Text Based Information Retrieval PART I. - PART II Retrieval Task: From Questions to Answers

Orsolya Lukacs-Kisbandi

May 23, 2017

1 Introduction

The aim of this project was to create both an unsupervised and supervised model to solve a real life problem as well as to gain a better understanding of the ideas and concepts presented during lectures. The task was to implement models which could help in finding relevant answers to questions on CQA (Community Question Answering) forums.

2 Part A - Unsupervised retrieval model

For the first part of the assignment we were asked to build an unsupervised retrieval model which could be used to rank the answers to a given question by relevancy.

To achieve this, first the relevant parts of the input file had to be extracted (Questions and Comments) than the input data had to be pre-processed. For obtaining the expected text format the function *cleantext* was used.

```
def clean_text(a):
    nr = len(a)
    text = []
    for i in xrange(0, nr ):
        letters = re.sub("[^a-zA-Z]", " ", a[i].get_text())
        words = letters.lower().split()
        stops = set(stopwords.words("english"))
        useful_w = [w for w in words if not w in stops]
        text.append(" ".join(map(lambda x: lemmatizer.lemmatize(
            x), useful_w)))
    return(text)
```

Listing 1: Function to pre-process text

This function first removes every non-alphabetical letter from the input, than converts the text into lower case letters. In some cases this might cause worse results, since sometimes upper case letters represent names or places which would provide additional information. After importing a set of stop words, such as *the*, *or*, *and*, they are all removed from the text, because they don't contain any useful information. The set of words is joined together after lemmatization, every word is normalized into their base form.

The list returned by the above function is turned into a Bag of Words representation. The Bag of Words model learns a vocabulary from all the data, then models each comment/question by counting the number of times each word appears.

TF-IDF(Term-Frequency-Inverse Document-Frequency) is a term weighting method, which was used to determine which words are most relevant to a document. To calculate TF-IDF we need to know how often does a word occur in the document (term frequency = tf), and the inverse of the number of documents from the collection in which it appears in (document frequency = df). After understanding how TF-IDF works, the skit learn built in library function was used. The reason for this, besides the fact that the built in function is very efficient, is that there was a misunderstanding about the task and I spent a lot of time trying to solve another problem. As I understood, we had to build an unsupervised model to find the ten most relevant answers to new questions in the answer set from the training data.

Using the results from TF-IDF, the cosine similarity between every question and comments could be computed. The cosine similarity between two vectors is a measure that determines the cosine of the angle between them. This is a good measure of similarity in Text based information retrieval because it is not only taking into consideration the magnitude of each word-count obtained by TF-IDF, but also the angle between the documents. If only the magnitude would be taken into consideration it would yield in inaccurate results, since two text vectors might have a small difference in magnitude, but if the angle between them is large, that means they are not similar. The cosine similarity can be obtained by the following function:

```
def cosine_sim(text1, text2):
     intersection = set(text1.keys()) & set(text2.keys())
     num = sum([text1[x] * text2[x] for x in intersection])
     sum1 = sum([text1[x]**2 for x in text1.keys()])
     sum2 = sum([text2[x]**2 for x in text2.keys()])
     den = math.sqrt(sum1) * math.sqrt(sum2)
return float(num)/den
```

Listing 2: Cosine similarity

This similarity measure was used used to determine a ranking between answers.

2.1 Other methods

My first approach to solving the task was to create Bag of Words representation of the comments and than cluster them around the questions they were answering. For the distance measure I was using Euclidean distance, than after realizing it was not a good measure for relevancy also tried cosine similarity. This turned out to be an inefficient approach since, the bag of words representation without any further processing when clustered, resulted in bad results. This is because due to this type of representation, when similarity was calculated, it preferred documents which repeated the same words as the question.

3 Part B - Supervised retrieval model

For the second part of the assignment we were asked to build a supervised neural-network based retrieval model which could be used to rank the answers to a given question by relevancy as well as to predict labels for the answers.

The same way as in the first part of the assignment, the first step was to pre-process the input data. After obtaining the text the skit.learn function, *TfidfVectorizer* was used to obtain the TF-IDF vectors with the maximum number of features set to 20.000, which returns a mapping of terms to feature indices and the learned idf vector. By applying the built model to the data we get the features which we use to calculate the similarity measures between questions and comments.

The selected type of neural-network was Long Short Term Memory networks (LSTM) because of it's scalability and good performance. LSTMs are a version of RNNs (Recurrent Neural Networks) but they can learn long term dependencies, they are trained using Backpropagation through time and overcome the vanishing gradient problem. Instead of neurons, LSTM networks have memory blocks that are connected through layers, which have components that provide more functionality than a classical neuron and a memory for recent sequences. A block operates on an input sequence and uses sigmoid activation units to control its' behaviour.

The library used to build the classification model was Keras, a Pyhton library which has many built in functions for information retrieval and data mining tasks. The number of neurons used to build the model is 128, because we are solving a classification problem we use a Dense output layer with a single neuron and a sigmoid activation function to make 0 or 1 predictions for the two classes. When compiling the model, we use accuracy as metrics, binary cross-entropy as loss and 'adam' optimizer, which is a first-order gradient-based optimization of stochastic objective functions.

4 PART II

The third part of the assignment was to implement a *Learning to rank* model which would make use of pair-wise ranking and loss functions. This model is built on an algorithm that learns to rank from a ground-truth ranking. For the ground truth the three labels *Good, Potentially useful, Bad* ,(in the code having values 2, 1, 0) were used. For this task two approaches were explored, an LinearSVC based SVM model and a Feed Forward Neural Network model. Creating the features vectors works the same way as in the previous parts, the text is extracted, cleaned, and applying TF-IDF the feature vectors are created. The output of TF-IDF was sued to clacluate the cosine similarity between answers and questions as well as between two asnwers.

4.1 SVM

The SVM approach makes use of the Python built in function *LinearSVC* which works on the principles of linear support vector classification. Some of the functions of LinearSVC were overridden to apply the new functions to our problem allowing to use the pair-wise approach.

```
def toPairs(X,simans, y):
    X_trans = []
    y_trans = []
    y = np.asarray(y) # Turn input labels into array
    if y.ndim == 1:
        y = np.c_[y, np.ones(y.shape[0])]
    #concatonate the two arrays y and an array of ones
    comb = itertools.combinations(range(X.shape[0]), 2)

    for k, (i, j) in enumerate(comb):
    # For each element in the combination
    # k is the number of combinations
        val = X[i,0] - X[j,0] + cos_sim(simans[i].reshape(1, -1)
            ,simans[j].reshape(1,-1))
        X_trans.append(val[0])
        if y[i, 0] == y[j, 0]:
            if cos_sim(simans[i].reshape(1, -1),simans[j].
                reshape(1,-1))>0: y_trans.append(y[i, 0])
            # if the two answers are similar and have the same
                label use label
            elif X[i,0] - X[j,0] > 0 : y_trans.append(1)
            # if the answers are not similar and the first is
                better
            else: y_trans.append(-1)
            # if the answers are not similar and the second is
                better
        else: y_trans.append(np.sign(y[i, 0] - y[j, 0]))
        # if they are not similar, use the sign of the
            difference of cosine similarity
    return np.asarray(X_trans), np.asarray(y_trans).ravel()
```

Listing 3: Pairwise transform function

The first challenge of this part was to create the pair-wise representation of the feature vectors. For every answers the cosine similarity to it's question was calculated, as well as the similarity between every two answers, and used instead of the feature vectors. Creating the pairs was done with the function *toPairs*, which creates all possible combinations from the input data and assigns a label to them when training. The label was chosen carefully, since we want to predict which answer is better out of the two. In case *X[i]*'s label had higher value than *X[j]* (meaning it was more relevant to it's question) the label was +1 otherwise -1. If both had the same value, the label got assigned based on the cosine similarities, since the question with the larger similarity value was the more relevant one.
The class *RankSVM* is the classifier which is built on *svm.LinearSVC*. The function *fit* invokes the transforming function and than fits the model. The function *predict* is predicting the labels for the test data.
After predicting all the labels for the combinations, we decide on the ordering of the answers to a questions based on them. After reviewing the the method used to train and predict the rankings, a dimensionality reduction was applied, instead of combining every answer to every other answer, they were only combined within a question.

4.2 Feed Forward Neural Network

Feed forward neural networks are For this approach the same functions and methods were used to create the pairwise representations, but the model this time is a Feed Forward Neural Network. There are six layers, every layers' nodes are connected to every node in the next layer, building the network. The output layer has only one neuron, since it is a binary classification. Unfortunately this models' performance was not evaluated because it was overflowing the memory and by the time I found a way to reduce the dimension it had to be submited.

4.3 SVM VS. ANN

ANN (Artificial Neural Networks) and SVM (Support Vector Machines) are two popular strategies for supervised machine learning and classification. Both approaches have their advantages and disadvantages. Support Vector Machines are often better to ANNs because they avoid two major weaknesses of ANNs. The first one is that ANNs are often influenced by local minima, especially Feed Forward Neural Networks, since there is no way in this type ow network to step back, ot often converges to a local instead of a global minima. The other drawback is that ANNs tend to overfit if they have too many iterations. When using SVMs we can exploit useful features of the model such as, with the help of margin maximization, Ranking SVM can have a good generalization ability. In our case both approaches work well.

5 Results

For the first part of the assignment the achieved MAP results was (MAP for SYS): 0.5274. If we take into consideration that it is an unsupervised this score is realistic.
For the second part of the assignment, when predicting labels for the development set, the model resulted in F1 0.52 accuracy, the precision value is 0.40 and the recall value is 0.73, the score for the system is 0.79 and the mean reciprocal rank is 67. The MAP prediction is (MAP for SYS): 0.5934.
For the third part of the assignment, the MAP value was 0.5384, which is not as it was expected, since it performs only slightly better than the unsupervised model. If we look at the MRR (Mean reciprocal rank) we can see that it's value is 63.13, which is a fairly good result.

References

[1] Large Margin Rank Boundaries for Ordinal Regression *Advances in Large Margin Classifiers*. R. Herbrich, T. Graepel, and K. Obermayer. 115-132, Liu Press, 2000.

[2] Efficient algorithms for ranking with SVMs *Special Issue on Learning to Rank*. O. Chapelle and S. S. Keerthi, Information Retrieval Journal, , 2009

[3] Support Vector Machines for Text Categorization *IEEE Conference* A. Basu, C. Watters, and M. Shepherd

[4] http://fa.bianp.net/blog/2012/learning-to-rank-with-scikit-learn-the-pairwise-transform/