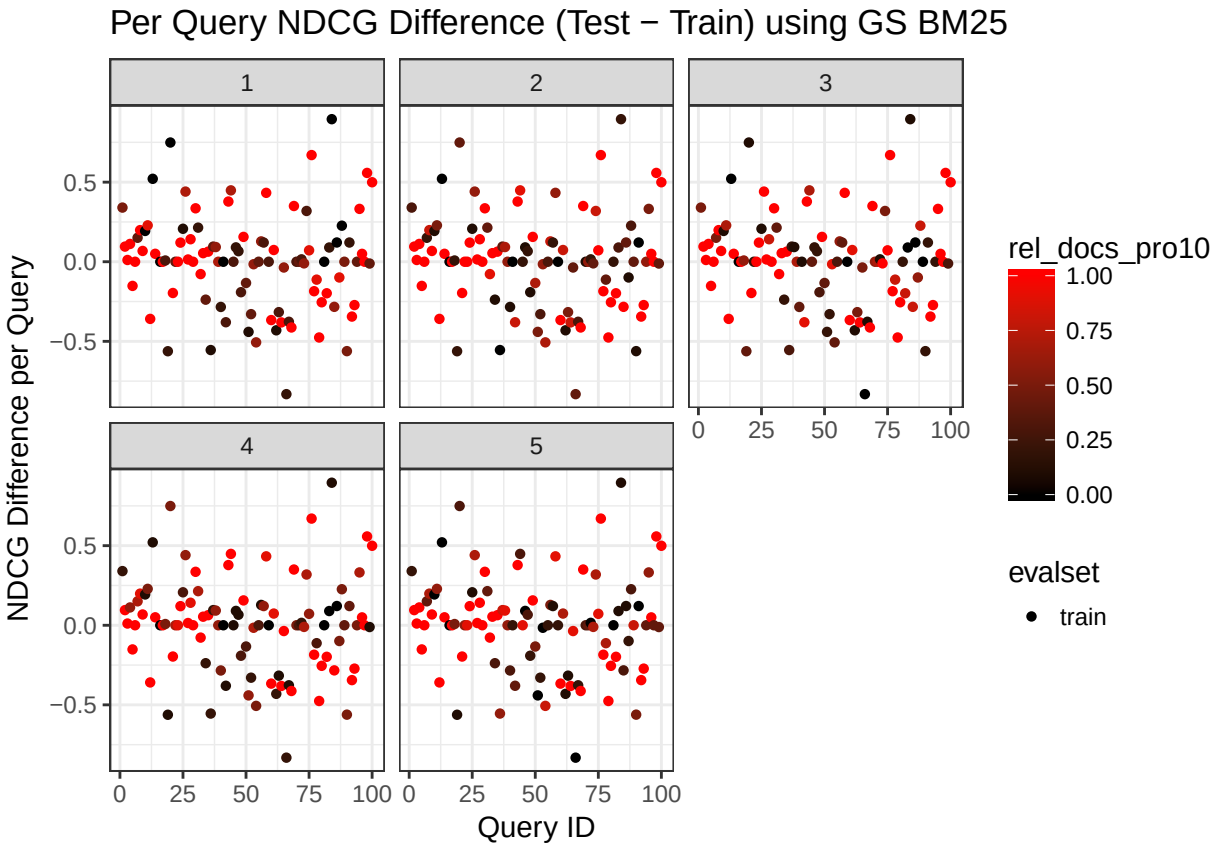

Figure 4: NDCG difference (test set - training set) per query computed using grid search BM25. Each facet shows result from a specific training set.

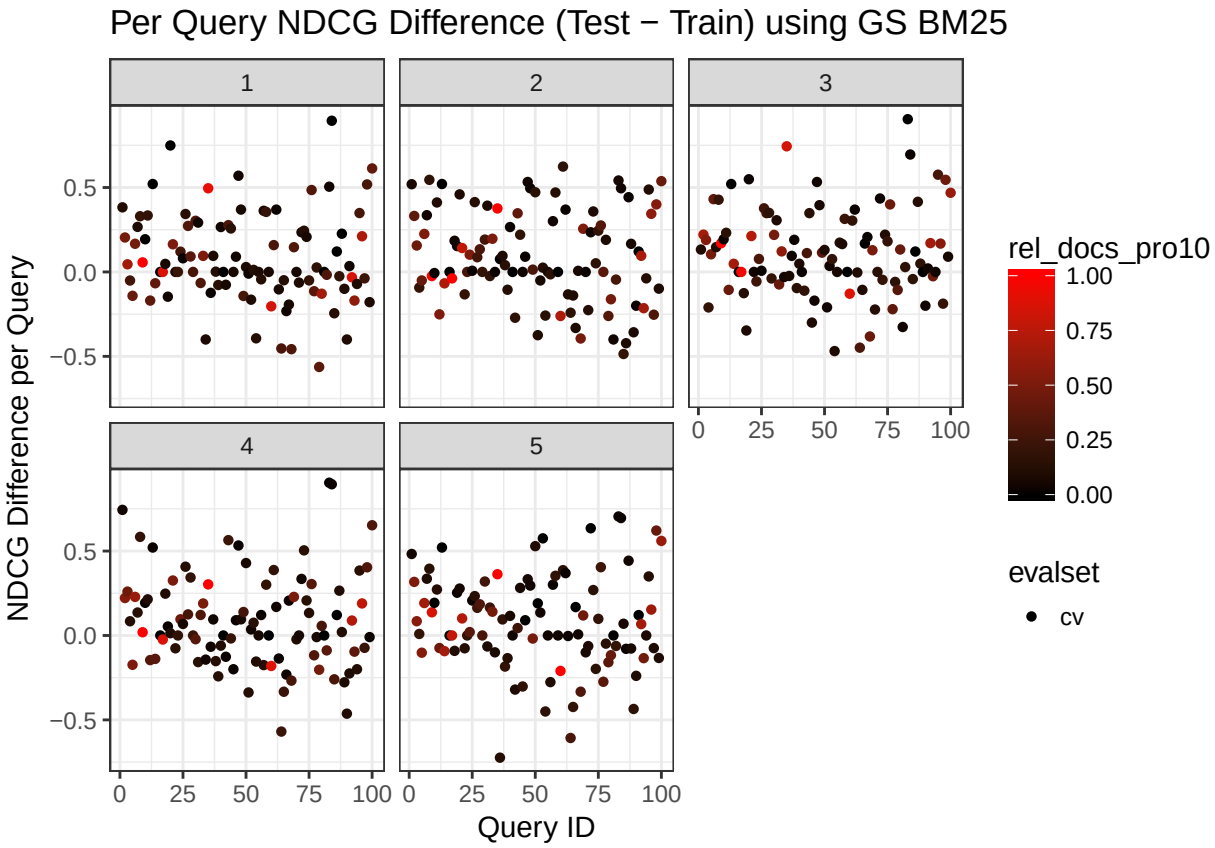

Figure 5: NDCG difference (test set - cv) per query computed using grid search BM25. Each facet shows result from a specific training set.