Capstone Project Report

Juanyan Li

February 16, 2017

1 DEFINITION

1.1 PROJECT OVERVIEW

Stack Exchange is a network of Q&A communities with different topics. The problem I try to solve here is a simplified version of the original Kaggle competition: Transfer Learning on Stack Exchange Tags. The initial competition aims to explore the possibility of transferring knowledge learned from multi-tagged Stack Exchange questions to a new field. More specifically, it attempts to predict the tags of questions in the field of physics by learning tagged questions from other fields: biology, cooking, cryptography, diy, robotics and travel.

The simplified problem that are being solved in this project will be described in the next subsection. It utilizes the dataset provided by the competition. The dataset originates from the Stack Exchange data dump, containing questions w/ or w/o tags from the aforementioned seven topics.

Next subsection will clarify the problem the project is trying to solve.

1.2 PROBLEM STATEMENT

Different from the original Kaggle competition, the problem here instead focuces on the prediction of a topic each question belongs to. The classifier will be designed to learn the information mentioned in the title as well as the question and correlates it with the topic the question. The input for the classifier would be the context in the question and its output would be one of the seven categories (biology, cooking, cryptography, diy, robotics, travel and physics).

In order to build such a classifier, one must first decide features the classifier uses as the input. Appropriate features should be able to correctly represent the knowledge or information that resides in the each question. A baseline approach for feature engineering is proposed by using term frequency-inverse document frequency (tf-idf) in the field of information retrieval. It provides a straightforward vector representation of text data. In comparison, I implemented another feature engineering model called Distributed Memory Model of Paragraph Vectors (PV-DM, also known as "doc2vec") - a neural network based unsupervised method that learns the vector representation of paragraphs.

Once the mathematical features for the training data is acquired, the next step would be choosing the classification model (classifier). In the project, mainly two types of classification models are considered: Naive Bayes Classifier and Random Forest, since both of them are proved to be good classifiers in various tasks.

Technical details for each methods shall be revealed in the Analysis section.

1.3 METRICS

To obtain a comprehensive understanding of the performance of each feature-classifier combination, precision, recall and F1-score will be used as the evaluation metrics.

Precision evaluates the classifier's ability to correctly label samples with their true tags. It is calculated as follows:

$$Precision = \frac{tp}{tp + fp} \tag{1.1}$$

where tp is the number of true positives and fp is the number of false positives.

Recall, on the other hand, evaluates the classifier's ability to find positive labels for all samples. It is calculated as follows:

$$Recall = \frac{tp}{tp + fn} \tag{1.2}$$

where fn is the number of false negatives.

F-score attempts to give an overall evaluation of the classifier by aggregating both precision and recall. In this case, F_1-score is used which weights precision and recall equally:

$$F_1 = \frac{2}{\frac{1}{Precision} + \frac{1}{Recall}} = \frac{2 \cdot Precision \cdot Recall}{Precision + Recall} \tag{1.3}$$

To extend the calculated scores to a multi-class setting, two approaches are adopted and evaluated separately:

i) The first approach calculates metrics for each label and then find out unweighted means as the overall scores.

ii) The second one also calculates the metrics for individual labels first but then weights them by the number of true instances for each label to come up with the final scores.

The next section will give concrete analysis on the problem itself.

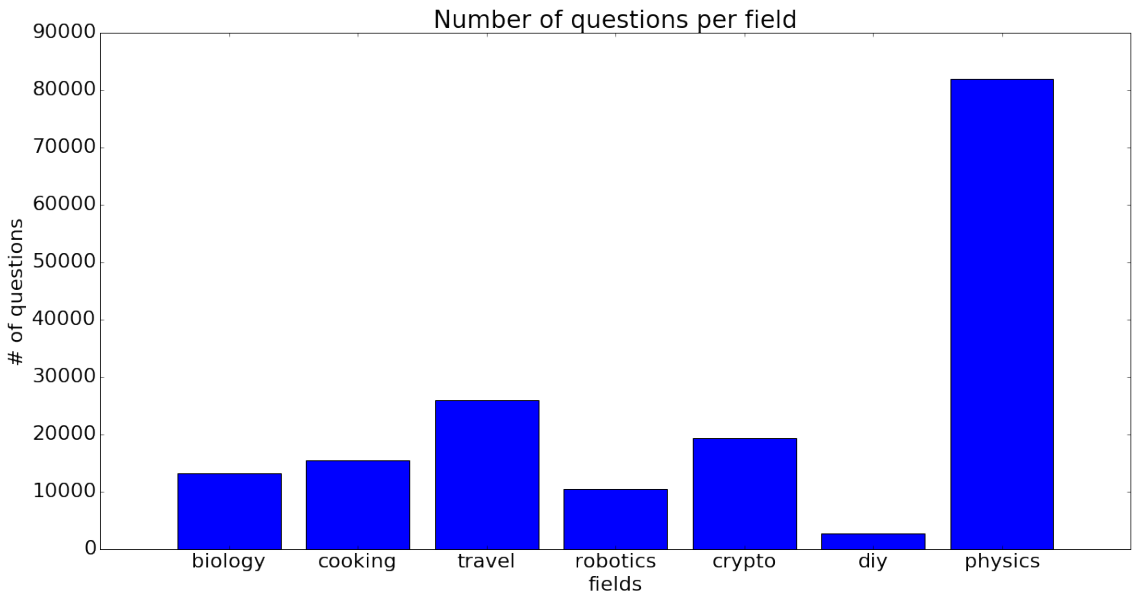
2 Analysis

In the first sub-section, a basic exploration of the dataset will be given, which mainly includes underlying statistics of the dataset. In the follow up section, methods for feature engineering (tf-idf and PV-DM) will be described in details. The last sub-section builds a baseline model using tf-idf as features and Gaussian Naive Bayes model as classifier. Results of the baseline model will be presented as well.

2.1 Data Exploration

As mentioned in the first section, the Stack Exchange dataset used in this project contains seven fields with a number of questions in each fields. See Figure 2.1. for the distribution of number of questions in each fields.

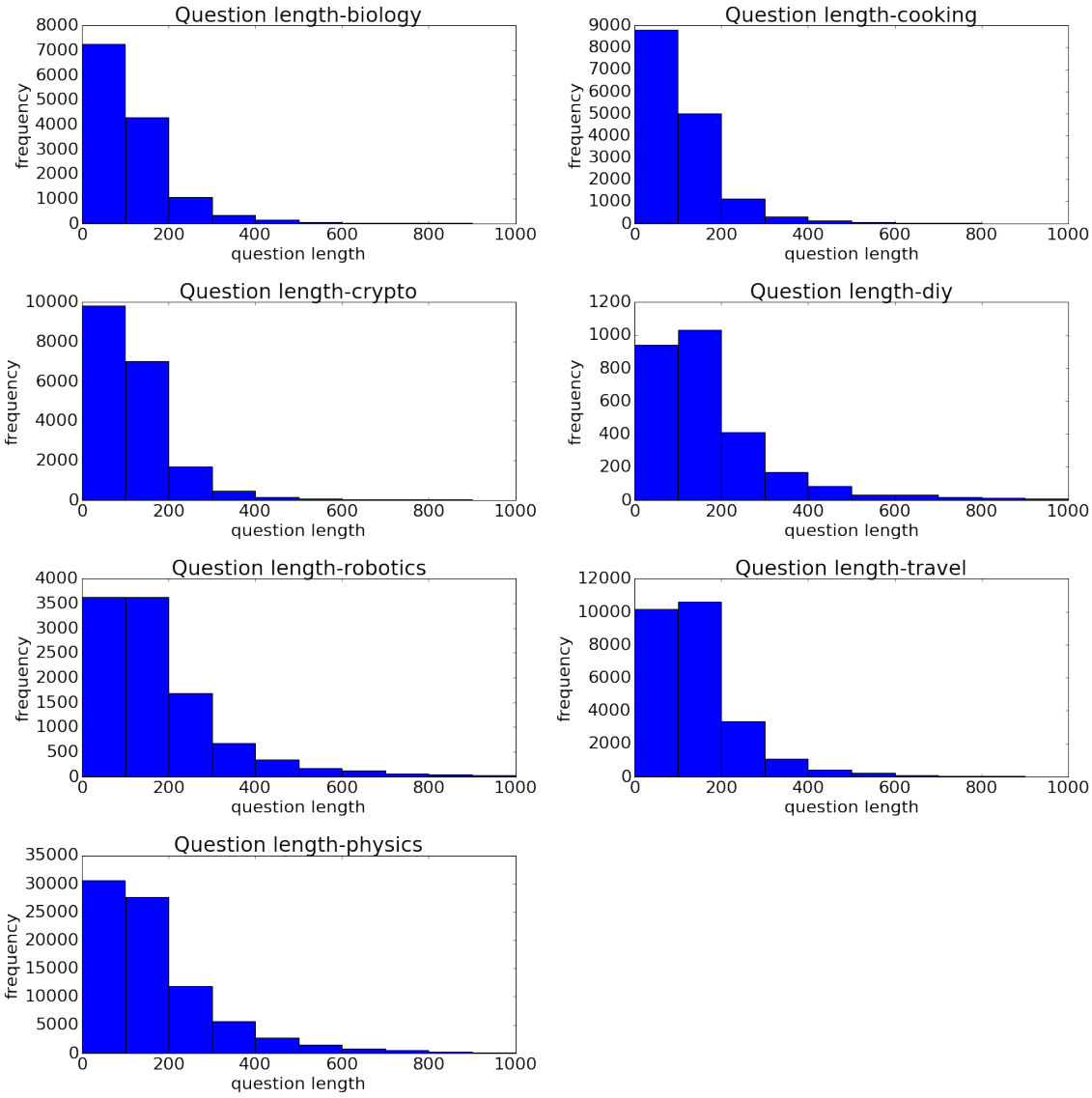
Figure 2.1:



Notice that the data heavily skews towards the physics field. A balanced dataset is preferred in order to avoid getting a biased classifier. This can be achieved by upsampling/downsampling in the training stage.

To take a further look into the dataset, a distribution of question length (number of tokens) is calculated based on each topic (See Figure 2.2). This gives an intuitive understanding of how much information is contained in each individual question.

Figure 2.2:



As can be seen above, most of the questions have less than 500 words/tokens. The figures shown above are capped to 1000 tokens. Actual distributions would have a much longer tail.

Those questions can be seen as outliers. Some of them are so because the author that raised the question pasted a large chunk of materials from other sources, which might not be useful. In the subsequent training stage, those outliers will be removed from the training samples.

2.2 Data Preprocessing

Before feature engineering, the data needs to be preprocessed first. General steps that are used in both TF-IDF and PV-DM are:

i) Data from all seven fields are loaded and combined into one dataframe in pandas;

ii) To simplified things further, all questions' titles and their contents are combined into one chunk of text;

iii) All html tags are removed from the raw combined content;

iv) Remove questions that have too many tokens (>1000).

For TF-IDF, which will be dicussed more in the next subsection, there is existing configurable functionality in scikit-learn. For "doc2vec" model, however, I implemented my own version on TensorFlow both for the purpose of learning and having more control on the training procedures.

2.2.1 Preprocessing for PV-DM

PV-DM essentially uses two large matrices to represent the embeddings of words as well as paragraphs. If the vocabulary for the words is of size V with embedding dimension d_w and the number of paragraphs in the training process is P with embedding dimension d_p, then the matrices' sizes are $V \times d_w$ and $P \times d_p$ respectively.

The first step to prepare inputs for PV-DM model is to build a vocabulary of words based on the training corpus. `UNKNOWN` and `NULL` symbols are also added for infrequent words and padding.

The next step is then to split the documents into sentences since I dont't want the sliding window to move across consecutive sentences. Each sentence belongs to a certain paragraph which is indexed with numbers for embeddings lookup during training stage.

After that, each sentence is then tokenized and transformed into a list of numbers based on the vocabulary built earlier. Sentences with too few or too much tokens are removed from the training set.

The last step happens during training where `NULL`s are padded at the begining of the sentence based on the configurable window size. Each batch data generated contains paragraph ids for each sentence as well as the encoded sentence itself. Batch labels returned are the ids of the next words the model tries to predict.

2.3 Feature Engineering

Let's discuss more about how to extract useful features from the corpus that could serve the purpose of classfying questions into corresponding topics.

2.3.1 TF-IDF

One common way to encode documents as numeric vectors is the vocabulary represnetation. Assume that the dataset has a vocabulary of size V in terms of unique words/tokens. Then for each document, it numeric representation is a vector of length V, which is usually a large number. For each word appearing in the document, the corresponding position in the vector is marked as the counts of that word, otherwise 0. Such representation clearly captures some of the characteristics of the document. However, it also tends to be noisy as many of the words appearing in a sentence are common words that can be easily seen from other sentences.

In order to reduce the noise level, tf-idf is preferred over simple term frequency. Instead, for each cell in the feature vector, term frequency or word counts is normalized by inverse document frequency (idf), hence the name. idf is calculated as follows:

$$idf(w, C) = \log \frac{N}{|\{d \in C; t \in d\}|} \tag{2.1}$$

where w is the word being calculated; C is the corpus data; N is the term frequency and d is each document in the corpus. Intuitively, the more the word appearing across documents, the less important it is in the final vector representation.

2.3.2 PV-DM

Also known as "doc2vec", the model is originally inspired by yet another famous embedding model called "word2vec" which translates words in a corpus into vector representations through neural netword based unsupervised training. The training mechanism is summarized in figure 2.3. Given a window consisting of consecutive words in a document, the input of the model is the concatenation/average of the document embedding and all the word embeddings within that window. The output of the model tries to predict the the next word outside the training window. Gradient descent method is used to optimized the loss function - usually calculated by negative sampling and cross entrophy.

In order to efficiently train the doc2vec model, a GPU implementation is conducted using TensorFlow the training is carried out using a GTX 1080 with 8GB memory. In this implementation, embeddings for paragraph/question is set to be 1000 while word embeddings is set to be 200. A sliding window of size 5 is adopted and both paragraph embedding and word embeddings are concatenated to form the final input vector. The major concern here is that I wanted to preserve the sequential nature of the inputs. To do so and to avoid overconsumption of memory, I chose a smaller embedding size and also a smaller window size. The final model is trained using gradient based optimization with a batch size of 64.

Given the output logits x and the target t, the cross entropy for this single input is:

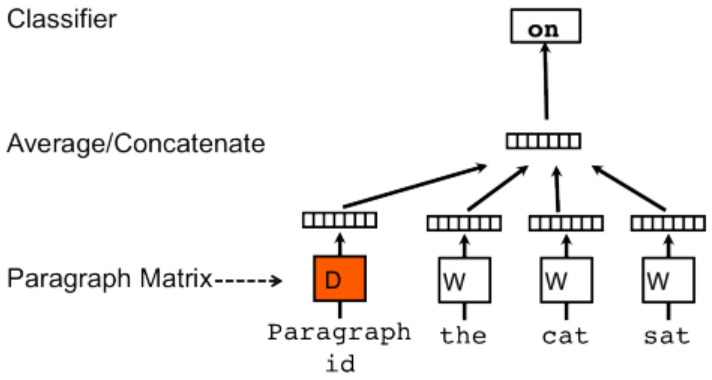
$$E(x,t) = -t \cdot \log\sigma(x) + (1-t)\log(1-\sigma(x)) \tag{2.2}$$

where $\sigma(\cdot)$ is the sigmoid function.

Since the size of the vocabulary is always a huge number, it is computationally expensive to calculate the cross entrophy against the whole vocabulary each time. Also, such computation will certainly obscure the importance of true labels, namely the right next word outside the window the model trying to predict. Instead, a negative sampling is performed to draw the true label with a bunch of other false labels and the cross entrophy is computed based on these samples. In the current implementation, a total of 100 samples are drawn for each input window including the true label.

In the training stage, it turned out that adaptive gradient ("AdaGrad") was able to converge to a smaller loss. AdaGrad is a gradient-based optimization method that takes advantage of historical gradient information and use it as way to adaptively adjust learning rate at each step.

Figure 2.3:



After training the model with 5 epochs, the final 1000-length paragraph embeddings are used as representative features. To understand the features a bit more, t-distributed stochastic neighbor embedding (t-SNE) is used to map the high-dimension data on a 2-D plane. t-SNE achieves the mapping by minimizing the Kullback-Leibler divergence of low-dimention distribution Q from the high-dimention P. Given N high-dimension points x_1 through x_N, t-SNE first calculates their pair-wise simularities through:

$$p_{ij} = \frac{p_{i|j} + p_{j|i}}{2N} \tag{2.3}$$

where $p_{j|i}$ is computed as:

$$p_{j|i} = \frac{\exp(-||x_i - x_j||^2/2\sigma_i^2)}{\sum_{k \neq i} \exp(-||x_i - x_k||^2/2\sigma_i^2)} \tag{2.4}$$

The low-dimension similarities, on the other hand, is calculated using a Student-t distribution with 1 degree of freedom:

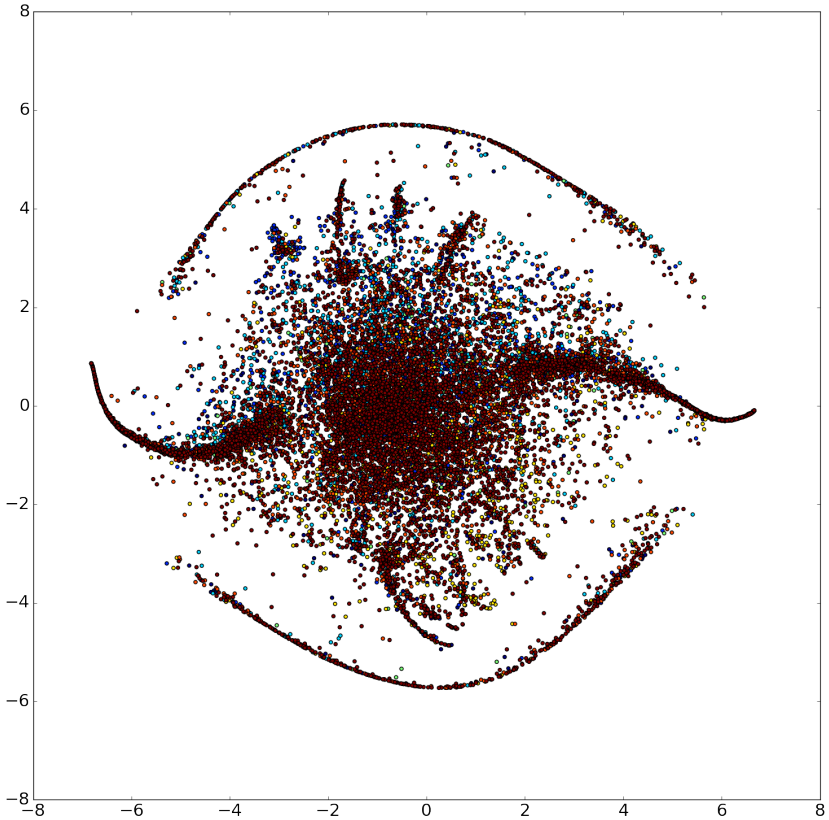
$$q_{ij} = \frac{(1 + ||y_i - y_j||^2)^{-1}}{\sum_{k \neq m}(1 + ||y_k - y_m||^2)^{-1}} \tag{2.5}$$

The KL divergence is hence:

$$KL(P||Q) = \sum_{i \neq j} p_{ij} \log \frac{p_{ij}}{q_{ij}} \tag{2.6}$$

The final mapping is visualized in a 2-D plane (see Figure 2.4).

Figure 2.4: t-SNE mapping of paragraph embeddings



The mapping indicates that there are underlying structures of the embedding features. In subsequent sections, I will resort to finding classification models that can make use of the

embeddings. Next sub-section will focus on setting up the baseline model for topic classification.

2.4 Baseline Model

The baseline model used tf-idf vector as input to a Gaussian Naive Bayes classifier. Naive Bayes models assume the independence between features and use Bayes' theorem to infer the class of the each data point:

$$\tilde{y} = arg\max_{y} P(y) \prod_{i=1}^{n} P(x_i|y) \tag{2.7}$$

where $\tilde{y}$ is the estimated class given a data point $x = x_1, x_2, \cdots, x_n$ and y is the true class. In practice, Maximum A Posteriori (MAP) is used to estimate $P(y)$ and $P(x_i|y)$. In the Gaussian setting, the likelihood of a feature given class y is assumed to be Gaussian, namely:

$$P(x_i|y) = \frac{1}{\sqrt{2\pi\sigma_y^2}} \exp\left(-\frac{(x_i - \mu_y)^2}{2\sigma_y^2}\right) \tag{2.8}$$

where σ_y and μ_y is the standard deviation and mean of the underlying Gaussian distribution.

2.4.1 Experiment

The experiment is carried out by splitting the dataset into training and testing subsets. In this case, the testing size is 0.75. To ensure the same train-test split can be reproduced, a fixed random state is set in numpy. In the baseline model, only the 1,000 most frequent words are chosen in the vector representation of each question in order to fit into the memory.

2.4.2 Baseline Results

As mentioned in previous section, the performance of the classification model is evaluated from three metrics: precision, recall and F1-score. Since there are in total seven classes, an average of each metric is calculated across each class. In the unweighted case, data imbalance is not considered while in the weighted case metrics are averaged based on the number of instances in each class. The final results are shown in table.

Table 2.1: Baseline model

	Precision	Recall	F1-Score
unweighted	0.62	0.74	0.64
weighted	0.82	0.73	0.75

To obtain more granular details of the results, a confusion matrix is calculated and visualized (see Figure 2.4). As can be seen from the map, topics like cooking, crypography, robotics and

travel are easily distinguished by the baseline classification model while other topics such as biology, diy and physics are not well classified. Biology is sometimes confused with cooking (0.09) or physics (0.06) and diy is often confused with physics (0.22). And vise versa physics questions are often mistaken as biology (0.09) or diy (0.08).

2.5 Solution Model

Previous section has decribed how the baseline model is developed. In this section, different variants of the classification models are experimented and compared.

2.5.1 Normalization

The preliminary results show that using raw paragraph embeddings only does not lead to good classification results (see Table 2.2).

Table 2.2: GNB with doc2vec

	Precision	Recall	F1-Score
unweighted	0.24	0.21	0.13
weighted	0.42	0.17	0.14

This is probably because in the training stage weights in doc2vec model are updated independently. This leads to the fact that the weights within and across samples are of different scales. Hence I decided to perform normalization to see if better results can be achieved. Normalization is implemented within each sample across all features, namely all the values in the document's embedding. *L2* norm is used in this case. After normalization, it is shown that the classification results are improved significantly (see Table 2.3).

Table 2.3: GNB with normalized doc2vec

	Precision	Recall	F1-Score
unweighted	0.50	0.57	0.52
weighted	0.65	0.60	0.61

2.5.2 Feature concatenation

Although normalization has improved classification model's performance using merely doc2vec features, the results are still underperformed compared to baseline model. The next fine-grained method I came up with is to concatenate doc2vec features with existed tf-idf features. It turns out that it did improve the prediction results on testing set, but only by a small scale (see Table 2.4).

Table 2.4: GNB with normalized doc2vec + tf-idf

	Precision	Recall	F1-Score
unweighted	0.62	0.75	0.65
weighted	0.82	0.74	0.76

2.5.3 Random Forest

The results indicates that TF-IDF provides better features in the task of predicting questions topics of unseen dataset. To further refine classification model's performance, one of the ensemble methods, Random Forest, is chosen in the experiment. Using 20 estimators and default settings in sklearn, the model shows great improvement (see Table 2.5).

Table 2.5: Random forest with tf-idf

	Precision	Recall	F1-Score
unweighted	0.84	0.74	0.77
weighted	0.86	0.86	0.86

3 Conclusion

To sum up, all the results so far are presented in Figure 3.1 and 3.2. Considering the nature of the problem itself, it tries to predict what topic a particular question belongs to. One of the possible explanation of why TF-IDF out-performs more contrived features (PV-DM, or doc2vec) is that the former one, as a bag-of-words method, encodes keywords more explicitly while the latter one puts more efforts in capturing syntatic and semantic details. Intuitively, if asked to manually label a question with certain topic, one can get a pretty good guess by by merely scanning through the keywords/terminologies being used. Therefore, although doc2vec seems to have better results in tasks like sentiment analysis, as mentioned in the original paper, there is no guarantee that it would do better in general.

Figure 3.1: Model results (weighted)

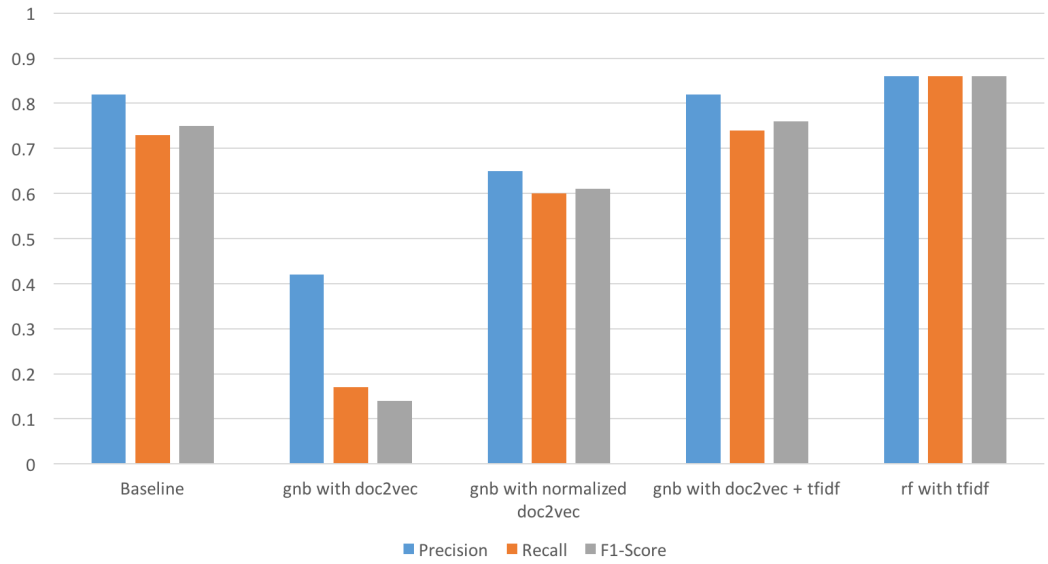
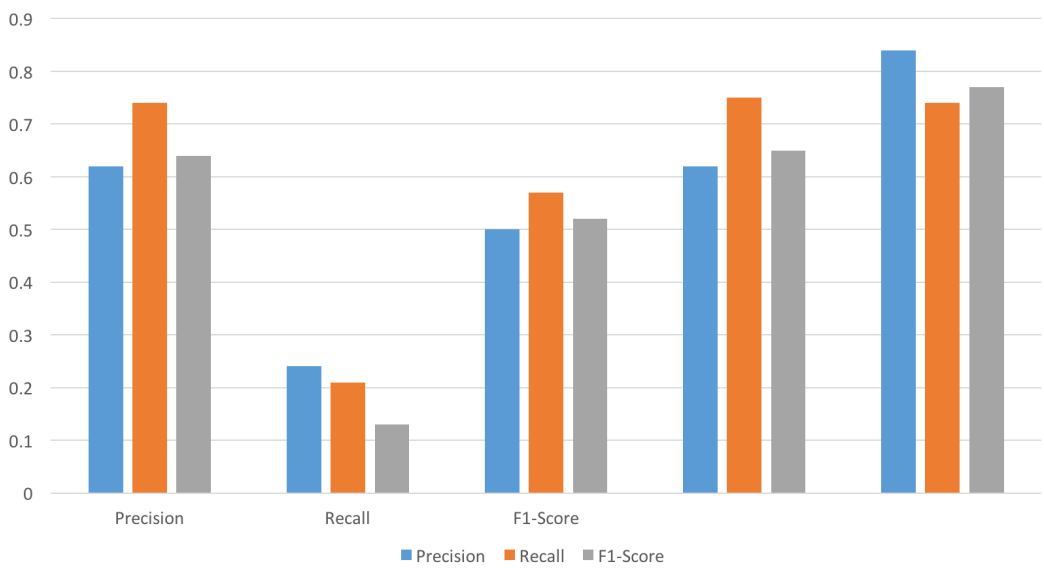


Figure 3.2: Model results (unweighted)



4 Reference

[1] Le, Quoc V., and Tomas Mikolov. "Distributed Representations of Sentences and Documents." ICML. Vol. 14. 2014.
[2] Kaggle competition: "Transfer Learning on Stack Exchange Tags", 2014.
[3] StÃl'fan van der Walt, S. Chris Colbert and GaÃńl Varoquaux. The NumPy Array: A Structure for Efficient Numerical Computation, Computing in Science & Engineering, 13, 22-30 (2011), DOI:10.1109/MCSE.2011.37
[4] John D. Hunter. Matplotlib: A 2D Graphics Environment, Computing in Science & Engineering, 9, 90-95 (2007), DOI:10.1109/MCSE.2007.55
[5] Fabian Pedregosa, GaÃńl Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake Vanderplas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, ÃL'douard Duchesnay. Scikit-learn: Machine Learning in Python, Journal of Machine Learning Research, 12, 2825-2830 (2011)
[6] Wes McKinney. Data Structures for Statistical Computing in Python, Proceedings of the 9th Python in Science Conference, 51-56 (2010)
[7] "Precision and recall" Wikipedia: The Free Encyclopedia. Wikimedia Foundation, Inc. 12 Feb 2017. Web. 10 Aug. 2004.
[8] "t-distributed stochastic neighbor embedding" Wikipedia: The Free Encyclopedia. Wikimedia Foundation, Inc. 12 Feb 2017. Web. 10 Aug. 2004.
[9] MartÃŋn Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Rafal Jozefowicz, Yangqing Jia, Lukasz

Kaiser, Manjunath Kudlur, Josh Levenberg, Dan ManÃľ, Mike Schuster, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda ViÃľgas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.