Final Report CA 006 Curriculum Gap Analysis for FAA COE TTHP

PI: Forrest Sheng Bao, University of Akron (current affiliation: Dept. of Computer Science, Iowa State University)
Contact information fsb@iastate.edu

Table of Contents

Knowing the coverage of job tasks in a curriculum is a necessary step to ensure the effectiveness and efficiency of a curriculum. In this report, we will briefly outline the major steps that we used to computationally estimate the coverage of courses given a job task. Sample results of analysis will be given in the end, along with spreadsheets of full analysis.

We model our problem as performing a text searching or matching task. Specifically, we search each job task against the materials of hundreds of courses that FAA provided to us. Based on the search result, we calculate the coverage of each task in the curriculum.

The overall flowchart is given below in Fig.1. The inputs of our system are a list of 1,000 tasks and a set of documents of FAA training courses. And the output of the system is a coverage report. There are five major steps on the flowchart:

1. Document and tasks are regularized initially.
2. Then the tokens of course documents are used to train two models. They are LDA and Word2vec.
3. Next, task query is expanded by utilizing expanded search.
4. The expanded task query call ranking model to compute documents coverage score by using different metrics.
5. At last, the coverage score of documents is used to compute the coverage report which including both the courage score of class over tasks and the task coverage score.

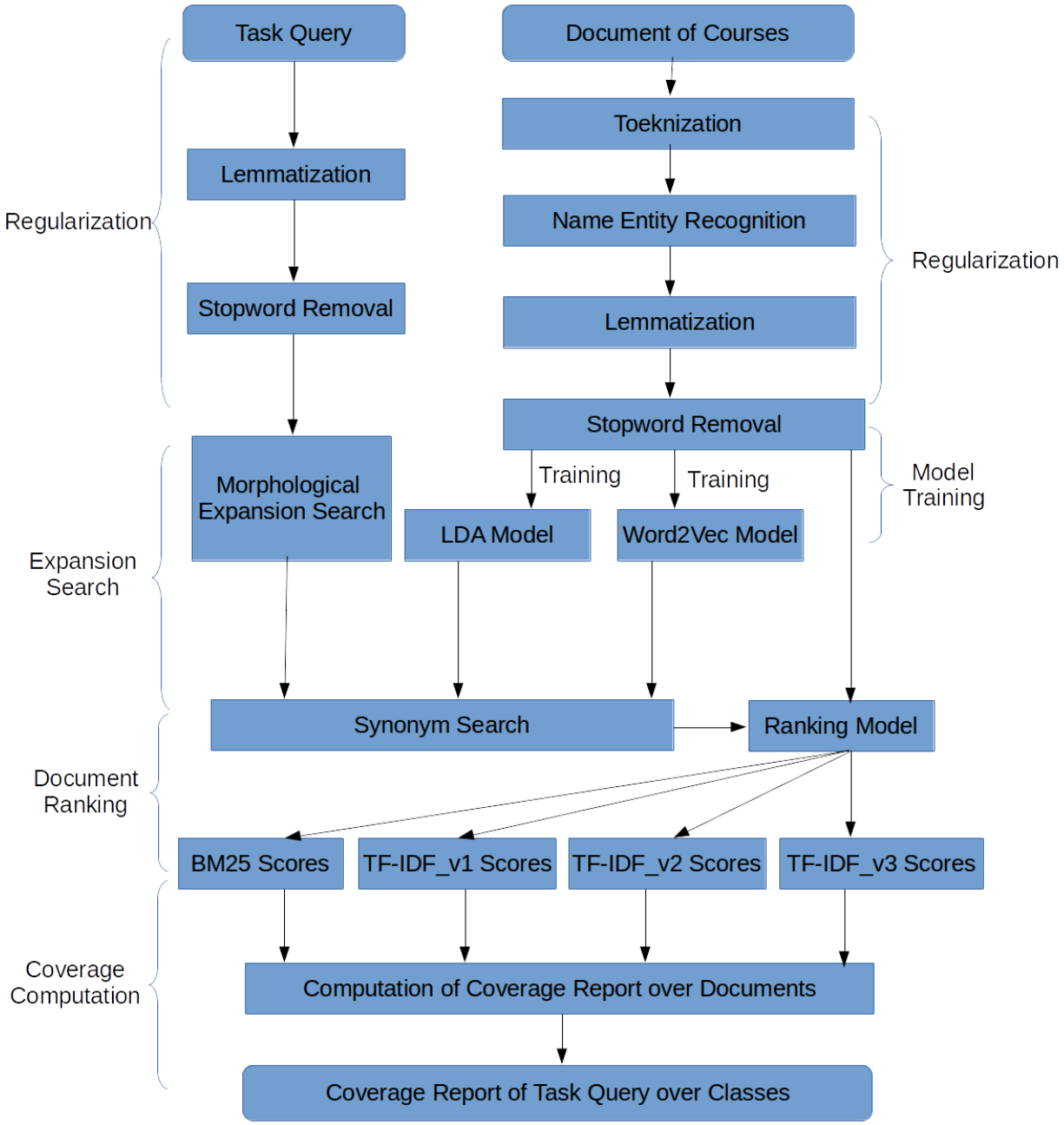

Figure 1. Workflow of coverage retrieval system

1. Text extraction

The first step of analyzing course coverage is to extract text from different types of files, including PDF, PPTX, DOCX, and text files, that contain the course materials. For PDF, we used the command line program `pdftotext` on Linux. For PPTX and DOCX, we used Python's module `pptx` and `docx`. For DOC, we used the command line program `antiword` on Linux. We assume that all documents are in UTF-8 encoding.

2. Lexical and syntactical analysis (Regularization)

Once text is extracted from a file, the next step is to “clean up” the text, such as rejecting whitespace characters from the text, recognizing phrases that cannot be separated, etc.

2.1 Tokenization

Tokenization is the step in NLP that slices the text into small units, usually words or phrases. Due to the variety of file formats, our tokenization is relatively exhaustive but thorough. The steps are given below.

1. Reject unrecognized characters and standardize non-alphanumeric character. All regular expression used in this task is shown below.

Regular Expression	Before	After
`ur'[\{\}\(\)\[\]|\=\+\`<>]+|_{2,}|\-{2,}|\.{2,}'	'{plotting}', '[was]',	' plotting ‘, ' was ‘
ur" [^A-Za-z0-9]{2,} “	‘a1’	‘ ’
ur"\([a-zA-Z]\)| s |'s “	"What's”	"What”
ur'\- | \-’	‘sub- section’	‘sub-section’
ur' [a-zA-Z][0-9]\. |\#[0-9]’	‘a1.1’, ‘#1’	‘ ’, ‘ ’
ur' /|/ '	'back/ forth’	'back/forth’
ur' (\\)|(\\) ‘	‘http:\\ www.google.com’	‘http:\\www.google.com’
ur"(?i)(n\/a)|(?i)(i\.e\.)|(?i)(cont'd)|(?i)(contd)”	" N/A i.e. cont'd Cont'd CONt'd contd Contd”	' ‘
ur"'|”|''|º”	"' ‘33º.16’00”N 32’30” 33'123" 325°07’00”W ‘"	‘” °33°.16'00"N 32'30" 33'123" 325°07'00"W ”’

2. Tag named entities that contains digits, such as “B747” which is short for “Boeing 747”. Denote the strings tagged as the set N. The table below indicates all regular expression we use to discover named entities.

Regular Expression	Example
ur'^[0-9][0-9.]+(\/[0-9]+)*[\.\,\;\:\'\"]*$'	'3.1415926' , '2.5/22/24/24'
ur'^([A-Za-z]+[0-9]+\/)*[A-Z]+[0-9]+[\.\,\;\:\'\"]*$\|^[ab]{1}[0-9]{3,}[\.\,\;\:\'\"]*$'	'A186 A186/A185/A184', 'usrQQQ666/EEW888F'
ur'^([0-9]+\/)+[0-9]+[\.\,\;\:\'\"]*$'	'09/28/2017'
ur'^([0-9]+[\-])+[0-9]+[\.\,\;\:\'\"]*$'	'999-999-9999'
ur'^[0-9]+(\/)[0-9A-Za-z]+[\.\,\;\:\'\"]*$'	'GSSO/CS115ISU'
ur'^[A-Za-z0-9]+\-[A-Za-z0-9]+\-[0-9]+[\.\,\;\:\'\"]*$'	'666-xxxx-666', 'WWE-WWW2'
ur'^([0-9]+\,)+[0-9]+[\.\,\;\:\'\"]*$'	'100,000'
ur'^[0-9]+[A-Z]+[\.\,\;\:\'\"]*$'	'666666Z'
ur'^[0-9]+_[A-Za-z]+_[0-9]+_[a-zA-Z0-9._]+[\.\,\;\:\'\"]*$'	'88888888_XX_88_XXXXXXXXXX_X'
ur'^([0-9]+\-){1,2}[0-9]*(\.)?[a-zA-Z]*[\.\,\;\:\'\"]*$'	'1111-1111-1111AB'
ur'^([0-9]+\:)?[0-9]+\:[0-9]+[a-zA-Z]{0,2}[\.\,\;\:\'\"]*$\|^[0-9]+\:[0-9]+[a-zA-Z]{0,2}\-[0-9]+\:[0-9]+[a-zA-Z]{0,2}[\.\,\;\:\'\"]*$'	'02:10:00am'
ur'^[0-9]+\.[0-9]+[a-zA-Z]*[\.\,\;\:\'\"]*$'	'88.88X'
ur'^[0-9]+[.,][a-zA-Z]+[\.\,\;\:\'\"]*$'	'2.xxx'
ur"^.*([0-9]+['])([0-9]+[\"]?)?[FWESN]?[\.\,\;\:\'\"]*$\|^.*([0-9]+['])?([0-9]+[\"])[FWESN]?[\.\,\;\:\'\"]?$"	'"33°.16'00"N"'
ur'^[a-zA-Z]+\-[0-9]+(s)?[\.\,\;\:\'\"]*$'	'xX-66 XX-66s'
ur'^[a-zA-Z]+(\/)[a-zA-Z]+[0-9]+[\.\,\;\:\'\"]*$'	'XX/XX66', 'XXXxx/XxX6666'
ur'^[A-Z]{1,8}[0-9]+[A-Z]{1,8}[\.\,\;\:\'\"]*$'	'AAAAAAAA88', 'AAA88'
ur'^[A-Za-z]+[0-9]*[\/\-][A-Za-z]*[0-9]+[A-Za-z]*[\.\,\;\:\'\"]*$'	'xx66-xx66', 'xx66-xx66xx'
ur'^[A-Z]+(\.)?[0-9]+[\.\-\/]?[0-9]+[a-z]?[\.\,\;\:\'\"]*$'	'xx66-66', 'xx-66', 'xx/66'
ur'^[A-Za-z]*[0-9]*_[0-9]*[A-Za-z]*[0-9]*[\.\,\;\:\'\"]*$'	'666_666', 'xxx666_666xxx', 'xxx666_xxx666', 'xxx666_666xxx666'
ur'^0x[ABCDEF0-9]+[\.\,\;\:\'\"]*$'	'0xFFFFF'
ur'^ILP.+|IPM.+'	'ILP15-xxxxx'
ur'^[0-9]+(\-)?[A-Z]+[0-9]+[\.\,\;\:\'\"]*$'	'"222-XXX222', '666XXX666"'
ur'^[0-9\.]+[%°Ω]?[\.\-]?[0-9\.]*[%°Ω][\.\,\;\:\'\"]*$'	'"33°', '33Ω', '33.33%"'
ur'.*[^a-zA-Z]{2,}[\-]?(?i)[(ft)(feet)(mile)(km)(Mbps)(Gbps)(kbps)(GHz)(nW\/cm2)(µV)(w/cm2)(V)(HZ)(Mb\/s)(kb\/s)(gb\/s)(msec/DIV)(sec/DIV)(dBm)(dB)	'20mile', '666w/cm2'

(inch)(m)(degree)(foot)(minute)(\'MSL)(footmarkings)(word)(MHz)(Mb)(kb)(gb)(ampere)(volt)]\W*$|'	
ur'^[0-9]+(\.)[0-9]+[\-]?[\.0-9]*(\/[A-Z]+)?[\.\,\;\:\'\"]*$'	'20.20-20', '20.20-20/AB'
ur'^([ABCDEF0-9]{2,4}[\-\.\:])*[ABCDEF0-9]{2,4}[\.\,\;\:\'\"]*$'	'AA-AA-AA-AA-AA'
ur'^[0-9]+(?i)[(jan)(feb)(mar)(apr)(may)(jun)(jul)(aug)(sept)(oct)(nov)(dec)(January)(February)(March)(April)(May)(June)(July)(August)(September)(October)(November)(December)][0-9]+\W*$|'	'28sept2017'
ur'^([0-9]+[\-\.\:][a-zA-Z]+){1,4}[\.\,\;\:\'\"]*$'	'2.xxx', '2-xxx', '2:xxx'

1. A string is separated into proto-tokens by whitespace characters (space, tab, newline, return, formfeed)
2. Then we turn proto-tokens into tokens.
    a. Each proto-token (e.g., "does1989mean") is separated into sub-proto-token (e.g., "does 1989 mean"), which can only be an alphabetical string, a string of digits which probably contains commas or periods (e.g., "4567," or "0567.6789,678."), or an ending of sentence punctuation (i.e., a period, a question mark, or a exclamation mark)
    b. For each sub-proto-token,
        i. If it is a string of digits which probably contains commas or periods (e.g., "4567," or "0567.6789,678."), or an ending of sentence punctuation, turn it into a token and add it into the set N.
        ii. Elseif it is a capitalized string with or without an ending "s" (e.g. "DNA" or "DNAs"), and its length is between 1 and 9 characters, turn the capitalized string into a token and add it into the set N.
        iii. Elseif it is in a dictionary (we used all words in Wikipedia and SCOWL [1]) and has a word frequency above 100, we just treat the sub-proto-token as a token.
        iv. For all other cases, very likely that insertion or deletion happened on at least one word in input document, e.g., "landing gear" becomes "landinggear" or "table" becomes 2 sub-proto-tokens "ta ble". To solve this problem, we perform word segmentation and spellcheck/spell correction.
            1. First, run the sub-proto-token through the Viterbi segmentation algorithm. If the sub-proto-token cannot be segmented, drop it. Otherwise, turn every segment has a frequency greater than 1000 into a temporary token list (empty at the beginning of segmentation and denoted as S).
            2. Then, run the sub-proto-token through a spellchecker [2] and return a word denoted as W.
            3. If the set S is not empty (e.g., at least one segment returned in Step 1.a.iv.1), make a judgement on using the result from segmentation or spellcheck. Given a dictionary, if the total frequency of all segments in the set S is higher than the frequency of the word W, use the segments in S as the tokens. Otherwise, the word W.
            4. Else (i.e. failed segmentation),
                a. If the sub-proto-token (or its lowercase) is in the dictionary, turn it (or its lowercase) into a token,
                b. Else (the sub-proto-token is not in the dictionary), form the token based on the context.

2.2 Named Entity Recognition (NER)

We perform NER in two steps. In the first step, if an element of set N and its context (50 words downward) are all not in the dictionary, we remove the element from the set N. In the second step, we use an existing NER tagger (from Stanford Core NLP NER tagger [3][13]) to further identity the remaining of set N. Denote the set of all NERs as set NE. For all tokens in the difference between N and NE (i.e., $N \backslash NE$), we lower their cases. Examples are in the table below:

a list of tokens before NER	a list of tokens after NER
[u'this', u'plotting', u'was', u'published', u'by', u'the', u'poet', u'Nekrassov']	[u'this', u'plotting', u'be', u'publish', u'by', u'the', u'poet', u'Nekrassov']
['While', 'in', 'France", ‘,', 'Christine', 'Lagarde', 'discussed', 'short-term', 'stimulus', 'efforts', 'in', 'a', 'recent', 'interview', 'with', 'the', 'Wall', 'Street"Journal', '.']	[('While', 'O'), ('in', 'O'), ('France', 'LOCATION'), (',', 'O'), ('Christine', 'PERSON'), ('Lagarde', 'PERSON'), ('discussed', 'O'), ('short-term', 'O'), ('stimulus', 'O'), ('efforts', 'O'), ('in', 'O'), ('a', 'O'), ('recent', 'O'), ('interview', 'O'), ('with', 'O'), ('the', 'O'), ('Wall', 'ORGANIZATION'), ('Street', 'ORGANIZATION'), ('Journal', 'ORGANIZATION'), ('.', 'O')]

2.3 Lemmatization

Because the English language is an inflexional language, meaning that the form of a word can be alternated based on gender, plurality, person, etc., we need to store every token into its canonical form (e.g., "children" to "child" or "took" to "take"). This step is known as lemmatization in NLP. We do it in two steps. In step 1, POS (part-of-speech) tags are applied to each token using NLTK's POS Tagger [4]. In step 2, the POS tags assist the NLTK's WordNet lemmatizer [4] to lemmatize tokens.

a list of tokens before Lemmatization	a list of tokens after Lemmatization
[u'this', u'plotting', u'was', u'published', u'by', u'the', u'poet', u'Nekrassov']	[u'this', u'plotting', u'be', u'publish', u'by', u'the', u'poet', u'Nekrassov']
[u'my', u'laptop', u'was', u’taken’, u’away’, u’by’, ‘someone’]	[u'my', u'laptop', u'be', u’take’, u’away’, u’by’, ‘someone’]

2.4 Stopword removal

In NLP, a word that is too frequent (e.g., "the", "there") or too rare are considered as the noise, denoted as stopwords. We use two sets of stop words. One is the NLTK's stopwords and the other is a Python module known as **stop_words** [5].

3. Search Expansion

Due to the synonyms, searching using the exact words in a task is too limited. For example, ... Therefore, we perform search expansion to prevent missing related documents. We expand the search query by leveraging the following three techniques jointly: expand various morphological forms of nouns in search query, find synonyms of each word in search query, estimating and associating the weight to words in expanded search query including their synonyms.

Two NLP approaches, word embedding and latent Dirichlet allocation (LDA), are employed to find synonyms. Suppose we a have a list $\mathbf{Q} = [q_1, \cdots, q_l]$, where q_i is a string of the ith task query. For each $q_i \in \mathbf{Q}$, we process it in the following order: regularize the query, perform the morphological expansion, search in synonyms and output the expanded search query $\mathbf{q}_i' = [o_1, \cdots, o_h]$ where each tuple $o_j = (q_j', b_j)$. q_j' is the jth word of $\mathbf{q}_i'$ and b_j is the weight of q_j'.

3.1. Regularization of query

Before we use the query to search information, we have to regularize the query. The regularization of query transforms a string of query to a list of words.

a. A string q_i is divided into a list of words by whitespace characters (space, tab, newline, return, formfeed)
b. do lemmatization of the list of words
c. Remove STOPWORD from the list of words

3.2. Morphological expansion search

Once the regularization of query is done, we get a list of words with their POS tag. We then do the morphological expansion search. Morphological expansion search is a common method to expand the search query. And we perform the following steps:

a. Let A be an empty dictionary
b. For each word of search query:
    i. Add the word to A and set its weight value to 1.0
    ii. If the tag of the word is NN or NNP:
        1. If it is a capitalized string:
            a. Add the lowercase form of the word in to A and set its weight value to 0.5
            b. Add a copy of the word where the first character is capitalized to A and set its weight value to 0.5
        2. Else (Not all characters of the word is lowercase letter)
            a. Add the lowercase form of the word to A and set its weight value to 0.9
            b. Add a copy of the word where the first character is capitalized to A and set its weight value to 0.9
    iii. Else (the tag of the word is NNS or NNPS):
        1. Add a copy of the lowercase form of the word where the last character is removed to A and set its weight value to 0.5
        2. Add a copy of the word where the last character is removed and the first character is capitalized to A and set its weight value to 0.5
        3. If the last character of the word is letter s or S:

   a. If the word is a capitalized string:
      i. Add a copy of the word where the last character is removed to A and set its weight value to 1.0
   b. for all other cases:
      i. Add a copy of the lowercase form of the word to A and set its weight value to 1.0
4. For all other cases:
   a. Add a copy of the lowercase form of the word to A and set its weight value to 0.9

c. Convert $A = \{p_1'' : b_1, \cdots, p_r'' : b_r\}$ to a list of tuple $X = [x_1, \cdots, x_r]$ where each tuple $x_i = (p_i'', b_i)$ for the query token y_i and its weight value b_i

For example, “VSRP” is the abbreviation of “Voluntary Safety Reporting Program”. In the tokenization of task description, if we meet a term VRSPs, which is the plural form of VRSP, the term will not be recovered to its singular form due to the problem of lemmatizer. So in this step, we manually operate the plural form of these terms by referring the POS tag of each token. The task description is a complete sentence such that we can use parsing tree to get the POS tag of each token. And the POS TAG can tell us whether it is a plural form or not. The token VRSPs will be recognized as a plural form and we expand the task token VRSPs to two tokens VRSP, VRSPs, Vrsp, Vrsps, vrsp, vrsps. The last four new tokens contains two diverse-case form of token VRSP and VRSPs. They are titled-case form and lower-case form. All these new expanded term will be assign with different weight. Singular form and plural form have a weight of 1. Titled-case form and lower-case form have a weight of 0.5 since they are dramatically different from the original form.

3.3. Synonym search

The method mentioned in Section 3.2 outputs a list of tuples $X = [x_1, \cdots, x_p]$ where each tuple $x_i = (y_i, c_i)$ for the query token y_i and its weight c_i obtained above. Synonyms to each token in the query can be found by leveraging word embedding and topic modeling. Here we use Word2vec as the approach for word embedding and LDA as the approach for topic modeling.

3.3.1. Word2vec

3.3.1.1. Word embedding

Unlike image or speech data, text information is very sparse. Correlation on spelling does not definitely mean similarity in meaning, e.g., “cat” vs. “hat”, while difference on spelling does not necessarily mean irrelevance in meaning, e.g., “controversial” vs. “questionable”. In order to find correlated or relevant words, NLP researchers usually map words into a high-dimensional space. There, the relationship between words can be easily measured using cosine similarity. Fig.2 is a example showing the the three words in 2-dimensional vector space. The angle between two word vectors determines the cosine similarity between two words. It is clear that “apple” is close to “banana” but “boat” is far away from them.

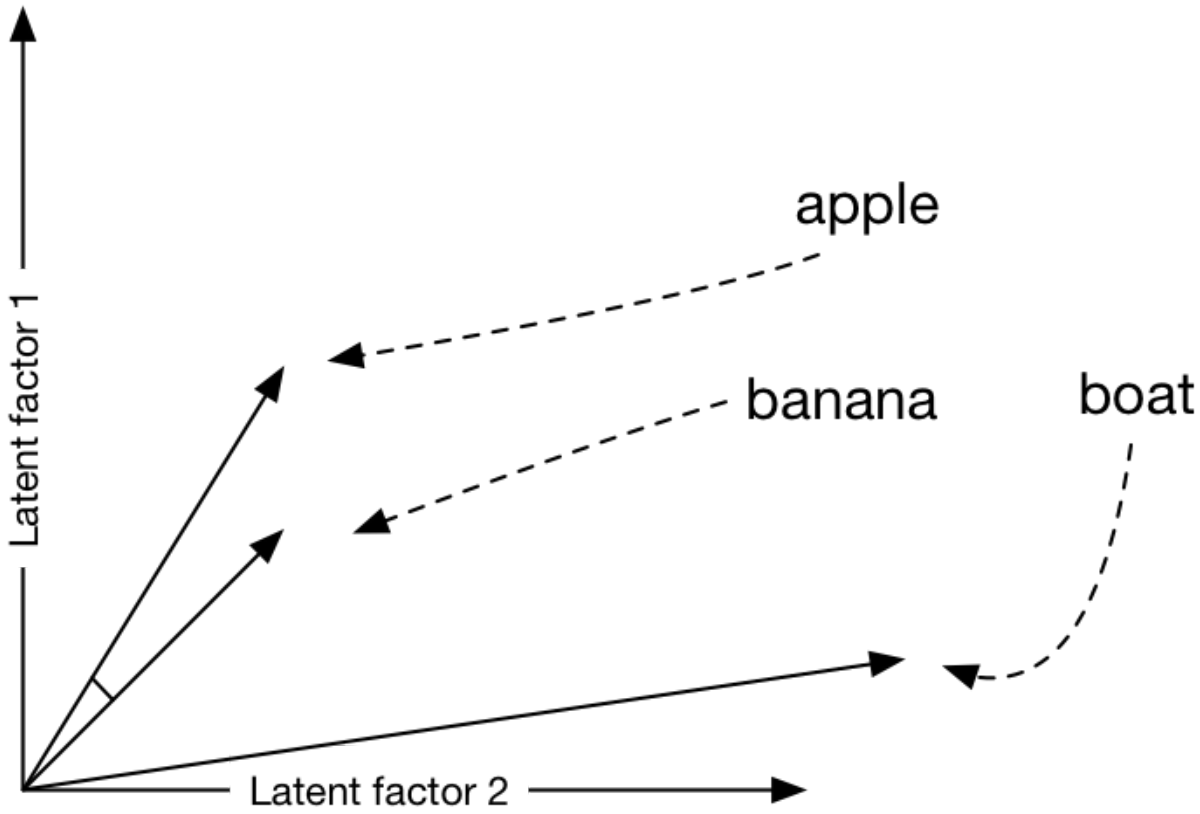

Figure 2. Visualization for word2vec cosine similarity [12]

The most successful word embedding is achieved using a model known as Word2vec, meaning word to vector [11]. The Word2vec model transforms a word to a fixed-length vector. In our system, the model is generated by training a skip-gram neural network model. The Word2vec model actually does not use the neural network but the weights of hidden layer. The task behind the skip-gram neural network is that given a input word in the middle of a sentence, pick one nearby word at random [6]. The network is supposed to tell us the probability of every word in corpus vocabulary being the nearby word of the input word. For example, Fig.3 shows the training samples of a sentence “The quick brown fox jumps over the lazy dog”. Here, we define nearby words is the left two words and the right two words of the input word in a sentence.

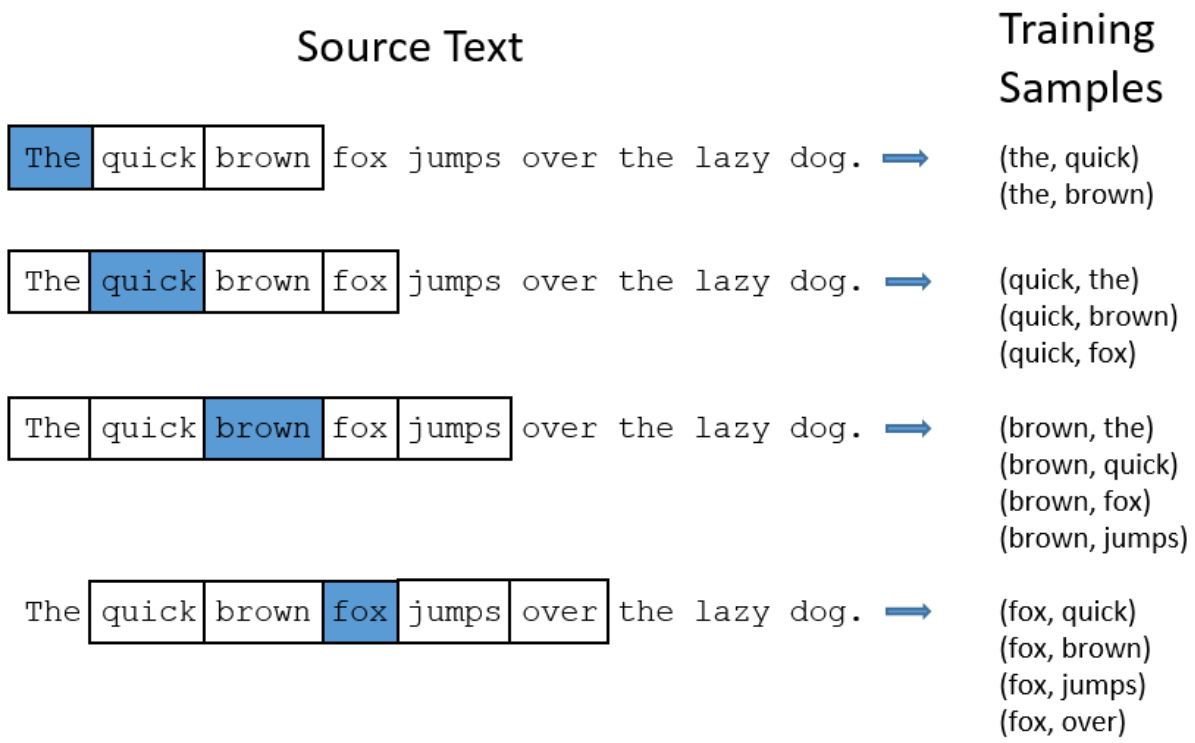

Figure 3. Training samples of skip-gram model [6]

Each training sample is shown as a pair of words. The greater the occurrence number of a pair of words is, the more likely to be the nearby word of each other they are. For instance, if the input word is “fox”, a well-trained network is possibly to output a higher probability of word “Brown” and “jumps” than it will for “Sunday”, which is irrelevant to “Brown” and “jumps”.

Fig.4 shows the structure of skip-gram neural network. The hidden layer does not have activation function and the output layer uses softmax function.

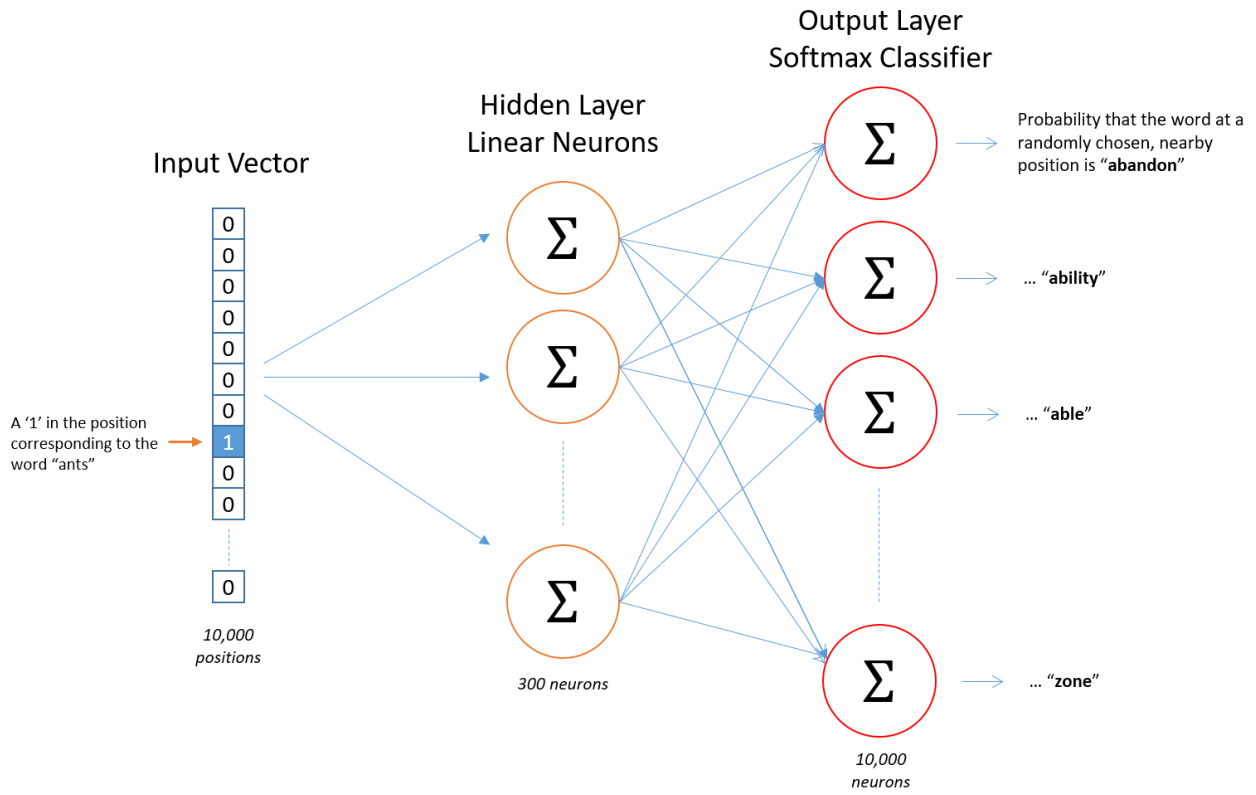

Figure 4. Architecture of skip-gram neural network [6]

Once the network above is trained, we use the weight matrix of hidden layer as the vector table for all words in the vocabulary of corpus. For example, the above figure shows that the weight matrix of hidden layer is a 10000x300 matrix because hidden layer has 300 neurons and the output layer has 10000 neurons. The vector of each word can be used to describe the relationship between similar words. For example, similar words like "Paris", "London", "Rome" will be placed close to each other in vector space. What's more, they will have a similar distance to the countries where they are. For instance, ''Rome" - "Italy" = "Paris" - "France" [7].

We train an word2vec model using the text from all course materials by using Python module Gensim [8]. The training parameters of this model is shown below

Parameter	Value
Vector Size (dimensionality of the feature vectors)	300
Window Size (maximum distance between the current and predicted word within a sentence)	5
Min_count (ignore all words with total frequency lower than this)	3
Negative Sampling (how many "noise words" should be drawn)	5
iteration (number of iterations over corpus)	40
Algorithm	Skip-gram

Alpha (initial learning rate)	0.05

3.3.1.2. Using Word2Vec in expanded search

We use Word2vec model to find synonym words and set their weight in four steps:

1. After we input a word of query to Word2vec model, it output a list of synonyms with their similarities between them and the input word.
2. Sort the list of synonyms by a descending order of their similarities.
3. Find the argument of the minimum of the first derivative of synonyms' similarities.
4. Output all word before the augument with their similarities.

It should be noted that similarities of synonyms range from 0 to 1 such that we consider them as their weights directly. The algorithm pseudo code is shown below.

Input: A set Q of m words representing all words in training corpus, a list of tuple $\mathbf{X} = [x_1, \cdots, x_p]$ where each tuple $x_i = (y_i, c_i)$ for a query token y_i and its weight c_i obtained above in Section 3.2 representing the query, and a word2vec model $f : Q \mapsto \mathbb{R}^n$.
Output: A list of tuples L

- Step 1. Let L be an empty list.
- Step 2. For each element (y_i, c_i) in $\mathbf{X}$:
  - Step 2.1. Compute a list of tuples $\mathbf{S} = [(w_1, s_1), \cdots, (w_m, s_m)]$ such that $s_i = \frac{\mathbf{v_0} \cdot \mathbf{v_i}}{\|\mathbf{v_0}\| \|\mathbf{v_i}\|}$, where $\mathbf{v_0} = f(y_i)$ and $\mathbf{v_i} = f(w_i)$, for all $w_i \in Q \backslash \{y_i\}$.
  - Step 2.2. Sort the list $\mathbf{S}$ based on the first element of each tuple. Denote the result as $\mathbf{S'} = [(w'_1, s'_1), \cdots, (w'_m, s'_m)]$ such that $s'_i \geq s'_j,\ i < j, \forall i, j \in [1..m]$.
  - Step 2.3. Let $g = \text{argmin}([s'_1, \cdots, s'_m])$.
  - Step 2.4. Create a list $l = [(w'_1, c_i \times s'_1), \cdots, (w'_g, c_i \times s'_g)]$ and append l into L.

3.3.2 Latent Dirichlet Allocation (LDA)

3.3.2.1 Topic modeling

Topic modeling is based on the observation that words cluster according to the topics. For example, words like "spoon", "plate" and "appetite" appear together often when being mentioned in a discussion about food. The word "spoon" is unlikely to co-occur with words like "keyboard", "mouse", and "monitor", which are related to a discussion about computers. Such clustering of words is a good sign to find synonyms or related words at large. Each cluster of word is known as a topic in NLP. Finding the clustering and the chance that a word appears in a topic is known as topic modeling. In this project, we use the text from all course materials as the corpus to build a model of topics.

Latent Dirichlet Allocation (LDA) is so far the most successful way to automatically discover topics. It generates a word distribution over topics. For each word, we know the probability that the word occurs in certain topic. Inversely, for each topic, we can also obtain the topic distribution over words. Fig.5 below shows the top words of four topics. Words with same color are the top words of one topic. And we can see that the top words of each topic carry the information of their associated topics. For example, "music", "movie" and "opera" in the first topic column are all related to art. "school", "students" and "teachers" in the fourth column are all related to education.

"Arts"	"Budgets"	"Children"	"Education"
NEW	MILLION	CHILDREN	SCHOOL
FILM	TAX	WOMEN	STUDENTS
SHOW	PROGRAM	PEOPLE	SCHOOLS
MUSIC	BUDGET	CHILD	EDUCATION
MOVIE	BILLION	YEARS	TEACHERS
PLAY	FEDERAL	FAMILIES	HIGH
MUSICAL	YEAR	WORK	PUBLIC
BEST	SPENDING	PARENTS	TEACHER
ACTOR	NEW	SAYS	BENNETT
FIRST	STATE	FAMILY	MANIGAT
YORK	PLAN	WELFARE	NAMPHY
OPERA	MONEY	MEN	STATE
THEATER	PROGRAMS	PERCENT	PRESIDENT
ACTRESS	GOVERNMENT	CARE	ELEMENTARY
LOVE	CONGRESS	LIFE	HAITI

The William Randolph Hearst Foundation will give $1.25 million to Lincoln Center, Metropolitan Opera Co., New York Philharmonic and Juilliard School. "Our board felt that we had a real opportunity to make a mark on the future of the performing arts with these grants an act every bit as important as our traditional areas of support in health, medical research, education and the social services," Hearst Foundation President Randolph A. Hearst said Monday in announcing the grants. Lincoln Center's share will be $200,000 for its new building, which will house young artists and provide new public facilities. The Metropolitan Opera Co. and New York Philharmonic will receive $400,000 each. The Juilliard School, where music and the performing arts are taught, will get $250,000. The Hearst Foundation, a leading supporter of the Lincoln Center Consolidated Corporate Fund, will make its usual annual $100,000 donation, too.

Figure 5. An article from AP corpus with words of different topics color-coded [9]

Further, we can compute the document distribution over topic by accumulating the word distribution over topics. From the Fig.6, it is clear that words in yellow topic have higher probability than those in pink and blue ones, which means the article in the figure has a higher probability to talk about the yellow topic. Top words like "gene", "dna" and "genetic" in yellow topic are all related to gene.

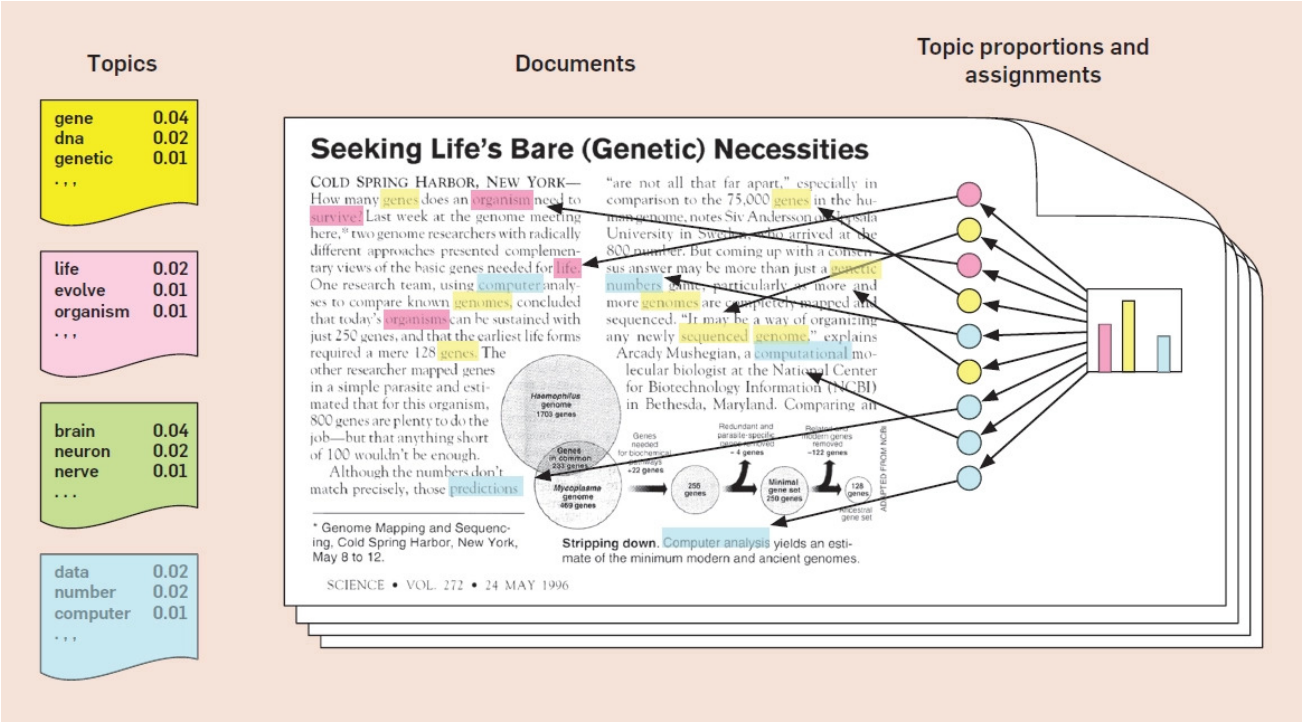

Figure 6. An article with different proportion of topics[10]

3.3.2.2. Using LDA in expanded search

We employ LDA for expanded search in the following steps:

a. After we input a word of query to LDA model, the model returns a list of topics with the probabilities that the input word occurs in each topic
b. Sort the list of topics by a descending order of their probabilities.
c. Compute the first derivative along the list.
d. Find the argument of the minimum of the first derivative, i.e., the vertex or tipping point.
e. For each topic before the argument
    i. Use LDA model to find a list of word with their probabilities that they occur in certain topic.
    ii. Sort the list of words by a descending order of their probabilities.
    iii. Find the argument of the minimum of the first derivative of words' probabilities
    iv. Output all word before the augument with their similarities.

It should be noted that probabilities of words occur in certain topic range from 0 to 1 such that we consider them as their weights directly.

4. Coverage Estimation

With the search result returned, we can calculate the coverage of a job task in a course. We leverage two methods: TF-IDF (Term frequency–Inverse document frequency) and Okapi BM25 (Best Match 25). These two methods are frequently used by search engines to rank search result.

Suppose that the input of coverage computation is a expanded search query $\mathbf{q}'_i = [o_1, \cdots, o_h]$ where each tuple $o_j = (q'_j, b_j), \forall i \in [1..h]$

4.1 TF-IDF

TF-IDF is computed as $\mathrm{tfidf}(q', d, D) = \mathrm{TF}(q', d) \cdot \mathrm{IDF}(q', D)$, where q' is the term of expanded result. Also, q' is in document d of document set D.

Term frequency, denoted as $f_{q',d}$, means the frequency that term q' occurs in document d. Following the convention that a term frequency is usually smoothed, we use the following two equations to compute the term frequency for ranking:

$\mathrm{TF1}(q', d) = \log(1 + f_{q',d})$

$\mathrm{TF2}(q', d) = 0.5 + 0.5 \cdot \frac{f_{q',d}}{\max\{f_{t,d}:t \in d\}}$
where t is a term of document d.

Similarly, inverse document frequency can be defined in various ways. We adopt the following two commonly used equations:
$\mathrm{IDF1}(q', D) = \log \frac{N}{|\{d \in D: q' \in d\}|}$

$\mathrm{IDF2}(q', D) = \log \frac{N - |\{d \in D: q' \in d\}| + 0.5}{|\{d \in D: q' \in d\}| + 0.5}$
where N is the total number of documents in the corpus and $|\{d \in D : q' \in d\}|$ is the number of documents where the word t occurs.

We compute three versions of TF-IDF score for expanded search result $\mathbf{q}'_i = [o_1, \cdots, o_h]$ where each tuple $o_j = (q'_j, b_j)$

TF-IDF_v1 score: $R_1(d, \mathbf{q}'_i) = \sum_{(q'_j, b_j) \in \mathbf{q}'_i} \mathrm{TF1}(q'_j, d) \cdot \mathrm{IDF1}(q'_j, D) \cdot b_j$
TF-IDF_v2 score: $R_2(d, \mathbf{q}'_i) = \sum_{(q'_j, b_j) \in \mathbf{q}'_i} \mathrm{TF1}(q'_j, d) \cdot \mathrm{IDF2}(q'_j, D) \cdot b_j$
TF-IDF_v3 score: $R_3(d, \mathbf{q}'_i) = \sum_{(q'_j, b_j) \in \mathbf{q}'_i} \mathrm{TF2}(q'_j, d) \cdot \mathrm{IDF2}(q'_j, D) \cdot b_j$

4.2 BM25

However, TF-IDF does not perform well for short document. BM25 (Best Matching 25) is more suitable for short document. BM25 rank score is calculated as

BM25 score: $R_4(d, \mathbf{q}'_i) = \sum_{(q'_j, b_j) \in \mathbf{q}'_i} \mathrm{IDF2}(q'_j, D) \cdot \frac{f_{q'_j,d} \cdot (k_1 + 1)}{f_{q'_j,d} + k_1 \cdot \left(1 - v + v \cdot \frac{|d|}{\mathrm{avgdl}}\right)} \cdot b_j$
where $f_{q'_j,d}$ is word frequency of q'_j in the document d of document set D, $|d|$ is the length of the document d in words, and avgdl is the average document length in the text collection from which documents are drawn. k_1 and v are free parameters.

Finally, we obtain a matrix of coverage score for every document and task. Since every document belongs to a specific class, we then compute the score for each class by comparing the coverage score of all document under the same class directory. If a class has multiple documents, the maximum score of all documents of the class is treated as the coverage score of the class.

5. Coverage Report

For each task, we compute a score based on multiple metrics, such as BM25 and TF-IDF. For each metrics, the top 10 classes that best cover the task i will be listed, denoted as $\mathrm{C}_i^k = [c_{i,1}^k, \cdots, c_{i,10}^k]$ where the class $c_{i,j}^k$ is the j-th class most relevant to the task query i by using metric k. The coverage score of the task is then defined as the geometric average of all C_i^k from different metrics. Hence, it's the average of 40 numbers. Let $\mathbf{U} = [\mathbf{u}_1, \cdots, \mathbf{u}_z]$ be a list of class sets where $\mathbf{u}_j$ is a set of class document $\forall j \in [1..z]$.

To get the list C_i^k, the coverage function $R'_k(R'_1(\mathbf{u}_j, \mathbf{q}'_i))$ is used to compute the coverage score that jth class is relevant to the task query i by using metric k, $\forall i \in [1..h], \forall j \in [1..z], \forall k \in [1..4]$
$R'_k(\mathbf{u}_j, \mathbf{q}'_i) = \max\{R_k(d, \mathbf{q}'_i) : d \in \mathbf{u}_j\}$, where $R_k(d, \mathbf{q}'_i)$ is defined in Section 4, $\forall k \in [1..4]$.

The coverage score of jth class is calculated by using the coverage function above
Score of Metric TF-IDF_v1: $R'_1(\mathbf{u}_j, \mathbf{q}'_i) = \max\{R_1(d, \mathbf{q}'_i) : d \in \mathbf{u}_j\}, \forall i \in [1..h], \forall j \in [1..z]$

Score of Metric TF-IDF_v2: $R'_2(\mathbf{u}_j, \mathbf{q}'_i) = \max\{R_2(d, \mathbf{q}'_i) : d \in \mathbf{u}_j\}, \forall i \in [1..h], \forall j \in [1..z]$

Score of Metric TF-IDF_v3: $R'_3(\mathbf{u}_j, \mathbf{q}'_i) = \max\{R_3(d, \mathbf{q}'_i) : d \in \mathbf{u}_j\}, \forall i \in [1..h], \forall j \in [1..z]$

Score of Metric BM25: $R'_4(\mathbf{u}_j, \mathbf{q}'_i) = \max\{R_4(d, \mathbf{q}'_i) : d \in \mathbf{u}_j\}, \forall i \in [1..h], \forall j \in [1..z]$

Now the coverage score list $\mathrm{C}_i^{'k} = [R'_k(\mathbf{u}_1, \mathbf{q}'_i), \cdots, R'_k(\mathbf{u}_z, \mathbf{q}'_i)]$
Then we sort the coverage score list $\mathrm{C}_i^{'k}$ by a descending order of coverage score and then save the top 10 coverage scores of class for task query i. The result is
$\mathrm{C}_i^k = \mathrm{sort}(\mathrm{C}_i^{'k}, \mathrm{cmd} = reverse)[:10] = [c_{i,1}^k, \cdots, c_{i,10}^k], \forall k \in [1..4]$
The coverage score of task *i* is defined as
$R_t(\mathbf{q}'_i) = \sqrt[40]{\prod_{k=1}^{4}\prod_{j=1}^{10} c_{i,j}^k}$

6. Summary of Result Files

The result of curriculum coverage analysis is presented in Excel spreadsheets. There are 3 files:

1. `rawsearch.xlsx` contains the result without expanded search (neither Section 3.3.1 nor Section 3.3.2 were used)
2. `w2v.xlsx` contains the result with Word2vec based expanded search (Section 3.3.1 was used)
3. `w2vlda.xlsx` contains the result with Word2vec and LDA based expanded search (both Sections 3.3.1 and 3.3.2 were used)

Each file contains 5 sheets:

a. Task Coverage Score
b. Top 10 classes with highest BM25 score
c. Top 10 classes with highest TF-IDF_v1 score
d. Top 10 classes with highest TF-IDF_v2 score
e. Top 10 classes with highest TF-IDF_v3 score

In the last 4 sheets (BM25 and TF-IDF v1-v3), the results are presented in 20 columns (B to U), in which one class corresponds to a pair of columns, one for class ID and the other for the score.

For example, screenshots of the second file are shown in Fig.7 and Fig.8. The first column in Fig.7 is the task description. The second column and third column are the class 1 No. and its coverage score respectively. The following two columns are the class 2 No. and its coverage score respectively, and so on and so forth. The column setting in Fig.8, Fig.9 and Fig.10 are same to that of Fig. 7

	A	B	C	D	E	F	G	H	I	J	K
1	**Task Description**	**Class 1**	**Score**	**Class 2**	**Score**	**Class 3**	**Score**	**Class 4**	**Score**	**Class 5**	**Score**
2	Support a safety culture	50314001	20.84	45519001	20.52	60005177	18.32	40527001	18.25	60005181	18.2
3	Adhere to the elements of a safety culture	60004563	14.44	50314001	14.19	60004710	14.09	60004817	14.09	45519001	13.89
4	Define an informed culture	50314001	27.89	45519001	26.61	60004563	19.63	55047	17.47	40527001	17.26
5	Define a reporting culture	50314001	24.01	45519001	23.78	60004563	13.71	55088001	13.59	55047	13.4
6	Define a learning culture	50314001	25.01	45519001	24.51	50118	16.63	60004563	15.16	55047	14.82
7	Define a just culture	50314001	18.98	45519001	18.66	55088001	13.6	50118	12.99	55085	10.61
8	Define a flexible culture	45519001	26.82	50314001	25.64	60004563	17.7	55047	17.35	50148001	16.14
9	Adhere to Voluntary Safety Reporting Program (VSR	60004561	72.19	50314001	70.28	50310	65.35	60004995	64.99	50046	59.88
10	Define Voluntary Safety Reporting Program (VSRP)/	50314001	81.42	60004561	73.84	50310	67.15	60004995	66.68	50046	61.07
11	Identify VSRP/ATSAP purpose	50314001	21.78	50331001	16.12	60004821	16.09	55049001	15.77	60005161	14.94
12	Identify VSRP/ATSAP benefits	50314001	26.86	50331001	20.01	55049001	20.01	60004561	17.25	60005161	17.17
13	Identify VSRP/ATSAP content	50314001	21.78	50331001	17.96	60004821	17.91	55049001	17.58	60004561	16.54
14	Enter VSRP/ATSAP report information	55085	18.71	57079	17.41	55088001	17.05	55093	16.86	60005161	16.81
15	Establish teamwork and professional standards	55047	20.7	60004563	19.96	50148001	17.82	50331001	17.29	50085001	15.97
16	Support a teamwork environment	55047	14.03	50043	13.83	50046	13.77	60004563	13.64	50148001	13.49
17	Participate in training and other professional develop	40527001	21.58	45519001	21.53	55047	21.48	60004704	21.41	60004820	21.02
18	Maintain facility awareness	60004807	24.9	60004809	24.9	60005177	24.85	60005181	24.85	60005182	24.85
19	Inform the supervisor of important teamwork situatio	60004744	25.93	55047	22.52	60004563	22.1	50043	20.1	50046	20.07
20	Apply professional standards	50034	21.12	50148001	17.76	43074001	15.8	50046	14.33	45468001	14.3
21	Assess personal workload	50314001	23.35	50148001	19.52	45465001	16.93	60004704	15.35	60005177	15.33

Figure 7. BM25 Coverage Score of top 4 classes over 20 tasks with expanded search of Word2vec

	A	B	C	D	E	F	G	H	I	J	K
1	**Task Description**	**Class 1**	**Score**	**Class 2**	**Score**	**Class 3**	**Score**	**Class 4**	**Score**	**Class 5**	**Score**
2	Support a safety culture	45519001	60.64	50314001	59.01	60004563	56.2	60004820	46.31	60005181	45.98
3	Adhere to the elements of a safety culture	60004563	65.63	50314001	45.06	55047	41.37	45519001	40.88	60004710	37.94
4	Define an informed culture	60004563	63.67	50314001	41.63	55047	39.45	45519001	38.95	40527001	31.92
5	Define a reporting culture	60004563	51.67	45519001	43.88	50314001	39.98	55047	37.33	60004995	28.76
6	Define a learning culture	60004563	55.81	55047	39.51	50314001	35.71	45519001	29.39	50331001	29.03
7	Define a just culture	60004563	37.82	55047	29.19	50314001	26.6	40527001	22.84	60004995	22.84
8	Define a flexible culture	60004563	61.74	55047	45.28	50314001	33.03	45519001	32.28	50148001	27.18
9	Adhere to Voluntary Safety Reporting Program (VSR	60004561	176.74	60005177	173.99	60005181	173.21	60004704	169.65	50314001	169.4
10	Define Voluntary Safety Reporting Program (VSRP)/	60005177	189.01	60005181	188.36	60004704	185.15	50314001	183.56	60004561	180.07
11	Identify VSRP/ATSAP purpose	60004561	42.23	50314001	41.16	50043	40.8	60005161	40.3	50046	30.55
12	Identify VSRP/ATSAP benefits	50314001	45.89	60004561	39.1	50043	38.17	60005161	36.16	55049001	30.6
13	Identify VSRP/ATSAP content	60004561	50.49	50314001	46.89	50043	44.6	60005161	32.77	60004995	32.57
14	Enter VSRP/ATSAP report information	60005161	43.93	60004561	40.71	50314001	37.7	60004820	35.95	60005181	35.86
15	Establish teamwork and professional standards	55047	49.48	60004563	35.53	40527001	33.15	50046	29.59	50043	29.25
16	Support a teamwork environment	55047	35.52	45519001	31.85	60005181	30.52	60004820	30.52	50043	30.11
17	Participate in training and other professional develop	60004704	65.33	60004820	64.4	60005181	64.03	45519001	62.75	60005177	61.05
18	Maintain facility awareness	60005177	66.49	60004807	66.39	60004809	66.39	60005182	65.45	60005181	65.45
19	Inform the supervisor of important teamwork situatio	55047	64.34	60004563	53.57	60004744	41.98	60004820	41.85	60004704	41.82
20	Apply professional standards	50148001	56.01	40527001	43.81	50034	36.47	45519001	35.02	60005181	33.94
21	Assess personal workload	60004704	41.14	60005177	40.73	60005181	39.62	45519001	38.39	60004820	31.99

Figure 8. TF-IDF_v1 Coverage Score of top 4 classes over 20 tasks with expanded search of Word2vec

	A	B	C	D	E	F	G	H	I	J	K
1	**Task Description**	**Class 1**	**Score**	**Class 2**	**Score**	**Class 3**	**Score**	**Class 4**	**Score**	**Class 5**	**Score**
2	Support a safety culture	50314001	8.18	45519001	6.91	40527001	6.88	60004563	6.83	55047	6.7
3	Adhere to the elements of a safety culture	60004563	8.07	55047	7.73	50148001	6.33	60004817	6.22	60004710	6.22
4	Define an informed culture	50314001	10.1	45519001	9.95	55047	8.71	60004563	7.89	40527001	6.3
5	Define a reporting culture	50314001	8.61	45519001	8.5	55047	6.79	60004563	5.86	60004463	5.03
6	Define a learning culture	50314001	8.91	45519001	8.75	55047	7.25	60004563	6.3	50118	6.29
7	Define a just culture	50314001	6.91	45519001	6.78	55047	5.05	60004563	5.05	55049001	4.47
8	Define a flexible culture	50314001	9.6	45519001	9.47	55047	8.11	60004563	7.06	50148001	6.29
9	Adhere to Voluntary Safety Reporting Program (VSR	60004561	28.52	50314001	27.54	50310	25.21	50046	24.72	60004995	24.11
10	Define Voluntary Safety Reporting Program (VSRP)/	50314001	30.32	60004561	29.31	50310	26.04	50046	25.52	60004995	24.98
11	Identify VSRP/ATSAP purpose	50314001	8.28	60004821	6.85	55049001	6.75	50331001	6.75	60005161	6.55
12	Identify VSRP/ATSAP benefits	50314001	10.15	55049001	7.8	50331001	7.75	60004561	6.86	60005161	6.64
13	Identify VSRP/ATSAP content	50314001	8.28	60004821	7.33	55049001	7.14	50331001	7.12	60004561	7.03
14	Enter VSRP/ATSAP report information	60004561	6.33	55147001	6.24	57079	6.18	55085	6.03	60004704	5.86
15	Establish teamwork and professional standards	50148001	7.26	12051	7.11	55047	6.65	60004563	6.59	50331001	6.5
16	Support a teamwork environment	50148001	5.07	50046	4.93	50043	4.81	55047	4.69	50331001	4.52
17	Participate in training and other professional develop	55049001	8.06	60004995	7.98	50331001	7.83	50034	7.79	50085001	7.78
18	Maintain facility awareness	60004704	9.63	60004820	9.56	60005181	9.54	60005177	9.09	60005161	9.09
19	Inform the supervisor of important teamwork situatio	60004744	9.06	60004563	7.54	55047	7.52	40527001	7.43	55090	7.43
20	Apply professional standards	50034	8.37	50148001	6.99	50046	6.26	43074001	6.03	45468001	5.89
21	Assess personal workload	50314001	9.51	50148001	8.17	45465001	6.19	60004705	5.27	60004490	5.02

Figure 9. TF-IDF_v2 Coverage Score of top 4 classes over 20 tasks with expanded search of Word2vec

	A	B	C	D	E	F	G	H	I	J	K
1	**Task Description**	**Class 1**	**Score**	**Class 2**	**Score**	**Class 3**	**Score**	**Class 4**	**Score**	**Class 5**	**Score**
2	Support a safety culture	50314001	5.22	40527001	4.45	45519001	4.44	55047	4.36	60004563	4.34
3	Adhere to the elements of a safety culture	60004563	5.17	55047	4.95	50148001	3.99	60004710	3.91	60004817	3.91
4	Define an informed culture	50314001	6.8	45519001	6.7	55047	5.84	60004563	5.29	40527001	4.19
5	Define a reporting culture	50314001	5.86	45519001	5.79	55047	4.6	60004563	3.97	60004463	3.4
6	Define a learning culture	50314001	6.1	45519001	5.99	55047	4.95	50118	4.31	60004563	4.3
7	Define a just culture	50314001	4.75	45519001	4.67	55047	3.47	60004563	3.47	55049001	3.08
8	Define a flexible culture	50314001	6.61	45519001	6.51	55047	5.57	60004563	4.85	50148001	4.32
9	Adhere to Voluntary Safety Reporting Program (VSR	60004561	18.11	50314001	17.43	50310	15.83	50046	15.52	60004995	15.04
10	Define Voluntary Safety Reporting Program (VSRP)/	50314001	19.13	60004561	18.5	50310	16.24	50046	15.92	60004995	15.48
11	Identify VSRP/ATSAP purpose	50314001	5.38	60004821	4.13	55049001	4.07	50331001	4.06	60005161	4.03
12	Identify VSRP/ATSAP benefits	50314001	6.6	55049001	4.99	50331001	4.95	60004561	4.69	60005161	4.05
13	Identify VSRP/ATSAP content	50314001	5.38	60004821	4.58	50331001	4.47	60004561	4.47	55049001	4.46
14	Enter VSRP/ATSAP report information	55085	4.52	60004561	4.48	57010001	4.46	60004704	4.4	57079	4.4
15	Establish teamwork and professional standards	50148001	4.81	12051	4.59	55047	4.39	60004563	4.35	50331001	4.29
16	Support a teamwork environment	50148001	3.38	50046	3.28	50043	3.2	55047	3.12	50331001	2.99
17	Participate in training and other professional develop	60004995	4.85	50034	4.8	55049001	4.79	50085001	4.77	50331001	4.74
18	Maintain facility awareness	60004704	5.91	60004820	5.89	60005181	5.89	60005177	5.76	60005161	5.76
19	Inform the supervisor of important teamwork situatio	60004744	5.93	55047	4.89	60004563	4.88	40527001	4.88	55090	4.87
20	Apply professional standards	50034	5.39	50148001	4.3	50046	3.91	43074001	3.79	45468001	3.72
21	Assess personal workload	50314001	6.47	50148001	5.58	45465001	4.21	60004705	3.52	60004490	3.36

Figure 10. TF-IDF_v3 Coverage Score of top 4 classes over 20 tasks with expanded search of Word2vec

Fig.11 shows the final coverage score of 20 tasks (the first sheet of each spreadsheet file). the first column is task description and the second is the task coverage score.

1	Task Description	Task Coverage Score
2	Support a safety culture	12.73
3	Adhere to the elements of a safety culture	10.88
4	Define an informed culture	12.16
5	Define a reporting culture	10.05
6	Define a learning culture	10.79
7	Define a just culture	8.14
8	Define a flexible culture	11.22
9	Adhere to Voluntary Safety Reporting Program (VSR	41.53
10	Define Voluntary Safety Reporting Program (VSRP)/	43.69
11	Identify VSRP/ATSAP purpose	10.33
12	Identify VSRP/ATSAP benefits	11.36
13	Identify VSRP/ATSAP content	11.18
14	Enter VSRP/ATSAP report information	11.15
15	Establish teamwork and professional standards	9.82
16	Support a teamwork environment	8.09
17	Participate in training and other professional develop	14.55
18	Maintain facility awareness	17.14
19	Inform the supervisor of important teamwork situatio	13.13
20	Apply professional standards	10
21	Assess personal workload	10.4

Figure 11. Coverage Score of 20 tasks with expanded search of Word2vec

References:

1. Atkinson, K., “Spell Checker Oriented Word Lists (SCOWL)”, http://wordlist.aspell.net/ (2016).
2. Norvig, P., "How to write a spelling corrector", *http://norvig.com/spell-correct.html* (2007).
3. Manning, C. D., Mihai S., Bauer J., Finkel, J. R., Bethard, S., and McClosky, D., "The stanford CoreNLP natural language processing toolkit", *In ACL (System Demonstrations)*, pp. 55-60, (2014).
4. Bird, S., "NLTK: the natural language toolkit", *In Proceedings of the COLING/ACL on Interactive presentation sessions*, pp. 69-72. Association for Computational Linguistics, (2006).
5. stop-words 2015.2.23.1, https://pypi.python.org/pypi/stop-words (2015).
6. McCormick, C., “Word2Vec Tutorial-The Skip-Gram Model", http://mccormickml.com/2016/04/19/word2vec-tutorial-the-skip-gram-model/ (2016).
7. Nicholson, A. C., and Gibson, A., "Deeplearning4j: Open-source, Distributed Deep Learning for the JVM", https://deeplearning4j.org/ (2017).
8. Rehurek, R., and Sojka, P., "Software framework for topic modelling with large corpora", *In Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks,* (2010).
9. Blei, D., Ng, A., and Jordan, M., "Latent Dirichlet allocation", *In Journal of machine Learning research*, pp. 993-1022, (2003).
10. Blei, D., "Probabilistic topic models", In *Communications of the ACM* 55, no. 4: 77-84., (2012).
11. Mikolov, Y., Chen, K., Corrado, G., and Dean, J., "Efficient estimation of word representations in vector space", *In arXiv preprint arXiv:1301.3781* (2013).
12. Bernhardsson E., “Nearest neighbor methods and vector models – part 1”, https://erikbern.com/2015/09/24/nearest-neighbor-methods-vector-models-part-1.html (2015).
13. Finkel, J. R., Grenager, T., and Manning. C., “Incorporating Non-local Information into Information Extraction Systems by Gibbs Sampling”, *In Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics,* pp. 363-370., (2005).