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 Abstract

2 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.

3 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. Prioritize generalizability over speed. Thus even if parts of the framework makes 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.

3.1 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 come 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.

3.2 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 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 NDCG evaluator is written to generate the NDCG score of a set of retrieved document, measured against the training set.

3.3 Test Layer

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

4 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`.

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 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.

5 Result and Discussion

6 Conclusion

7 Future Work