Automatic Keyphrase Extraction System --- Stand alone module

Author : Navaneeth Kumar K

Description : This documentation explains about the Automatic Keyphrase Extraction System and the step by step procedure for extracting keyphrases.

Project Related Softwares installed

1) python -- version 2.7.3 already installed
2) nltk, numpy
3) orange

Project Details (Structure of the project):
The keyphrase folder contains all the python files, config files and shell script files for automatic keyphrase extraction. The input files and output files are stored inside the subfolders of the keyphrase folder.

1) *../keyphrase* --- contains training data (*.tab), posfilter rules (*.txt), python files (*.py) related to preprocessing steps and shell scripts (*.sh) for each step of machine learning process.

2) *../keyphrase/transcript* --- contains audio transcript files <transcript name>.txt

3) *../keyphrase/keys* --- contains <transcript_name>.key files for the corresponding <transcript_name>.txt files. The *.key files contain the manually extracted keys of the corresponding transcript files.

4) *../keyphrase/videotextslides* --- contains subfolders for each <transcript_name> containing OCR extracted text.

5) *../keyphrase/audio_features* --- contains python files (*.py) for calculating audio features

6) *../keyphrase/video_features* --- contains python files (*.py) for calculating video features

7) *../keyphrase/ml_audio* --- contains python files (*.py) related to machine learning procedure of Keyphrase Extraction System.

8) *../keyphrase/output* ---- ouput for each step of Keyphrase Extraction is written into this folder

Following are the three broad stages for keyphrase extraction:

1) **Pre-processing Stage:**
   This takes the text transcript file as input and provides the candidate phrases as output. Following are the steps involved in pre-processing.
   a. Identify Part-Of-Speech Tagging (POS Tagging) & Stem the tokens to their root words.
   b. Extract candidate grams for keyphrase extraction based on POS filter rules and stopwords

2) **Feature Extraction Stage:**
   This stage consists of following steps:
   a. Audio Feature Extraction : Calculating audio features of candidate grams
   b. Video Feature Extraction : Calculating video features of candidate grams

3) **Machine Learning and Keyphrase Extraction Stage:**
   This stage consists of following steps:
   a. Preparing test data for the given transcript file
   b. Preparing training data for machine learning
   c. Executing machine learning procedure and extracting keyphrases

Each Step of the above said stages are explained in detail below.

1.a . **Identify Part-Of-Speech Tagging & Stem the tokens to their root words:**

Audio: The script takes transcript file as input from "transcript" folder and performs POS tagging and stemming

Python file: ../keyphrase/pos.py

Dependant files: nil

Command:

***python pos.py* <transcript_name>*.txt* <*Stemmer name*> <*enable/disable POS tagging*>**

e.g. 1. *python pos.py CG1.txt Porter-Stemmer enable*
the above sample script will perform POS tagging and porter stemming.
e.g 2. *python pos.py CG1.txt none enable*
the above script will perform POS tagging and no stemming
e.g. 3 *python pos.py CG1.txt Lancaster-Stemmer disable*
the above script will perform Lancaster-Stemming and no POS tagging

Input : <transcript_name>.txt present in transcript folder
Output: The output of this step consists of 3 files under "output" folder
a. "<transcript_name>_tagged.txt – stemmed token along with its POS tagging.
b. "<transcript_name>_untagged.txt – only stemmed token of the original transcript.
c. <transcript_name>_root.txt – contains stemmed version and root version of tokens

Shell script: ../keyphrase/posGen.sh. This shell script executes *python pos.py <transcript name>.txt Porter-Stemmer enable* command.

Command :

./posGen.sh '<transcript_name>.txt'

e.g. 1 . *./posGen.sh 'CG1.txt'*

Video: The script takes text extracted from text slides of video as input from "*videotextslides*" folder and performs POS tagging and stemming

Python file : ../keyphrase/text_stem.py
Dependant files: ../stopwords_video.py
Command :

python text_stem.py <transcript_name> <stemmer_name>

e.g.1 *python text_stem.py CG1.txt Porter-Stemmer*

Input : OCR extracted text from video frames present in the < transcript_name> folder inside the videotextslides folder

Output: The output of this step creates a new folder under "output folder" with the same name as the <transcript_name>. The folder contains contains the output files for each input OCR files which contains the stemmed token of the extracted text.

Shell script: ../keyphrase/textStemGen.sh. This shell script executes *python text_stem.py <transcript_name>.txt Porter-Stemmer* command.

Command :

./textStemGen.sh '<transcript_name>.txt'

e.g. 1 . *./textStemGen.sh 'CG1.txt'*

1.b. **Extract candidate grams for keyphrase extraction:**

Audio: The script extract upto ***<maximum_gram_size>*** grams that satisfy the pos filtering criteria given in pos_filter.txt file. In this case all the n-grams that satify the filter rules will be considered for candidate ngrams.

Python file: ../keyphrase/grams_pos.py

Dependant files: ../keyphrase/stop_words.py, ../keyphrase/pos_filter.txt

Command:

python grams_pos.py <transcript_name> <maximum_gram_size> <pos_filter_defined_file> <stemmer_name>

e.g.1. *python grams_pos.py CG1.txt 4 pos_filter.txt "Porter-Stemmer"*

pos_filter.txt file:

```
ANY,ANY,ANY,NN
ANY,ANY,ANY,IN
ANY,ANY,ANY,VB
ANY,ANY,NN
ANY,ANY,VB
ANY,ANY,JJ
JJ,NN
JJ,JJ
NN,NN
IN,NN
CC,NN
VB,NN
RP,NN
RB,NN
NN,VB
NN
VB
```

Input : <transcript_name>_tagged.txt present in output folder

Output: The output of this step consists of 2 files under "output" folder

a. "<transcript_name>_grams.txt – contains the stemmed candidate grams

b. "<transcript_name>_gramsroot.txt – contains the stemmed candidate grams along with their root forms.

Shell script: ../keyphrase/grams_posGen.sh. This shell script executes *python grams_pos.py <transcript_name>.txt 4 pos_filter.txt "Porter-Stemmer"* command.

Command :

./grams_posGen.sh '<transcript_name>.txt'

e.g. 1 . *./grams_posGen.sh 'CG1.txt'*
Video: The script extracts upto <*max_gram*> grams from stemmed ocr text.
Python file: ../keyphrase/extractgrams.py
Dependant files: ../keyphrase/stop_words.py, ../keyphrase/stopwords_video.py
Command:
python extractgrams.py <transcript_name> <max_gram>

e.g.1. *python extractgrams.py CG1.txt 4*

Input : Files containing **s**temmed text **of** OCR extracted text present in the <transcript_name> folder inside the output folder
Output: The output of this step consists of 2 files under "output" folder
a. "<transcript_name>_vgrams.txt – contains the stemmed candidate grams
b. "<transcript_name>_vgramsroot.txt – contains the stemmed candidate grams along with their root forms.

Shell script: ../keyphrase/extractGrams.sh. This shell script executes *python extractgrams.py <transcript_name> 4* command.
Command :
../keyphrase/extractGrams.sh '<transcript_name>.txt'

e.g. 1 . *./extractGrams.sh 'CG1.txt'*

With this the pre-processing and extraction of candidate n-grams in completed.

2. a. Audio Feature Extraction:
There are currently 5 features available for extracting keyphrases from audio transcript. For each candidate phrase that is extracted using pre-processing stage, the feature value is calculated.

2.a.1 **Cscore: –** This is useful to identify nested collocations thereby improving the accuracy of keyphrase extraction
Python file: ../keyphrase/audio_features/cscore.py
Dependant files: nil
Command:
python audio_features/cscore.py* <transcript_name>*.txt <Stemmer name>

e.g.1. *python audio_features/cscore.py CG1.txt "Porter-Stemmer"*

Input : <transcript_name>_untagged.txt and <transcript_name>_grams.txt present in output folder
Output: <transcript_name>_cscore.txt file in the output folder. This will contain cscore for each candidate phrase extracted.

Shell script: ../keyphrase/cscoreGen.sh. This shell script executes *python audio_features/cscore.py <transcript_name>.txt "Porter-Stemmer"* command.
Command :
./cscoreGen.sh '<transcript_name>.txt'

e.g. 1 . *./cscoreGen.sh 'CG1.txt'*

2.a.2 **Localspan: -** Phrases that are sub-topics, are often clustered on a specific segments of a lecture. We define Local Span as a feature to identify such locally clustered phrases.

Python file: ../keyphrase/audio_features/localspan.py
Dependant files: ../keyphrase/stem.py
Command:
python audio_features/localspan.py <transcript_name> <max_gram> <stemmer_name>

e.g.1. *python audio_features/localspan.py CG1.txt 4 "Porter-Stemmer"*

Input : <transcript_name>_untagged.txt and <transcript_name>_grams.txt present in output folder
Output: <transcript_name>_localspan.txt file in the output folder. This will contain localspan scores for each candidate phrase extracted.

Shell script: ../keyphrase/localspanGen.sh. This shell script executes *python audio_features/localspan.py <transcript_name>.txt 4 "Porter-Stemmer"* command.
Command :
./localspanGen.sh '<transcript_name>.txt'

e.g. 1 . *./localspanGen.sh 'CG1.txt'*

2.a.3 **Cuewords :-** Phrases following cuewords in the transcript are identified and scored.

Python file: ../keyphrase/audio_features/cue.py
Dependant files: ../keyphrase/audio_features/cuewords.py, ../keyphrase/stop_words.py, ../keyphrase/stem.py
Command:
python audio_features/cue.py <transcript_name> <stemmer_name><max_gram>

e.g.1. *python audio_features/cue.py CG1.txt "Porter-Stemmer" 4*

Input : <transcript_name>_untagged.txt present in output folder
Output: <transcript_name>_cue.txt file in the output folder. This will contain scores for cuewords.

Shell script: ../keyphrase/cueGen.sh. This shell script executes *python audio_features/cue.py <transcript_name>.txt "Porter-Stemmer" 4* command.
Command :
./cueGen.sh '<transcript_name>.txt'
e.g. 1 . *./cueGen.sh 'CG1.txt'*

2.a.4 **Tf-idf :-** Tf-idf values are calculated for candidate phrases present in the input file.

Python file: ../keyphrase/audio_features/tfidf.py
Dependant files: ../keyphrase/stop_words.py, ../keyphrase/stem.py
Command:
python audio_features/tfidf.py <stemmer_name> <max_gram> <transcript_name>

e.g.1. *python audio_features/tfidf.py "Porter-Stemmer" 4 "CG1.txt"*

Input : <transcript_name>.txt present in transcript folder
Output: <transcript_name>_tfidf.txt file in the output folder. This will contain tfidf scores for phrases.

Shell script: ../keyphrase/tfidfGen.sh. This shell script executes *python audio_features/tfidf.py "Porter-Stemmer" 4 <transcript_name>.txt* command.
Command :
./tfidfGen.sh '<transcript_name>.txt'
e.g. 1 . *./tfidfGen.sh' CG1.txt'*

2.a.5.1 **Dispersion** (Revised method)– Phrases that are topics seems to be dispersed throughout the document. Dispersion feature calculates dispaersion score for the phrases.

Python file: ../keyphrase/audio_features/dispersiontest.py
Dependant files: ../keyphrase/audio_features/dispersionmyversion60.py
Command:
python audio_features/dispersiontest.py <transcript_name>

e.g.1. *python audio_features/dispersiontest.py CG1.txt*

Input : <transcript_name>_untagged.txt and <transcript_name>_grams.txt present in output folder
Output: <transcript_name>_disp.txt file in the output folder. This will contain dispersion scores for phrases.

Shell script: ../keyphrase/*dispersionGen*.sh. This shell script executes *python audio_features/dispersiontest.py <transcript_name>.txt* command.
Command :
./dispersionGen.sh '<transcript_name>.txt'
e.g. 1 . *./dispersionGen.sh' CG1.txt'*

2.a.5.2 **Dispersion** (old method) – Phrases that are topics seems to be dispersed throughout the document. Dispersion feature calculates dispaersion score for the phrases.

Python file: ../keyphrase/audio_features/dispersion.py
Dependant files: ../keyphrase/stem.py
Command:
python audio_features/dispersion.py <transcript_name> <max_gram> <stemmer_name>

e.g.1. *python audio_features/dispersion.py CG1.txt 4 "Porter-Stemmer"*

Input : <transcript_name>_untagged.txt and <transcript_name>_grams.txt present in output folder
Output: <transcript_name>_disp.txt file in the output folder. This will contain dispersion scores for phrases.

Shell script: ../keyphrase/dispersionGen_old.sh. This shell script executes *python*

audio_features/dispersiontest.py <transcript_name>.txt 4 "Porter-Stemmer" command.

Command :

./dispersionGen_old.sh '<transcript_name>.txt'

e.g. 1 . ./dispersionGen_old.sh' CG1.txt'

2.b Video Features Extaction:

There are currently 3 features available for extracting keyphrases from OCR texts. For each candidate phrase that is extracted using pre-processing stage, the feature value is calculated.

2.b.1 **Contiguous Occurance Ratio :** Measures the contiguous occurrence of phrases in the OCR extracted silde content.

Python file: ../keyphrase/audio_features/contiguous_count_ratio.py

Dependant files: NIL

Command:

python video_features/contiguous_count_ratio.py <transcript_name> <output_file_extension>

***e.g.1** : python video_features/contiguous_count_ratio.py CG1.txt "_cont.txt"*

Input : Stemmed OCR files in <transcript_name> folder inside output folder. <transcript_name>_vgrams.txt present in output folder

Output: <transcript_name>_cont.txt file in the output folder. This will contain scores for phrases.

Shell script: ../keyphrase/Contiguous_Occurance_RatioGen.sh. This shell script executes *python video_features/contiguous_count_ratio.py <transcript_name>.txt "_cont.txt"* command.

Command :

./Contiguous_Occurance_RatioGen.sh '<transcript_name>.txt'

e.g. 1 . ./Contiguous_Occurance_RatioGen.sh 'CG1.txt'

2.b.2) **Frequency Occurance Ratio :** Measures the frequency of phrases across the OCR extracted slide contents

Python file: ../keyphrase/audio_features/freq_occur_ratio.py

Dependant files: NIL

Command:

python video_features/freq_occur_ratio.py <transcript_name> <output_file_extension>

***e.g.1** : python video_features/freq_occur_ratio.py CG1.txt _freq.txt*

Input : Stemmed OCR files in <transcript_name> folder inside output folder. <transcript_name>_vgrams.txt present in output folder

Output: <transcript_name>_freq.txt file in the output folder. This will contain scores for phrases.

Shell script: ../keyphrase/freq_occur_RatioGen.sh. This shell script executes *python video_features/freq_occur_ratio.py <transcript_name>.txt _freq .txt* command.

Command :

./freq_occur_RatioGen.sh '<transcript_name>.txt'

e.g. 1 . ./freq_occur_RatioGen 'CG1.txt'

2.b.3) **Phrase size :** Calculates the maximum height of a particular phrase in the OCR extracted slides

Python file: ../keyphrase/audio_features/max_height.py
Dependant files: NIL
Command:

python video_features/max_height.py <transcript_name> <output_file_extension> <inputfolder> <outputfolder> <extension>

e.g.1 *: python video_features/max_height.py CG1.txt output output "_phraseSize.txt"*

Input : Stemmed OCR files in <transcript_name> folder inside output folder. <transcript_name>_vgrams.txt present in output folder
Output: <transcript_name>_phraseSize.txt file in the output folder. This will contain scores for phrases.

Shell script: ../keyphrase/maxHeightGen.sh. This shell script executes *python video_features/max_height.py <transcript_name>.txt output output "_phraseSize.txt"* command.
Command :

./maxHeightGen.sh '<transcript_name>.txt'
e.g. 1 . ./ maxHeightGen.sh 'CG1.txt'

3. a) Preparing test data for the given transcript file

Next step is to create a .tab file with candidate keyphrases as rows and its relevant features values as the colums.

Python file: ../keyphrase/ml_audio/write.py
Dependant files: ../keyphrase/stem.py
Command:

python ml_audio/write.py <transcript_name> <list_of_features> <feature_file_extension> <stemmer_name>

e.g.1 *: python ml_audio/write.py CG1.txt 'cscore,localspan,cuewords,tfidf,dispersion' '_cscore.txt,_localspan.txt,_cue.txt,_tfidf.txt,_disp.txt' Porter-Stemmer*

Input : <transcript_name>_root.txt, <transcript_name>_vroot.txt, <transcript_name>_gramsroot.txt, <transcript_name>_vgramsroot.txt files in output folder. The feature score files mentioned in the python command <feature_file_extension> in output folder.
Output: <transcript_name>.tab file in the output folder.

Shell script: ../keyphrase/tabfileGen.sh. This shell script executes *python ml_audio/write.py <transcript_name>.txt 'cscore,localspan,cuewords,tfidf,dispersion' '_cscore.txt,_localspan.txt,_cue.txt,_tfidf.txt,_disp.txt' Porter-Stemmer* command.
Command :

./tabfileGen.sh '<transcript_name>.txt'
e.g. 1 . ./ tabfileGen.sh 'CG1.txt'

3. b) Preparing training data for machine learning

Next step is to create a training data file with candidate keyphrases as rows and its relevant features values as the colums. The last column contains the class information whether the phrase is keyphrase or not.

Python file: ../keyphrase/ml_audio/write_all.py
Dependant files: NIL
Command:

python ml_audio/write_all.py <list_of_tab_files_names> <TrainingDataSetName.tab> <inputfolder> <keysfolderLocation> <outputfolder> <StemmerName>

e.g.1 *: python ml_audio/write_all.py cg5.tab,ds7.tab,ds8.tab,ds10.tab,ds11.tab,ds12.tab,ds13.tab,ppl2.tab,ppl3.tab,ppl4.tab,ppl5.tab,ml8.tab,ml9.tab,ml10.tab,ml11.tab,ml12.tab TrainingDataSet2.tab output keys output Porter-Stemmer*

Input : <transcript_name>_gramsroot.txt, <transcript_name>_vgramsroot.txt files in output folder. The feature score files mentioned in the python command <feature_file_extension> in output folder.
Output: *<TrainingDataSetName>*.tab file in output folder.

3.c) Executing machine learning and extracting keyphrases

The current keyphrase extraction method uses naive bayes machine learning algorithm. Four different kinds of studies made on the impact of features on keyphrase extraction. The details are given below

3.c.1) Extracting Keyphrases with audio features only:

Python file: ../keyphrase/ml_audio/Naive-Bayes/ICMR_audio.py
Dependant files: ../keyphrase/ml_audio/Naive-Bayes/icmr_nbdisc.py
Command:

python ml_audio/Naive_Bayes/ICMR_audio.py <TrainingDataSet> <test data>

e.g.1 *: python ml_audio/Naive-Bayes/ICMR_audio.py trainingData.tab CG1.txt*

Input : <TrainingDataSetName>.tab present in keyphrase folder and <transcript_name>.tab present in output folder.
Output: <transcript_name>.txt file in the output folder. This file contains the extracted keyphrases

Shell script: ../keyphrase/ICMR-audio.sh. This shell script executes *python ml_audio/Naive-Bayes/ICMR_audio.py trainingData.tab <transcript_name>.tab* command.
Command :

./ICMR-audio.sh '<transcript_name>.txt' Trainingdataset.tab

e.g. 1 . *./ICMR-audio.sh 'CG1.txt' Trainingdataset.tab*

3.c.2) Extracting Keyphrase with video features only:

Python file: ../keyphrase/ml_audio/Naive-Bayes/ICMR_video.py
Dependant files: ../keyphrase/ml_audio/Naive-Bayes/icmr_nbdisc.py
Command:

python ml_audio/Naive_Bayes/ICMR_video.py <TrainingDataSet> <test data>

e.g.1 *: python ml_audio/Naive-Bayes/ICMR_video.py trainingData.tab CG1.txt*

Input : <TrainingDataSetName>.tab present in keyphrase folder and <transcript_name>.tab present in output folder.
Output: <transcript_name>.txt file in the output folder. This file contains the extracted

keyphrases

Shell script: ../keyphrase/ICMR-video.sh. This shell script executes *python ml_audio/Naive-Bayes/ICMR_video.py trainingData.tab <transcript_name>.tab* command.

Command :

./ICMR-video.sh '<transcript_name>.txt' Trainingdataset.tab

e.g. 1 . *./ICMR-video.sh 'CG1.txt' Trainingdataset.tab*

3.c.3) Extracting Keyphrases with audio and video features:

Python file: ../keyphrase/ml_audio/Naive-Bayes/ICMR_NaiveBayesStudy.py

Dependant files: ../keyphrase/ml_audio/Naive-Bayes/icmr_nbdisc.py

Command:

python ml_audio/Naive_Bayes/ICMR_NaiveBayesStudy.py <TrainingDataSet> <test data>

e.g.1 *: python ml_audio/Naive-Bayes/ICMR_NaiveBayesStudy.py trainingData.tab CG1.txt*

Input : <TrainingDataSetName>.tab present in keyphrase folder and <transcript_name>.tab present in output folder.

Output: <transcript_name>.txt file in the output folder. This file contains the extracted keyphrases

Shell script: ../keyphrase/ICMR-NaiveBayesStudy.sh. This shell script executes *python ml_audio/Naive-Bayes/ICMR_NaiveBayesStudy.py trainingData.tab <transcript_name>.tab* command.

Command :

./ICMR-NaiveBayesStudy.sh '<transcript_name>.txt' Trainingdataset.tab

e.g. 1 . *./ICMR-NaiveBayesStudy.sh 'CG1.txt' Trainingdataset.tab*

3.c.4) Extracting Keyphrases with audio and video features and applying rule-based study on it:

This is n step procedure. The steps are given below.

Step 1: Extraction of keyphrases

Python file: ../keyphrase/ml_audio/Naive-Bayes/ICMR_RBandNaiveBayesStudy.py

Dependant files: ../keyphrase/ml_audio/Naive-Bayes/icmr_nbdisc.py,
../keyphrase/ml_audio/Naive-Bayes/postprocessing_final.py

Command:

python ml_audio/Naive_Bayes/ICMR_RBandNaiveBayesStudy.py <TrainingDataSet> <test data>

e.g.1 *: python ml_audio/Naive-Bayes/ICMR_RBandNaiveBayesStudy.py trainingData.tab CG1.txt*

Input : <TrainingDataSetName>.tab -- present in keyphrase folder
<transcript_name>.tab --- present in output folder.
<transcript_name>_grams.txt. -- present in the output folder
<transcript_name>_corr.txt. -- present in the output folder

Output: <transcript_name>_a.txt. --This file contains the keyphrases extracted from audio modality. Present in output folder.
<transcript_name>_v.txt. --This file contains the extracted keyphrases from video

modality. Present in output folder.

<transcript_name>_analysis.txt. -- This file contains the list of extracted keyphrases both from audio and video modality. It also contains details like whether the keyphrase is extracted from audio or video or from both modality and whether the keyphrase is present in audio or video or in both modality.

<transcript_name>_keyCue.txt. -- This file contains the list of extracted keyphrases which are cuewords.

Shell script: ../keyphrase/ICMR_RBandNaiveBayesStudy.sh. This shell script executes *python ml_audio/Naive-Bayes/ICMR_RBandNaiveBayesStudy.py trainingData.tab <transcript_name>.tab* command.

Command :

./ICMR_RBandNaiveBayesStudy.sh '<transcript_name>.txt' Trainingdataset.tab

e.g. 1 . *./ICMR_RBandNaiveBayesStudy.sh 'CG1.txt' Trainingdataset.tab*

Step 2: Applying rules on the extracted keyphrases

Python file: ../keyphrase/ruleBased.py
Dependant files: nil
Command:

python ruleBased.py <transcript_name>

e.g.1 *: python ruleBased.py CG1.txt*

Input : <transcript_name>_analysis.txt. -- This file contains the list of extracted keyphrases both from audio and video modality. It also contains details like whether the keyphrase is extracted from audio or video or from both modality and whether the keyphrase is present in audio or video or in both modality.

<transcript_name>_keyCue.txt. -- This file contains the list of extracted keyphrases which are cuewords.

Output: <transcript_name>.txt file in the output folder. This file contains the extracted keyphrases

Shell script: ../keyphrase/ruleBased.sh. This shell script executes *python ruleBased.py <transcript_name>.txt* command.

Command :

./ruleBased.sh '<transcript_name>.txt'

e.g. 1 . *./ruleBased.sh 'CG1.txt'*