# Homework 2: QA Pipeline with Learning

## Goal:

The primary objectives of this assignment are 1. learn about the different modules in a Question Answering pipeline 2. Experiment with various learning algorithms on the different modules of the pipeline 3. Create a simple end-to-end QA system using a real dataset called Quasar-S.

## Dataset:

The dataset called Quasar-S is a set of 37,000 cloze-style question answers built from the definition of tags in StackOverflow. The dataset is presented to us in a JSON format and restructured into a dictionary as follows: Each record consists of the (i) origin (ii) questions: This is a list of dictionaries. Each element in the list corresponds to the information for a single question. Each question is accompanied by short and/or long snippets. (iii) candidates: A list of potential answer candidates for this dataset is also provided.

## Pipeline:

The stages of the pipeline are:

1) **Parsing** – The data provided was already parsed into training set, dev set and test set. Each of these sets were segmented into the samples and their corresponding labels. Each sample consisted of a question and candidate answers (labels) , essentially tags from the answers in Stackoverflow. The code to format and preprocess the train and dev set were provided.

2) **Pre-processing + Retrieval –** This part involves fetching the long snippets and short snippets of questions.

3) **Featurizer –** This stage models the samples into a set of features for the classification task. To learn the samples, it is necessary to convert the text data to numerical form. The most common ways to do this are: tokenizing strings and assigning numbers to each token, counting the token frequency, normalizing and diminishing the importance of tokens that occur very rarely. The final representation is called a feature vector. The different features experimented here are: (a) CountFeaturizer and (b) TfIdfFeaturizer.

    a. **CountFeaturizer:** Convert a collection of text documents to a matrix of token counts. It has various parameters like stop_words, ngram_range, lowercase, binary, etc

    b. **TfIdfFeaturizer:** Oftentimes, the frequency of unimportant words is much higher than words which are more meaningful. Because of their high frequencies, they

diminish the importance of other words. Therefore, we make use of the inverse document frequency (idf) along with the term frequency to weigh the tokens. Using this featurizer provides better scores than using the count vectorizer.

c. **HashVectorizer:** Uses hashing to find token string names to feature index mapping. Memory efficient technique for large datasets as it does not store vocabulary or dictionary in memory. Disadvantages: Cannot compute inverse transforms and IDFs.

4) **Classification –** In this step, we classify the samples to their respective classes based on a combination of their features. It is a supervised learning in which the system is required to assign a new instance to a category based on similarities with previous seen instances. The categories here are the various tags in StackOverflow answers. Here are the different classifiers used:

   a. **SVM Classifier –** A Support Vector Machine is a binary classifier that marks each sample as belonging to a class or not. For multiclass classification, it is viewed as a one-vs-rest classifier, where the labels are modeled as a one-hot-vector. It constructs a hyperplane or set of hyperplane in a high dimensional space. The SVM can be of non-linear kernels (Gaussian, rbf, poly, etc) or linear. I have used a linear SVM.

   b. **Multinomial Naïve Bayes Classifier –** This makes use of Bayes theorem to classify multinomially distributed data. The algorithm finds the probability of a token belonging to each class and assigns it to the class having highest probability. The function I have used, uses Laplace smoothing to account for features not present in training samples.

   c. **MLPClassifier –** It is a feed forward neural network of perceptrons on which an activation function is applied. The MultiLayerPerceptron that I have used consists of 3 hidden layers with 50, 40 and 30 neurons in the first, second, and third hidden layers. The activation that I have used is the sigmoid activation.

5) **Evaluation –** This block evaluates the results of the classification. It uses the sklearn metrics library to compute the precision, recall, accuracy and F measure. The metrics give a measure of how well the classifier labeled the samples. Precision – Fraction of relevant labels amongst the observed labels, Recall – fraction of relevant labels over the total number of possible labels, F1 measure - is the weighted average of the Precision and Recall, and accuracy gives the accuracy of classification

Below is a figure representing the flow of the data in the pipeline:

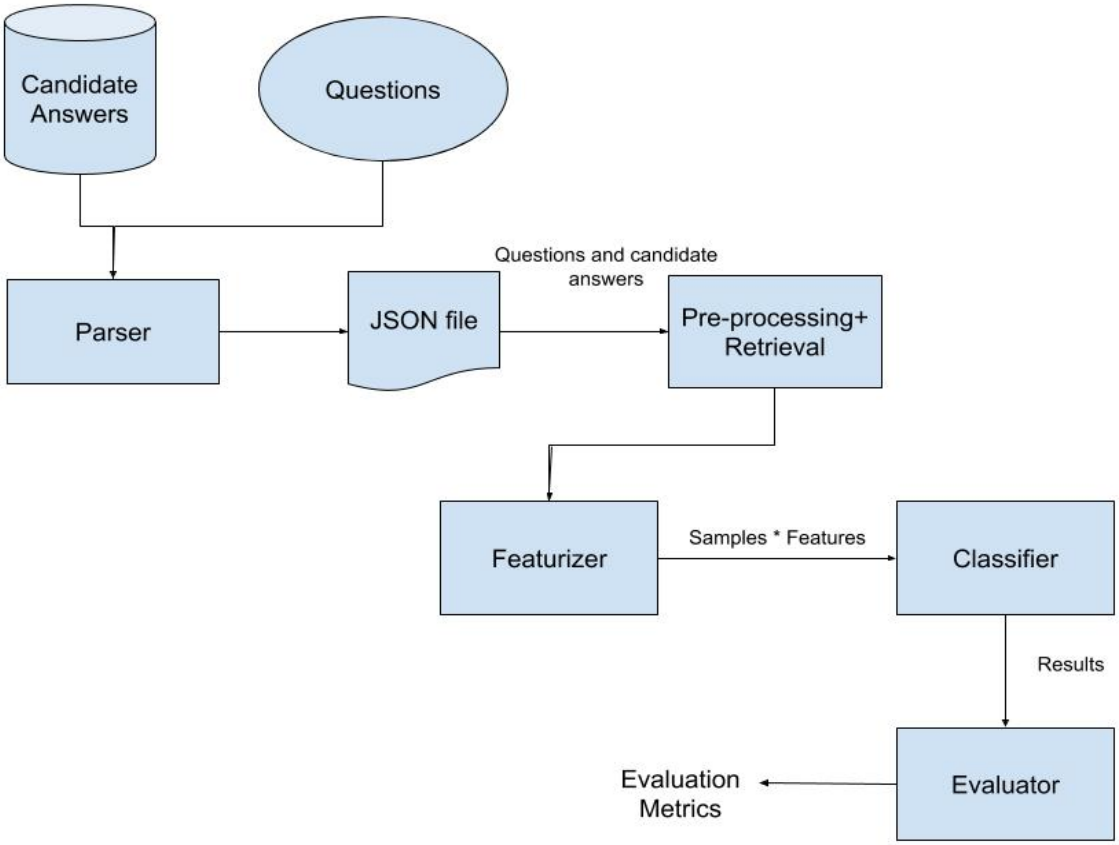


Figure 1: Block diagram of the QA Pipeline

## Experimental Results:

Below are the consolidated results of the runs of various featurizers and classifiers in the pipeline. I ran the pipeline for 3000 instances to increase the accuracy and other scores. A default of 10 samples gave poor baseline accuracy score of 0.0101.

| **Feature** | **Classifier** | **Accuracy** | **Precision** | **Recall** | **F-measure** |
|---|---|---|---|---|---|
| **Count based** | **MNB** | 0.057980 | 0.0029633 | 0.002124 | 0.001096 |
| **Count based** | **SVM** | 0.079324 | 0.019089 | 0.019681 | 0.016339 |

| | | | | | |
|---|---|---|---|---|---|
| **Count based** | **MLP** | 0.048104 | 0.012625 | 0.015101 | 0.012034 |
| **Tf-idf based** | **MNB** | 0.04237 | 0.000403 | 0.00125 | 3.2831E-04 |
| **Tf-Idf based** | **SVM** | 0.03458 | 3.0112E-05 | 0.000890 | 5.899E-05 |
| **Tf- Idf based** | **MLP** | 0.03409 | 3.0462E-05 | 0.000894 | 5.89162E-05 |
| **Hash based** | **MNB** | - | - | - | - |
| **Hash based** | **SVM** | 0.03458 | 3.0112E-05 | 0.000890 | 5.899E-05 |
| **Hash based** | **MLP** | 0.03408 | 3.0462E-05 | 0.000893 | 5.89162E-05 |

## Analysis:

### Count v/s Tf-Idf v/s Hash

The higher scores across all metrics – accuracy, precision, recall and f-measure for the Count based features than Tf-Idf based features is due to an imbalance of classes. The stopwords have not been removed from the datasets, and there are many special characters (could be belonging to math questions, or equations, or special character sequences in code) which pollute the dataset for a tf-idf based scoring of terms/tags. It also happens when there are words with high frequency that are very predictive of one of the classes.

Hashing gave similar results as TFIdf . It is much slower because each time we re-fit the vectoriser, it reloads the vocabulary into memory. It is les expensive with an increased risk of collisions in the hash map. The values for MNB classifier were not obtained because it could not compute the inverse transform (from feature indices to string feature names).

### MNB v/s SVM v/s MLP

Naive Bayes treats each token as independent token, whereas SVM looks at the interactions between them to a certain degree for classifying. We can see that the accuracy and precision is much higher for SVM in CountBased than for Bayesian classifier. It is harder to give a better comparison because MNB is a probabilistic model while SVM is geometric.

The SVM was much faster than the MLP, since it involves just computing which side of the hyperplane the point lies on. For an MLP, prediction requires successive multiplication of an

input vector by two 2D matrices (the weight matrices) for multiple layers. In MLP, network is adjusted such that the sum-of-square error between the network output and the actual value (target) is minimized, but for SVM it depends on deciding the side of decision boundary. The performance of the MLP largely depends on the network architecture. Due to the large corpus, a larger number of neurons at each hidden layer would have performed better classification. But due to memory constraints, I configured the hyperparamters to reflect a smaller set of neurons.

## Relative Performance Analysis:

I am going to compare the Count based model with SVM classifier and a tf-idf based MLP for their predicted labels. A value of 1 indicates that the prediction matched with the true label while a value of 0 indicates a mismatch with the true label.

**Let A = model with CountFeaturizer and SVM Classifier**

**B = model with TfIdfFeaturizer and MLP Classifier**

I wrote a script to compare the counts of the two models. Please find "script_compare.py" that takes file A.txt and B.txt as inputs.

| Input type(AB) | Counts |
|---|---|
| Easy Inputs (11) | 26 |
| Tough Inputs (00) | 2808 |
| Improvements (01) | 81 |
| Regressions (10) | 222 |

We see from the previous table that A is a better model than B in terms of accuracy, precision , recall and F-measure. Therefore a '1' on the weaker model indicates easiness and a '0' on the stronger model indicates that it was a tough input.

The easy inputs are those for which both the models gave the right labels. We see only 26 such labels.

The tough inputs are those for which both the models mismatched with the true label – we see a lot of such instances: 2808. This is why the overall accuracy and P,R,F scores are very low.

There is a slight improvement when the weaker model gives the right labels despite the stronger model giving wrong. We see 88 such instances.

The regression case is a larger case or expected case when the stronger model classifies correctly while the weaker model is wrong. There are 222 such instances.