# Task 1:

The first task was fairly straight-forward. My first bug-free implementation of the linear regression classifier using Tf-Idf features (sublinear scaling, no normalization) gave a NDCG score around 0.832 on the dev dataset. As a result of the changes I made to the Tf-Idf features as a part of task 2, the NDCG score rose to 0.857189.



# Task 2:

* My first implementation of task 2 included standardizing the dataset before taking the difference in features of two documents. It gave a NDCG score of about 0.834 & 0.837 for linear & RBF kernels respectively on the dev set.
* I started to focus on RBF kernel and thanks to the update to the pdf which asks to equally distribute the data, a small change gave a NDCG score around 0.839.
* I experimented on the code to standardize the dataset after taking the difference in features of two documents. This, along with a few modifications, gave me a NDCG score of about 0.843
* On comparing my ranked output with the actual relevance scores, I found that huge documents took precedence over small documents due to lack of normalization. But normalizing all the fields by using the body length decreased the NDCG score. Hence I normalized just the body with its length. This gave me a NDCG score around 0.848
* The above score gain encouraged me to normalize the title and header fields with its respective lengths thus giving a score about 0.853
* Using stemmers gave a score boost in PA3. Hence I decided to use them to stem terms in all fields except URLs as the terms in URLs were mostly not dictionary terms. This gave a NDCG score of about 0.8545.
* On analyzing the values of the URLs, I realized that just tokenizing them on non-alphanumeric characters is not the right thing to do. Tokenizing on non-alphanumeric characters and changes in cases (e.g.: “StanfordUniversity” -> “stanford” “university”) gave better results but there were tokens which were two words concatenated together with no case changes. As there was no good way to tokenize them, I started to tokenize on non-alphanumeric characters and checked whether a term was a subset of another term rather than comparing whether two terms are equal (e.g.: “stanforduniversity”.contains(“Stanford”)). This gave me a NDCG score of about 0.858
* Encouraged by the score above, I started to check whether terms were a substring of each other in all the fields instead of checking whether they are equal. This finally gave me a score of 0.85823 & 0.8601 for linear & RBF kernels respectively on the dev set.
* Performing a grid search on the value ranges provided in the pdf gave decreased score of 0.85948 for RBF kernel. Hence I moved back to default values.



# Task 3:

* I implemented BM25 feature using my manual weights form PA3. After a failed attempt of improving scores by experimenting with a couple of field weights and K1 values, I planned to use the coefficients from the linear regression classifier from Task 1 as the field weights. This increased my NDCG score.
* Next, I implemented the page rank feature. After experimenting with log, saturation and sigmoid models and altering its lambda prime values, I settled on the log model.
* I included the smallest window in the entire document as a feature. Unfortunately, this deceased my scores. Tried to exponentially scale the inverse of window size but no luck.
* Instead of finding the smallest window smallest window across the entire document, I introduced five features representing the smallest window in the five fields of a document. This increased my scores.





# Task 4 (Extra credit):

* Extension 1: Used the RankLib library to implement a pairwise neural net ranker using RankNet.
* Extension 2: Made use of the word vectors provided to form features as follows:
  + Constructed the field vectors of a document as the sum of the vectors of the terms in the field scaled sublinearly.
  + Constructed the query vector as the sum of the vectors of the terms in the field scaled sublinearly and weighted using the idf values.
  + The feature value of each field is the dot product of the field vector and the query vector.



# Error Analysis:

Analyzed the ranked output to the actual relevance files as a part of both Task 2 & 3 and tried a couple of ideas.

Ideas that worked:

* Normalizing body, title and headers with their lengths.
* Better tokenization of URLs.
* Using porter stemmer.
* Checking whether terms were a substring of a term rather than comparing whether they are equal.

Ideas that didn’t work:

* Introducing a new binary feature saying whether the document is a pdf or not.
* As pdfs had not headers as a part of it, tried cloning the title data to headers.
* Introduced the depth of the URL as a feature.