Using Multilingual Neural Re-ranking Models for Low Resource Target Languages in Cross-lingual Document Retrieval

Caitlin Westerfield

Senior Project
Bachelor of Science in Computer Science
Advisor: Dragomir Radev

Yale University
December 7th, 2018

1 Abstract

Low-resource target languages introduce many challenges for cross-lingual document retrieval and detection (CLDD), as well as re-ranking. First, while CLDD can be reduced to monolingual information retrieval by document translation using machine translation (MT) systems, such MT systems suffer from the lack of parallel data for low-resource target languages. Second, recent neural retrieval models that outperform traditional language modeling approaches suffer from the scarcity of relevance judgments in low-resource target languages. Due to these constraints, it is necessary to find ways to optimize the document retrieval process in ways that are not bound by the restraints on training data. This project focuses on exploring a variety of existing monolingual neural re-ranking models and applying them to the task of multilingual document retrieval.

2 Introduction

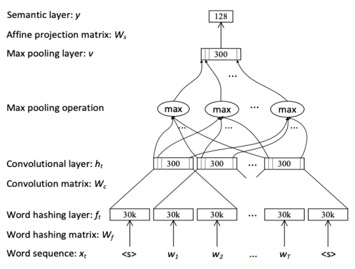
Cross-Lingual Information Retrieval (CLIR) is the task of retrieving and ranking relevant documents in one language based on queries in a different language. Currently, if a user searches for a specific query in English in a search bar, the returned documents will be almost entirely in English as well. While documents of the same language are often enough to satisfy the user, sometimes there may be a document in another language that better fits the user's needs. For example, a native Swahili speaker is travelling to the Philippines and wants to learn about the most beloved local restaurants. To search through only Swahili documents would signficantly decrease the user's chance of finding relevant information. Thus, CLIR aims to merge existing machine translation methods with information retrieval to return the most relevant documents for a given query regardless of language.

There are two major methods for development of CLIR systems. The first, and more traditional, approach is the modular approach. In these systems, the pipeline consists of two components: machine translation and monolingual information retrieval. First, an existing machine translation system translates all the documents into the query's language prior to indexing, thus converting CLIR into a monolingual problem (Nie, 2010). However, this method risks losing important information during the translation stage, especially regarding phrasing, locality, and syntax – all of which are useful for determining relevance in the retrieval process.

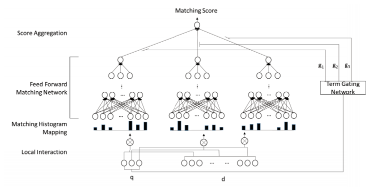
Another technique for CLIR is *learning to rank* directly from the relevance judgment of query-document pairs. In these models, relevance judgements are produced for each query and document pair that are then used to train the model to directly calculate relevance scores for other queries and documents. These types of models transcend language because they learn information retrieval and translation judgements at the same time. They also preserve semantic and locality information.

3 Problem Description

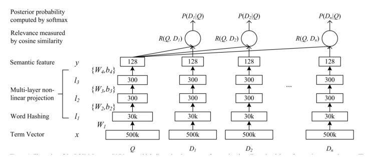
Although extensive work has been done in both the field of machine translation and information retrieval on these two strategies, the process of developing ways to optimize multilingual


Semantic layer: y
Affine projection matrix: W_s
Max pooling layer: v
Max pooling operation
Convolutional layer: h_t
Convolution matrix: W_c
Word hashing layer: f_t
Word hashing matrix: W_f
Word sequence: x_t
128
300
max
30k
<s>
w_1
w_2
w_T

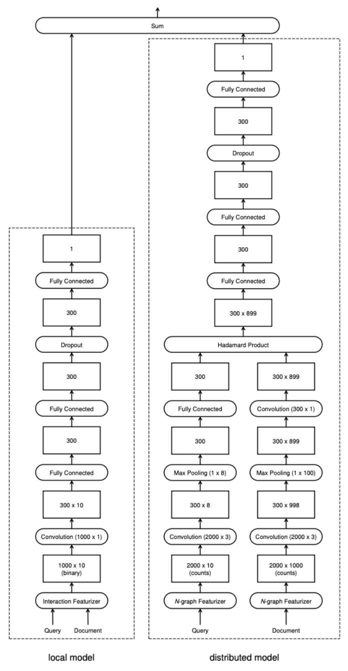
(a) Model of the CDSSM architecture (Shen et al., 2014)



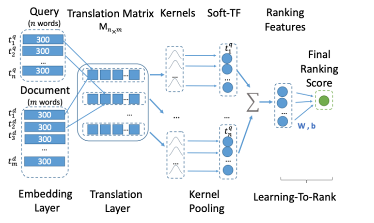
(b) Model of the DRMM architecture (Guo et al., 2016)


Posterior probability computed by softmax
Relevance measured by cosine similarity
Semantic feature
Multi-layer non-linear projection
Word Hashing
Term Vector
128
300
30k
500k
Q
D_1
D_2
D_n

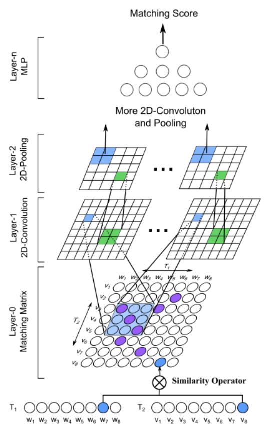
(c) Model of the DSSM architecture (Huang et al., 2013)



(d) Model of the DUET architecture (Mitra et al., 2017)



(e) Model of the K-NRM architecture (Xiong et al., 2017)



(f) Model of the MatchPyramid architecture (Pang et al., 2016)

Figure 1: Pre-Existing Monolingual Neural Re-ranking Models

CLDD systems by combining elements of both is still in its infancy. This project aims to dive deeply into the field of information retrieval focused on Neural Re-Ranking in order to determine which re-ranking methods work best in a multilingual setting. All six re-ranking systems are implemented using training data made up of the output of 11 distinct machine translation systems in order to optimize the training of the CLDD system.

4 Related Work on Neural Learning to Rank

Many neural learning to rank models have shown promising results on monolingual IR datasets (Shen et al., 2014; Guo et al., 2016; Huang et al., 2013; Mitra et al., 2017; Xiong et al., 2017; Pang et al., 2016).

4.1 MatchPyramid

The most basic method is MatchPyramid which was introduced in Pang et al. (2016). This method constructs a word level similarity matrix and uses meaningful matching patterns to determine word level matching signals.

4.2 K-NRM

K-NRM, or the kernal based neural ranking model, is another method that uses a translation matrix to compute word-level similarities (Xiong et al., 2017). In the original implementation, the translation matrix indicates the "translation" between words in the query and words in the document, however this can also be applied to a multilingual system. This model uses kernals to extract multi-level soft match features which then use *learning to rank* to determine a final ranking score.

4.3 DUET

Another model is the DUET model which was introduced in Mitra et al. (2017). This re-ranking system uses a dual approach to match the query to the document based on both a local and distributed representation of the text. The purpose of this "duet" is for the local model to capture properties such as proximity and position of relevant terms within the document while the distributed model detects related terms, synonyms, and other term-specific properties. This model takes into consideration location within the document in order to appropriately manage longer documents that may contain a mixture of many different sub-documents of varying topics.

4.4 DSSM & CDSSM

The Deep Structured Semantic Model (DSSM) introduced in Huang et al. (2013) is another neural re-ranking model that has been widely regarded favorably due to its use of a *word hashing* method. DSSM takes high-dimensional term vectors of queries and documents and projects them into low-dimensional letter-based n-gram concept vectors in a semantic feature space. Word hashing uses less space since it only has to store n-grams, thus allowing for large vocabularies – a trait that is useful for tasks such as document detection within large corpora. However, the original implementation of the DSSM model in Huang et al. (2013) only based re-ranking decisions on document titles rather than document text. Similarly, CDSSM is an implementation of the DSSM model that includes a convolution layer (Shen et al., 2014).

4.5 DRMM

In contrast to DUET and DSSM, the Deep Relevance Matching Model (DRMM) Guo et al. (2016) disregards the location of terms within the document and trains on the text of the document rather than the title. DRMM relies on relevance matching by comparing the term embeddings of each pair of terms within the query and the document in order to build local interactions. These are then used to generate fixed-length matching histograms that are employed in a feed forward matching network to learn the hierarchical matching patterns and return a score for the current query and document. The DRMM model is similar to other interaction-focused models such as MatchPyramid since it is based on matching signals. However, the histograms represent different levels of signal strengths rather than positions within the document.

5 Approach

This project was part of the MATERIAL competition that focuses on developing CLIR and machine translation systems for low-resource languages - or languages with minimal training data. For the competition, every six months a new language is released for competing universities to train their systems with which gradually become less and less well used (thus decreasing in the amount of training data available). Since this project was aimed at improving the Yale MATERIAL system, the two languages used were the languages prechosen by MATERIAL: Swahili and Tagalog.

To go about determining the best neural re-ranking system to use on multilingual systems, I relied heavily on the MatchZoo pre-existing implementations of the CDSSM, DRMM, DSSM, DUET, K-NRM, and MatchPyramid models (Fan et al., 2017). While these models were already structured for monolingual systems, I altered the implementation to use English queries to train on Swahili documents and test on Tagalog documents, as well as train on Tagalog documents and test on Swahili documents. I then calculated necessary evaluation metrics using scripts found in MatchZoo as well as in the TREC dataset that is associated with MATERIAL[1]. The output of the evaluation provided the necessary information to compare the re-ranking system and MT system pairs.

The re-ranking systems were trained on the filtered outputs of the machine translation systems. This was beneficial because the systems such as DBQT, PSQ, SMT, and NMT filter out any documents that they deem extraordinarily irrelevant which means that the training data was not primarily negatively labeled documents.

6 Datasets

The main bulk of this project used the MATERIAL dataset with specific focus on the English QUERY1 queries and the DEV1 Swahili and Tagalog text and audio documents. This dataset includes relevance pairs between the 300 English Queries and 844 Tagalog Documents, as well as these same English Queries and 813 Swahili Documents.

This project also relied on the monolingual GloVe embeddings (Pennington et al., 2014). Additional work on the MATERIAL team was done on implementing multilingual embeddings such as MUSE (Lample et al., 2017) which further improved the results of the multilingual re-ranking, however this work was the focus of another contributor to the MATERIAL project.

[1]https://trec.nist.gov/trec_eval/

	SW->TL			
	MAP	P@20	NDCG@20	AQWV
Deep Relevance Ranking				
MatchPyramid	20.85	4.40	28.86	23.89
DUET	21.70	4.78	31.44	30.65
K-NRM	25.44	5.00	34.71	30.07
CDSSM	22.03	4.57	31.05	29.74
DSSM	22.78	4.89	32.09	28.76
DRMM	**33.41**	**5.10**	**42.78**	**36.97**

Table 1: Evaluation results on the MATERIAL Dataset. The Deep Relevance Ranking is trained on English queries and Swahili documents, and tested on English queries and Tagalog documents. The results are based on the optimal ML system for each re-ranking system. See Appendix for breakdown by ML system.

7 Evaluation Method

For evaluation, I used the MatchZoo ad-hoc retrieval evaluation script to compute Precision, Mean Average Precision (MAP), and Normalized Discounted Cumulated Gain (NDCG) (Fan et al., 2017). As another detection-based metric, I also calculated the Actual Query Weighted Value (AQWV) introduced by NIST which is represented by this function:

$$\begin{gathered}
AQWV = 1 - (avg_{rel-q} P_m + \beta \cdot avg_{all-q} P_{fa}) \\
P_m = \frac{N_{miss}}{N_{relevant}} \\
P_{fa} = \frac{N_{fa}}{N_{total} - N_{relevant}} \\
\beta = \frac{C}{V}\left(\frac{1}{P_{relevant}} - 1\right)
\end{gathered} \tag{1}$$

P_m and P_{fa} refer to the probability of missing a relevant document and obtaining a false alarm (i.e. a document that's not relevant) respectively, C is the cost of an incorrect detection, V is the value of a correct detection, and P_{rel} is the probability of a document being relevant. The average P_m only considers queries with relevant documents since queries without relevant documents have an undefined P_m (observe that $N_{relevant} = 0$).

8 Results

The results displayed in Table 1 show that the implementation of DRMM drastically outperforms other methods in the MAP, NDCG@20, and AQWV metrics for training on Swahili and testing on Tagalog. These results indicate that training on the original pre-translated document text for information retrieval tasks is highly beneficial because it assigns relevance scores that more closely mirror the correct ranking of documents by relevance and thus makes cutoff learning more beneficial.

It is clear that overall DRMM produces the net best outcome for multilingual systems, which indicates that the features that are critical to DRMM also are most important for multilingual tasks. Since DRMM does not rely on location of terms within the document, this indicates that for multilingual tasks term location is not critical for predicting relevance to a query. Additionally, DRMM's signal matching approach using histograms is something to explore further as a potentially good structure for multilingual CLIR tasks.

9 Conclusion

This work proved that using neural re-ranking models significantly alters performance of CLIR systems for English Queries on Swahili Documents. Additionally, DRMM is the pre-existing monolingual re-ranking system that performs the best in a multilingual setting. The systems that work for re-ranking are also representative of what representation techniques could work on other CLIR tasks. Thus, due to the success of DRMM, it is likely that other tasks could benefit from focusing on signal-based matching using histograms and the discarding of term location information.

10 Future Work

While preliminary results on these models are promising in showing that neural re-ranking models can improve multilingual CLIR tasks, further work needs to be done to continue optimization of the system. The systems need to be evaluated on a wider variety of datasets and languages for both the documents and the queries. Currently all systems are being tested on English queries and documents in Swahili (while being trained on documents in Tagalog) so expanding to a wider variety will allow us to determine a more accurate picture of where re-ranking is helping as well as more areas for potential improvement.

Similarly, while this system is currently trained and tested on different languages, another step would be to create a pipeline such that systems can be trained and tested on the same language (for example train and test on Swahili documents but with English queries). This would be a way to keep all work in-domain and thus allow for easier evaluation in comparison to existing baselines.

Finally, while currently the re-ranking systems are each implemented in isolation, re-ranking system combination could be an excellent next step. By merging the most relevant aspects of each model together it would be possible to create a revolutionary re-ranking model specifcally for multilingual tasks.

11 Acknowledgements

I would like to thank Dragomir Radev and Rui Zhang for their guidance and support on this project, along with Trevor Shim, Garrett Bingham, and others who have contributed to the overarching MATERIAL project.

References

Yixing Fan, Liang Pang, JianPeng Hou, Jiafeng Guo, Yanyan Lan, and Xueqi Cheng. 2017. Matchzoo: A toolkit for deep text matching. *arXiv preprint arXiv:1707.07270*.

Jiafeng Guo, Yixing Fan, Qingyao Ai, and W Bruce Croft. 2016. A deep relevance matching model for ad-hoc retrieval. In *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management*, pages 55–64. ACM.

Po-Sen Huang, Xiaodong He, Jianfeng Gao, Li Deng, Alex Acero, and Larry Heck. 2013. Learning deep structured semantic models for web search using clickthrough data. In *Proceedings of the 22nd ACM international conference on Conference on information & knowledge management*, pages 2333–2338. ACM.

Guillaume Lample, Alexis Conneau, Ludovic Denoyer, and Marc'Aurelio Ranzato. 2017. Unsupervised machine translation using monolingual corpora only. *arXiv preprint arXiv:1711.00043*.

Bhaskar Mitra, Fernando Diaz, and Nick Craswell. 2017. Learning to match using local and distributed representations of text for web search. In *Proceedings of the 26th International Conference on World Wide Web*, pages 1291–1299. International World Wide Web Conferences Steering Committee.

Jian-Yun Nie. 2010. Cross-language information retrieval. *Synthesis Lectures on Human Language Technologies*, 3(1):1–125.

Liang Pang, Yanyan Lan, Jiafeng Guo, Jun Xu, Shengxian Wan, and Xueqi Cheng. 2016. Text matching as image recognition. *CoRR*, abs/1602.06359.

Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014. Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543.

Yelong Shen, Xiaodong He, Jianfeng Gao, Li Deng, and Gregoire Mesnil. 2014. Learning semantic representations using convolutional neural networks for web search. WWW 2014.

Chenyan Xiong, Zhuyun Dai, Jamie Callan, Zhiyuan Liu, and Russell Power. 2017. End-to-end neural ad-hoc ranking with kernel pooling. In *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 55–64. ACM.

12 Appendix

12.1 Relevant Code

The sections of code included below (except for the config files) are from the DRMM system implementation. However, very similar files were also used in all other system implementations. Also included in the implementation of these systems were programs taken directly from the MatchZoo GitHub account and thus should be referenced there.

12.1.1 run_drmm_system.sh

```bash
#!/bin/bash
# download the glove vectors

# define files that are used throughout including the corpus and
# word dictionary
python create_query_list.py
python generate_corpus_and_dict.py

# generate word embedding
python gen_w2v.py ../word_embedding/glove/glove.840B.300d.txt\
 word_dict.txt embed_glove_d300
python norm_embed.py embed_glove_d300 embed_glove_d300_norm
python gen_w2v.py ../word_embedding/glove/glove.6B.50d.txt\
 word_dict.txt embed_glove_d50
python norm_embed.py embed_glove_d50 embed_glove_d50_norm

echo "System    PosTrain    NegTrain    PosValid    NegValid    \
PosTest NegTest ndcg@3  map ndcg@20 precision@20    ndcg@5"\
 > system_results.txt
echo "System    Cutoff  Queries Pmiss   Pfa QWV" > \
aqwv_outputs.txt

mkdir predictions
mkdir cutoffs

for SYSTEM in DBQTWiktionaryMergedStemCheck umdNMT_unstemmed\
 EdiNMT_stemmed UMDPSQPhraseBasedCutoff097 \
 EdiNMT_TFIDF_Expanded umdSMT_stemmed EdiNMT_unstemmed\
 umdSMT_TFIDF_Expanded umdNMT_stemmed umdSMT_unstemmed\
 umdNMT_TFIDF_Expanded
do


#create the trainfile, devfile, and testfile both preprocessed
#and processed
python create_files.py $SYSTEM
```

```

# transfer the dataset into matchzoo dataset format
python transfer_to_mz_format.py
python prepare_mz_data.py

cat word_stats.txt | cut -d ' ' -f 1,4 > embed.idf
python gen_hist4drmm.py 60
python gen_binsum4anmm.py 20 # the default number of bin is 20

echo "Done with run_data..."

./run_drmm.sh

python generate_q-querys.py $SYSTEM

PARAM1=/data/projects/material/system_combination/data/\
relevance_data/1B-QUERY1-DEV/
PREDICTIONS=predictions/${SYSTEM}_predictions/
PARAM2=cutoffs/
PARAM3=/data/projects/material/system_combination/data/1B_dev.lst

for CUTOFF in 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 \
18 19 20 21 22 23 24 25
do
rm cutoffs/*

#generate special aqwv length predict file
python create_aqwv_folder.py $PREDICTIONS $CUTOFF
python aqwv.py $PARAM1 $PARAM2 $PARAM3 $SYSTEM $CUTOFF

done

done

python calculate_best_cutoffs.py
```

12.1.2 create_query_list.py

```
# coding: utf-8

import os
import sys
from argparse import ArgumentParser
import math

train_query_list =\
 "/data/corpora/azure2/storage2/data/NIST-data/\
 1A/IARPA_MATERIAL_BASE-1A/QUERY1/query_list.tsv"
```

```python
test_query_list =\
 "/data/corpora/azure2/storage2/data/NIST-data/\
1B/IARPA_MATERIAL_BASE-1B/QUERY1/query_list.tsv"

def main():
    querydict = {}
    with open(train_query_list, 'r') as f:
        content = f.read().splitlines()
        content.pop(0)
        for line in content:
            lineSplit = line.split("\t")
            querydict[lineSplit[0]] = lineSplit[1]
    with open(test_query_list, 'r') as f:
        content = f.read().splitlines()
        content.pop(0)
        testf = open("test_query_list.txt", "w+")
        for line in content:
            lineSplit = line.split("\t")
            querydict[lineSplit[0]] = lineSplit[1]
            testf.write(lineSplit[0] + "\t" +\
             lineSplit[1]+"\n")
        testf.close()

    newf = open("query_list.txt", "w+")
    for query in querydict:
        newf.write(query + "\t" + querydict[query] + "\n")
    newf.close()



main()
```

12.1.3 generate_corpus_and_dict.py

```python
import os
import nltk
from nltk.tokenize import TreebankWordTokenizer
import re

docpathtrain =\
 "/data/corpora/azure2/storage2/data/NIST-data/1A/\
 IARPA_MATERIAL_BASE-1A/DEV/text/mt_store/\
 umd-smt-v2.1_sent-split-v2.0-txt"
docpathtest =\
 "/data/corpora/azure2/storage2/data/NIST-data/1B/\
 IARPA_MATERIAL_BASE-1B/DEV/text/mt_store/\
 umd-smt-v2.1_sent-split-v2.0-txt"
audiopathtrain =\
 "/data/corpora/azure2/storage2/data/NIST-data/1A/\
IARPA_MATERIAL_BASE-1A/DEV/audio/mt_store/\
umd-smt-v2.1_material-asr-sw-v5.0-txt"
audiopathtest =\
 "/data/corpora/azure2/storage2/data/NIST-data/1B/\
IARPA_MATERIAL_BASE-1B/DEV/audio/mt_store/\
umd-smt-v2.1_material-asr-tl-v5.0-txt"


def createDictionary(dictionary):
    print "Creating Dictionary"

    newf = open("word_dict.txt", "w+")
    for word in dictionary:
        newf.write(word + " " + str(dictionary[word]) + "\n")
    newf.close()
    print "Done Creating Dictionary"

def createDocList(doclens, outputfile):
    newf = open(outputfile, "w+")
    for doc in doclens:
        newf.write(doc + "\n")
    newf.close()
    print "Done Making Document List"


def createCorpus(doclens, queryDict):
    print "Creating corpus"

    newf = open("corpus.txt", "w+")
    for query in queryDict:
        newf.write(query)
        for word in queryDict[query]:
            #print(word)
            newf.write(" " + word)
        newf.write("\n")
    for doc in doclens:
        newf.write(doc)
        for word in doclens[doc]:
            newf.write(" " + word)
        newf.write("\n")
    newf.close()
    print "Done creating corpus"

def createPreprocessed(doclens, dictionary, queryDict):
    print "Creating preprocessed file"

    newf = open("corpus_preprocessed.txt", "w+")
    for query in queryDict:
        newf.write(query)
        querylen = 0
        strn = ""
        for word in queryDict[query]:
            #print(word)
            if word in dictionary:
                strn += " " + str(dictionary[word])
                querylen += 1
        newf.write(" " + str(querylen) + strn)
        newf.write("\n")

    for doc in doclens:
        newf.write(doc + "  " + str(len(doclens[doc])))
        for word in doclens[doc]:
            newf.write(" " + str(dictionary[word]))
        newf.write("\n")
    newf.close()
    print "Done creating preprocessed"



def main():
    #docProcessed = {}
    docNotProcessed = {}
    dictionary = {}
    wordidcount = 0
    queryDict = {}
    delchars = ''.join(c for c in map(chr, range(256)) \
    if not c.isalnum())
    with open("query_list.txt", "r") as queryf:
        content = queryf.read().splitlines()
        for line in content:
            lineSplit = line.split("\t")
            words = re.findall(r"[\w']+", lineSplit[1])
            queryDict[lineSplit[0]] = words
            for word in words:
                if word not in dictionary:
                    dictionary[word] = wordidcount
                    wordidcount+=1

    for filename in os.listdir(docpathtrain):
        docId, ext = os.path.splitext(filename)
        docWords = []
        docWordIds = []
        with open(docpathtrain + "/" + filename, 'r') as f:
            text = f.read().replace('\n',' ')
            words = re.findall(r"[\w']+", text)
            for word in words:
                if word not in dictionary:
                    dictionary[word] = wordidcount
                    wordidcount += 1
                docWords.append(word)
        docNotProcessed[docId] = docWords

    for filename in os.listdir(audiopathtrain):
        docId, ext = os.path.splitext(filename)
        docWords = []
        docWordIds = []
        with open(audiopathtrain + "/" + filename, 'r') as f:
            text = f.read().replace('\n',' ')
            #tokenizer = TreebankWordTokenizer()
            #words = tokenizer.tokenize(text)
            words = re.findall(r"[\w']+", text)
            for word in words:
                if word not in dictionary:
                    dictionary[word] = wordidcount
                    wordidcount += 1
                docWords.append(word)

        docNotProcessed[docId] = docWords

    createDocList(docNotProcessed, "document_list_train.txt")

    onlytraindocs = {}
    for filename in os.listdir(docpathtest):
        docId, ext = os.path.splitext(filename)
        docWords = []
        docWordIds = []
        with open(docpathtest + "/" + filename, 'r') as f:
            text = f.read().replace('\n',' ')
            tokenizer = TreebankWordTokenizer()
            words = tokenizer.tokenize(text)
            for word in words:
                if word not in dictionary:
                    dictionary[word] = wordidcount
                    wordidcount += 1
                docWords.append(word)

        docNotProcessed[docId] = docWords
        onlytraindocs[docId] = docWords

    for filename in os.listdir(audiopathtest):
        docId, ext = os.path.splitext(filename)
        docWords = []
        docWordIds = []
        with open(audiopathtest + "/" + filename, 'r') as f:
            text = f.read().replace('\n',' ')
            tokenizer = TreebankWordTokenizer()
            words = tokenizer.tokenize(text)
            for word in words:
                if word not in dictionary:
                    dictionary[word] = wordidcount
                    wordidcount += 1
                docWords.append(word)
        docNotProcessed[docId] = docWords
        onlytraindocs[docId] = docWords

    createDocList(onlytraindocs, "document_list_test.txt")

    print "Done Processing Files"
    if "\n" in dictionary:
        del dictionary["\n"]
    createDictionary(dictionary)
    print("Vocab Size: " + str(len(dictionary.keys())))
    print("Highest Word index: " + str(wordidcount))
    createCorpus(docNotProcessed, queryDict)
    createPreprocessed(docNotProcessed, dictionary, \
     queryDict)
    print "Done with data processing"

main()
```

12.1.4 create_files.py

```
# coding: utf-8

import os
import sys
import argparse
import math

parser = argparse.ArgumentParser()
parser.add_argument("system")
args = parser.parse_args()

trainfile = "MaterialCorpus/sw-train.txt"
validfile = "MaterialCorpus/sw-valid.txt"
testfile = "MaterialCorpus/tg-test.txt"

train_query_annotation = \
 "/data/corpora/azure2/storage2/data/NIST-data/1A/\
 IARPA_MATERIAL_BASE-1A/DEV_ANNOTATION1/query_annotation.tsv"
test_query_annotation = \
 "/data/corpora/azure2/storage2/data/NIST-data/1B/\
 IARPA_MATERIAL_BASE-1B/DEV_ANNOTATION1/query_annotation.tsv"

systemName = args.system
path_to_datasets_sw = \
 "/data/projects/material/system_combination/SW_Matcher\
_Q1Q2Q3_All/SW_Matcher_Q1_All/" + systemName + "/q-"
path_to_datasets_tl =\
 "/data/projects/material/system_combination/TL_Matcher\
 _Q1Q2Q3_All/TL_Matcher_Q1_All/" + systemName + "/q-"

train_query_list =\
 "/data/corpora/azure2/storage2/data/NIST-data/1A/\
 IARPA_MATERIAL_BASE-1A/QUERY1/query_list.tsv"
test_query_list =\
 "/data/corpora/azure2/storage2/data/NIST-data/1B/I\
 ARPA_MATERIAL_BASE-1B/QUERY1/query_list.tsv"

doc_list_train = "document_list_train.txt"
doc_list_test = "document_list_test.txt"
corpus = "corpus.txt"
results_file = "system_results.txt"


def createRelevance(query_annotation, output_file_rel,\
 output_file_txt, docList, querylistfile, path_to_datasets):
    yesDict = {}
    with open(query_annotation, 'r') as f:
        content = f.read().splitlines()
        content.pop(0) # remove id identifiers
        for line in content:
            lineSplit = line.split()
            query = lineSplit[0]
            doc = lineSplit[1]
            if query in yesDict:
                yesDict[query][doc] = 1
            else:
                yesDict[query] = {}
                yesDict[query][doc] = 1
    print "Query Annotation Processed"

    doctextdict = {}
    with open(corpus, 'r') as f:
        content = f.read().splitlines()
        for line in content:
            lineSplit = line.split()
            doctextdict[lineSplit[0]] = [unicode(item,\
             "utf-8").encode("utf-8") \
             for item in lineSplit[1:]]

    # get names associated with queries
    querydict = {}
    with open(train_query_list, 'r') as f:
        content = f.read().splitlines()
        content.pop(0)
        print "Len of Train Queries: " + str(len(content))
        for line in content:
            lineSplit = line.split("\t")
            querydict[lineSplit[0]] = lineSplit[1]
    with open(test_query_list, 'r') as f:
        content = f.read().splitlines()
        content.pop(0)
        print "Len of Test Queries: " + str(len(content))
        for line in content:
            lineSplit = line.split("\t")
            querydict[lineSplit[0]] = lineSplit[1]

    newf = open(output_file_rel, "w+")
    newtxt = open(output_file_txt, "w+")
    posCount = 0
    negCount = 0
    for query in yesDict:
        querydocs = []
        with open(path_to_datasets + query + ".trec", 'r') \
         as querydocsfile:
            content = querydocsfile.read().splitlines()
            for line in content:
                lineSplit = line.split("\t")
                querydocs.append(lineSplit[2])
        for doc in querydocs:
            if (doc in doctextdict) and (doc in docList):
                if (doc in yesDict[query]):
                    posCount +=1
                    newf.write("1 " + query + " " + doc +\
                     "\n")
                    newtxt.write("1 \t " + querydict[query]\
                     + " \t  " + " ".join(doctextdict[doc])\
                     + "\n")
                    yesDict[query][doc] = 0
                else:
                    negCount += 1
                    newf.write("0 "+query + " " + doc + "\n")
                    newtxt.write("0 \t "+querydict[query]+ \
                    " \t "+" ".join(doctextdict[doc]) + "\n")
    newf.close()
    newtxt.close()
    with open(results_file, 'a') as results:
        results.write(str(posCount) +"\t")
        results.write(str(negCount) + "\t")
    print "Positive labels: " + str(posCount)
    print "Negative labels: " + str(negCount)


def main():
    with open(results_file, 'a') as results:
        results.write(systemName+"\t")
    # read in document list
    with open(doc_list_train, 'r') as f:
        docListTrain = f.read().splitlines()
    print "Doc list read in"
    lenTrain = len(docListTrain)
    lenValid = int(math.floor(0.2 * lenTrain))
    lenTrain = lenTrain - lenValid
    print "Len Train: " + str(lenTrain)
    print "Len Valid: " + str(lenValid)
    docListValid = docListTrain[:lenValid]
    docListTrain = docListTrain[lenValid:]

    print "Creating train relevance"

    createRelevance(train_query_annotation,\
     "relation_train.txt", trainfile, docListTrain,\
      train_query_list, path_to_datasets_sw)
    createRelevance(train_query_annotation,\
     "relation_valid.txt", validfile, docListValid,\
      train_query_list, path_to_datasets_sw)

    with open(doc_list_test, 'r') as f:
        docListTest = f.read().splitlines()
        print "Len Test: " + str(len(docListTest))
    print "Creating test relevance"
    createRelevance(test_query_annotation,\
     "relation_test.txt", testfile, docListTest,\
      test_query_list, path_to_datasets_tl)

    print "Done..."

main()
```

12.1.5 transfer_to_mz_format.py

```python
# coding: utf-8

import os
import sys
from argparse import ArgumentParser

parser = ArgumentParser()
parser.add_argument("-train", dest="trainfile")
parser.add_argument("-valid", dest="validfile")
parser.add_argument("-test", dest="testfile")

trainfile = "sw-train.txt"
validfile = "sw-valid.txt"
testfile = "tg-test.txt"

basedir = './MaterialCorpus/'
dstdir = './MaterialCorpusMZ/'
infiles = [ basedir + trainfile, basedir + validfile, \
basedir + testfile ]
outfiles = [ dstdir + trainfile, dstdir + validfile, \
dstdir + testfile ]

for idx, infile in enumerate(infiles):
    outfile = outfiles[idx]
    fout = open(outfile, 'w')
    for line in open(infile, 'r'):
        r = line.strip().split('\t')
        fout.write('%s\t%s\t%s\n' % (r[2], r[0], r[1]))
    fout.close()
```

12.1.6 run_drmm.sh

```sh
#cd ../

#currpath=`pwd`
# train the model
python ../MatchZoo/matchzoo/main.py --phase train --model_file\
 drmm_material.config




# predict with the model

python ../MatchZoo/matchzoo/main.py --phase predict --model_file\
 drmm_material.config
```

12.1.7 generate_q-querys.py

```python
# coding: utf-8

import os
import sys
import argparse
import math

parser = argparse.ArgumentParser()
parser.add_argument("system")
args = parser.parse_args()
systemName = args.system

outputdir = "predictions/" + systemName + "_predictions"
outputtemp = outputdir + "/q-"
inputfile = "predict.test.drmm.material.txt"
os.mkdir(outputdir)

querylist = "test_query_list.txt"

def main():
    rankings = {}
    with open(querylist) as querys:
        content = querys.read().splitlines()
        for line in content:
            lineSplit = line.split("\t")
            query = lineSplit[0]
            rankings[query] = []
    with open(inputfile) as f:
        content = f.read().splitlines()
        for line in content:
            lineSplit = line.split("\t")
            query = lineSplit[0]
            docrank = (lineSplit[2], lineSplit[4])
            if query in rankings:
                rankings[query].append(docrank)
            else:
                rankings[query] = [docrank]

    for query in rankings:
        newfile = open(outputtemp + query + ".tsv", "w+")
        newfile.write(query + "\n")
        for doc in rankings[query]:
            newfile.write(doc[0] + "\t" + doc[1] + "\n")
        newfile.close()


```

```

main()
```

12.1.8 create_aqwv_folder.py

```
import sys,os

#SYSTEM=sys.argv[1]
CUTOFF=int(sys.argv[2])
#inputdir="predictions/"+SYSTEM+"_predictions"
inputdir=sys.argv[1]
print("CUTOFF " + str(CUTOFF))
#os.mkdir("cutoffs/")

def main():
    for filename in os.listdir(inputdir):
        #queryid, ext = os.path.splitext(filename)
        inputf = open(inputdir+filename, "r")
        outputf = open("cutoffs/"+filename, "w+")
        content = inputf.read().splitlines()
        outputf.write(content[0]+"\n")
        content.pop(0)
        count=0
        while(count<CUTOFF and count<len(content)):
            outputf.write(content[count] + "\n")
            count+=1
        #print(count)
        outputf.close()
        inputf.close()

main()
```

12.1.9 calculate_best_cutoffs.py

```
import sys

def main():
    systems = {}
    with open("aqwv_outputs.txt", "r") as f:
        content = f.read().splitlines()
        content.pop(0)
        for line in content:
            lineSplit = line.split()
            system = lineSplit[0]
            qwv = lineSplit[5]
            if (system in systems):
                if (qwv > systems[system][0]):
                    systems[system] = [qwv, line]
            else:
                systems[system] = [qwv, line]
    with open("best_cutoffs.txt", "w+") as newf:
        newf.write("System  Cutoff  Queries Pmiss   Pfa\
            QWV\n")
        for system in systems:
            newf.write(systems[system][1] + "\n")
        newf.close()


main()
```

12.1.10 drmm_material.config

```
{
  "net_name": "DRMM",
  "global":{
      "model_type": "PY",
      "weights_file": "weights/drmm.material.weights",
      "save_weights_iters": 10,
      "num_iters": 400,
      "display_interval": 10,
      "test_weights_iters": 400,
      "optimizer": "adadelta",
      "learning_rate": 1.0
  },
  "inputs": {
    "share": {
        "text1_corpus": "corpus_preprocessed.txt",
        "text2_corpus": "corpus_preprocessed.txt",
        "use_dpool": false,
        "embed_size": 300,
        "embed_path": "embed.idf",
        "vocab_size": 51535,
        "train_embed": false,
        "target_mode": "ranking",
        "hist_size": 60,
        "text1_maxlen": 10,
        "text2_maxlen": 40
    },
    "train": {
        "input_type": "DRMM_PairGenerator",
        "phase": "TRAIN",
        "use_iter": false,
        "query_per_iter": 50,
        "batch_per_iter": 5,
        "batch_size": 100,
        "relation_file": "relation_train.txt",
        "hist_feats_file": "relation_train.hist-60.txt"
    },
    "valid": {
        "input_type": "DRMM_ListGenerator",
        "phase": "EVAL",
        "batch_list": 10,
        "relation_file": "relation_valid.txt",
        "hist_feats_file": "relation_valid.hist-60.txt"
    },
    "test": {
        "input_type": "DRMM_ListGenerator",
        "phase": "EVAL",
        "batch_list": 10,
        "relation_file": "relation_test.txt",
        "hist_feats_file": "relation_test.hist-60.txt"
    },
    "predict": {
        "input_type": "DRMM_ListGenerator",
        "phase": "PREDICT",
        "batch_list": 10,
        "relation_file": "relation_test.txt",
        "hist_feats_file": "relation_test.hist-60.txt"
    }
  },
  "outputs": {
    "predict": {
      "save_format": "TREC",
      "save_path": "predict.test.drmm.material.txt"
    }
  },
  "model": {
    "model_path": "./../MatchZoo/matchzoo/models/",
    "model_py": "drmm.DRMM",
    "setting": {
        "num_layers": 2,
        "hidden_sizes": [20, 1],
        "dropout_rate": 0.0
    }
  },
  "losses": [
    {
       "object_name": "rank_hinge_loss" ,
       "object_params": {
            "margin": 1.0
       }
    }
  ],
  "metrics": [ "ndcg@3", "ndcg@5", "map", "precision@20",\
   "ndcg@20" ]
}
```

12.1.11 dssm_material.config

```
{
  "net_name": "DSSM",
  "global":{
      "model_type": "PY",
      "weights_file": "weights/dssm.material.weights",
      "save_weights_iters": 10,
      "num_iters": 50,
      "display_interval": 10,
      "test_weights_iters": 50,
      "optimizer": "adam",
      "learning_rate": 0.001
  },
  "inputs": {
    "share": {
        "text1_corpus": "corpus_preprocessed.txt",
        "text2_corpus": "corpus_preprocessed.txt",
        "word_triletter_map_file": "word_triletter_map.txt",
        "target_mode": "ranking",
        "vocab_size": 51535,
        "embed_size": 1
    },
    "train": {
        "input_type": "Triletter_PairGenerator",
        "dtype": "dssm",
        "phase": "TRAIN",
        "use_iter": false,
        "query_per_iter": 50,
        "batch_per_iter": 5,
        "batch_size": 100,
        "relation_file": "relation_train.txt"
    },
    "valid": {
        "input_type": "Triletter_ListGenerator",
        "dtype": "dssm",
        "phase": "EVAL",
        "batch_list": 10,
        "relation_file": "relation_valid.txt"
    },
    "test": {
        "input_type": "Triletter_ListGenerator",
        "dtype": "dssm",
        "phase": "EVAL",
        "batch_list": 10,
        "relation_file": "relation_test.txt"
    },
    "predict": {
        "input_type": "Triletter_ListGenerator",
        "dtype": "dssm",
        "phase": "PREDICT",
        "batch_list": 10,
        "relation_file": "relation_test.txt"
    }
  },
  "outputs": {
    "predict": {
      "save_format": "TREC",
      "save_path": "predict.test.dssm.material.txt"
    }
  },
  "model": {
    "model_path": "./../MatchZoo/matchzoo/models/",
    "model_py": "dssm.DSSM",
    "setting": {
        "hidden_sizes": [300],
        "dropout_rate": 0.9
    }
  },
  "losses": [
    {
       "object_name": "rank_hinge_loss" ,
       "object_params": {
            "margin": 1.0
       }
    }
  ],
  "metrics": [ "ndcg@3", "ndcg@5", "map", "precision@20",\
   "ndcg@20" ]
}
```

12.1.12 cdssm_material.config

```
{
  "net_name": "DSSM",
  "global":{
      "model_type": "PY",
      "weights_file": "./weights/dssm.material.weights",
      "save_weights_iters": 10,
      "num_iters": 50,
      "display_interval": 10,
      "test_weights_iters": 50,
      "optimizer": "adadelta",
      "learning_rate": 1.0
  },
  "inputs": {
    "share": {
        "text1_corpus": "corpus_preprocessed.txt",
        "text2_corpus": "corpus_preprocessed.txt",
        "word_triletter_map_file": "word_triletter_map.txt",
        "vocab_size": 51535,
        "embed_size": 50,
        "train_embed": true,
        "target_mode": "ranking",
        "text1_maxlen": 50,
        "text2_maxlen": 200
    },
    "train": {
        "input_type": "Triletter_PairGenerator",
        "dtype": "cdssm",
        "phase": "TRAIN",
        "use_iter": false,
        "query_per_iter": 50,
        "batch_per_iter": 5,
        "batch_size": 100,
        "relation_file": "relation_train.txt"
    },
    "valid": {
        "input_type": "Triletter_ListGenerator",
        "dtype": "cdssm",
        "phase": "EVAL",
        "batch_list": 10,
        "relation_file": "relation_valid.txt"
    },
    "test": {
        "input_type": "Triletter_ListGenerator",
        "dtype": "cdssm",
        "phase": "EVAL",
        "batch_list": 10,
        "relation_file": "relation_test.txt"
    },
    "predict": {
        "input_type": "Triletter_ListGenerator",
        "dtype": "cdssm",
        "phase": "PREDICT",
        "batch_list": 10,
        "relation_file": "relation_test.txt"
    }
  },
  "outputs": {
```

```
    "predict": {
      "save_format": "TREC",
      "save_path": "predict.test.cdssm.material.txt"
    }
  },
  "model": {
    "model_path": "./../MatchZoo/matchzoo/models/",
    "model_py": "cdssm.CDSSM",
    "setting": {
        "kernel_count": 50,
        "kernel_size": 3,
        "hidden_sizes": [10],
        "dropout_rate": 0.9
    }
  },
  "losses": [
    {
       "object_name": "rank_hinge_loss" ,
       "object_params": {
            "margin": 1.0
       }
    }
  ],
  "metrics": [ "ndcg@3", "ndcg@5", "map", "precision@20",\
   "ndcg@20" ]
}
```

12.1.13 duet_material.config

```
{
  "net_name": "DUET",
  "global":{
      "model_type": "PY",
      "weights_file": "weights/duet.material.weights",
      "save_weights_iters": 10,
      "num_iters": 400,
      "display_interval": 10,
      "test_weights_iters": 400,
      "optimizer": "adam",
      "learning_rate": 0.001
  },
  "inputs": {
    "share": {
        "text1_corpus": "corpus_preprocessed.txt",
        "text2_corpus": "corpus_preprocessed.txt",
        "use_dpool": false,
        "embed_size": 50,
        "embed_path": "embed_glove_d50_norm",
        "vocab_size": 51821,
        "train_embed": false,
        "target_mode": "ranking",
        "text1_maxlen": 20,
        "text2_maxlen": 40
    },
    "train": {
        "input_type": "PairGenerator",
        "phase": "TRAIN",
        "use_iter": false,
        "query_per_iter": 50,
        "batch_per_iter": 5,
        "batch_size": 100,
        "relation_file": "relation_train.txt"
    },
    "valid": {
        "input_type": "ListGenerator",
        "phase": "EVAL",
        "batch_list": 10,
        "relation_file": "relation_valid.txt"
    },
    "test": {
        "input_type": "ListGenerator",
        "phase": "EVAL",
        "batch_list": 10,
        "relation_file": "relation_test.txt"
    },
    "predict": {
        "input_type": "ListGenerator",
        "phase": "PREDICT",
        "batch_list": 10,
        "relation_file": "relation_test.txt"
    }
  },
  "outputs": {
    "predict": {
      "save_format": "TREC",
      "save_path": "predict.test.duet.material.txt"
    }
  },
  "model": {
    "model_path": "./../MatchZoo/matchzoo/models/",
    "model_py": "duet.DUET",
    "setting": {
        "lm_kernel_count": 32,
        "lm_hidden_sizes": [30],
        "dm_kernel_count": 32,
        "dm_kernel_size": 3,
        "dm_q_hidden_size": 32,
        "dm_d_mpool": 3,
        "dm_hidden_sizes": [30],
        "lm_dropout_rate": 0.5,
        "dm_dropout_rate": 0.5
    }
  },
  "losses": [
    {
       "object_name": "rank_hinge_loss",
       "object_params": { "margin": 1.0 }
    }
  ],
  "metrics": [ "ndcg@3", "ndcg@5", "map", "precision@20",\
   "ndcg@20" ]
}
```

12.1.14 knrm_material.config

```
{
  "net_name": "KNRM",
  "global":{
      "model_type": "PY",
      "weights_file": "weights/knrm.material.weights",
      "save_weights_iters": 10,
      "num_iters": 50,
      "display_interval": 10,
      "test_weights_iters": 50,
      "optimizer": "adam",
      "learning_rate": 0.001
  },
  "inputs": {
    "share": {
        "text1_corpus": "corpus_preprocessed.txt",
        "text2_corpus": "corpus_preprocessed.txt",
        "use_dpool": false,
        "embed_size": 300,
        "embed_path": "embed_glove_d300_norm",
        "vocab_size": 51468,
        "train_embed": true,
        "target_mode": "ranking",
        "text1_maxlen": 10,
        "text2_maxlen": 40
    },
    "train": {
        "input_type": "PairGenerator",
        "phase": "TRAIN",
        "use_iter": false,
        "query_per_iter": 50,
        "batch_per_iter": 5,
        "batch_size": 100,
        "relation_file": "relation_train.txt"
    },
    "valid": {
        "input_type": "ListGenerator",
        "phase": "EVAL",
        "batch_list": 10,
        "relation_file": "relation_valid.txt"
    },
    "test": {
        "input_type": "ListGenerator",
        "phase": "EVAL",
        "batch_list": 10,
        "relation_file": "relation_test.txt"
    },
    "predict": {
        "input_type": "ListGenerator",
        "phase": "PREDICT",
        "batch_list": 10,
        "relation_file": "relation_test.txt"
    }
  },
  "outputs": {
    "predict": {
      "save_format": "TREC",
      "save_path": "predict.test.knrm.material.txt"
    }
  },
  "model": {
    "model_path": "./../MatchZoo/matchzoo/models/",
    "model_py": "knrm.KNRM",
    "setting": {
        "kernel_num": 21,
        "sigma": 0.1,
        "exact_sigma": 0.001,
        "dropout_rate": 0.0
    }
  },
  "losses": [
    {
       "object_name": "rank_hinge_loss",
       "object_params": { "margin": 1.0 }
    }
  ],
  "metrics": [ "ndcg@3", "ndcg@5", "map", "precision@20",\
   "ndcg@20" ]
}
```

12.1.15 matchpyramid_material.config

```
{
  "net_name": "MatchPyramid",
  "global":{
     "model_type": "PY",
      "weights_file": "weights/matchpyramid.material.weights",
      "save_weights_iters": 10,
      "num_iters": 100,
      "display_interval": 10,
      "test_weights_iters": 100,
      "optimizer": "adadelta",
      "learning_rate": 1.0
  },
  "inputs": {
    "share": {
        "text1_corpus": "corpus_preprocessed.txt",
        "text2_corpus": "corpus_preprocessed.txt",
        "use_dpool": true,
        "embed_size": 300,
        "embed_path": "embed_glove_d300_norm",
        "vocab_size": 51468,
        "train_embed": true,
        "target_mode": "ranking",
        "text1_maxlen": 10,
        "text2_maxlen": 40
    },
    "train": {
        "input_type": "PairGenerator",
        "phase": "TRAIN",
        "use_iter": false,
        "query_per_iter": 50,
        "batch_per_iter": 5,
        "batch_size": 100,
        "relation_file": "relation_train.txt"
    },
    "valid": {
        "input_type": "ListGenerator",
        "phase": "EVAL",
        "batch_list": 10,
        "relation_file": "relation_valid.txt"
    },
    "test": {
        "input_type": "ListGenerator",
        "phase": "EVAL",
        "batch_list": 10,
        "relation_file": "relation_test.txt"
    },
    "predict": {
        "input_type": "ListGenerator",
        "phase": "PREDICT",
        "batch_list": 10,
        "relation_file": "relation_test.txt"
    }
  },
  "outputs": {
    "predict": {
      "save_format": "TREC",
      "save_path": "predict.test.matchpyramid.material.txt"
    }
  },
  "model": {
    "model_path": "./../MatchZoo/matchzoo/models/",
    "model_py": "matchpyramid.MatchPyramid",
    "setting": {
        "kernel_count": 64,
        "kernel_size": [3, 3],
        "dpool_size": [3, 10],
        "dropout_rate": 0.95
    }
  },
  "losses": [
    {
       "object_name": "rank_hinge_loss" ,
       "object_params": {
            "margin": 1.0
       }
    }
  ],
  "metrics": [ "ndcg@3", "ndcg@5", "map", "precision@20",\
   "ndcg@20" ]
}
```

12.1.16 reformat_output.py

```python
import sys,os

systems = ['cdssm', 'drmm', 'dssm', 'duet', 'knrm',\
 'matchpyramid']
langs = ['SWTL', 'TLSW']

for syst in systems:
    with open('final_output_' + syst + '.txt','w+') as\
     outputf:
        outputf.write("Lang\tSystem\tCutoff\tPmiss\tPfa\t\
        QWV\tndcg@3\tmap\tndcg@20\tprecision@20\tndcg@5\n")
        for lang in langs:
            pathto = syst + "_system_" + lang + "/";
            sysDict = {}
            with open(pathto + "best_cutoffs.txt", 'r') as\
             cutoffsf:
                content = cutoffsf.read().splitlines()
                content.pop(0)
                for line in content:
                    lineSplit = line.split()
                    sysDict[lineSplit[0]] = [lineSplit[1],\
                     lineSplit[3], lineSplit[4], lineSplit[5]]
            with open(pathto + "system_results.txt", 'r') as\
             resultsf:
                content = resultsf.read().splitlines()
                content.pop(0)
                for line in content:
                    lineSplit = line.split()
                    sysDict[lineSplit[0]].append(lineSplit[7])
                    sysDict[lineSplit[0]].append(lineSplit[8])
                    sysDict[lineSplit[0]].append(lineSplit[9])
                    sysDict[lineSplit[0]].append(lineSplit[10])
                    sysDict[lineSplit[0]].append(lineSplit[11])
            for key in sysDict:
                outputf.write(lang + "\t" + key)
                value = sysDict[key]
                for i in range(0, len(value)):
                    outputf.write("\t" + value[i])
                outputf.write("\n")
```