Improving Search for ExploreCourses

Sam Redmond
Stanford University
Stanford, CA 94305
sredmond@stanford.edu

Eddie Wang
Stanford University
Stanford, CA 94305
eddiew@stanford.edu

1 Introduction

Currently, using Stanford's default course catalog, ExploreCourses, is a frustrating experience for users. The current search implementation on ExploreCourses generates course results by substring matching a user's query against course codes, titles, instructor names, and descriptions, and then sorts the results in alphabetical order by course code. While not terrible, this strategy often leads to situations where the most relevant courses are not on the first page of results, which is the source of many headaches come class registration time. Most significantly, the most relevant results are often not even on the first page. Students have to search several different variations of a single concept in order to find the best courses. Additionally, since ExploreCourses works by exact substring match, it can be sensitive to slight differences in search terms.

We chose to pursue this problem because improving search for ExploreCourses will benefit not just ourselves as students but also the whole of the Stanford community. The classes we take are integral to our intellectual vitality, so it is very important that students have the right tools in hand to find courses.

The input to our algorithm is a string representing a user's query (e.g. 'machine learning'). Using a composite model that we have built, we then generate an ordered list of courses with high relevance to the given search string (e.g. [CS229, . . .]).

2 Related Work

The usage of vectors to represent the meaning of words for the purpose of analysis has been studied extensively [1]-[3]. The current state-of-the-art model, word2vec, is described in [2]. Recently, an extension to word2vec to obtain vectors for sentences and paragraphs was proposed in [4]. This is done by treating each paragraph, or label, as a context word used as an additional input feature. This project uses the Distributed Memory Model of Paragragh Vectors (PV-DM) described in that paper, with a neural network language model (NNLM) as described in [3].

No prior literature exists on improving search specifically for ExploreCourses, but usage of word vectors in search engines is described in [5] and is widespread today. Additionally, neural networks have been applied to 'topic spotting' [6], a very similar problem to searching for relevant courses. That being said, modern search engines still favor substring matching over semantic matching, and the technique used in [6], where a neural network classifier is trained for each of 92 topics, would be infeasible for our dataset of 14 thousand courses.

Additionally, we initialize training with pretrained word vectors obtained from [7]. The pretrained vectors were trained on roughly 100 billion words from Google News, using a skip-gram model with negative sampling as described in [2].

3 Dataset and Features

Our data comes from the 2015-2016 ExploreCourses course listings [8], and contains information about the 14 thousand courses Stanford offers this year. For each course, we scrape its code, title, and description, giving us a dataset of 35 thousand unique tokens and 1.3 million total tokens.

```
CS106A: Programming Methodology.
Introduction to ... and testing. Uses the Java programming language. ...
```

Our PV-DM model must be trained on a list of labeled sentences, where the label for each sentence corresponds to the course with which it is associated. To obtain the labeled sentence list for a particular course, we apply the following processing steps:

1. Tokenize sentences with the Punkt sentence tokenizer from the NLTK natural language processing library [9]. This tokenizer is an unsupervised machine learning algorithm pretrained on a general English corpus, and it can differentiate between true sentence breaks and periods used in abbreviations. We treat the course title as a sentence.
2. Tokenize words with Penn-Treebank tokenization.
3. Identify and condense likely bigrams. We condense likely bigrams, such as `machine learning`, into a single token `machine_learning`, because the bigram often has a different meaning than the average of the component words. This is done according to the collocation likelihood ratio metric described in [10].
4. Tag each sentence with the course code.

After processing the example course, we obtain the following labeled sentence list:

```
[(['programming', 'methodology'], 'cs106a'),
 (['introduction', 'to', ..., 'and', 'testing', '.'], 'cs106a'),
 (['uses', 'the', 'java', 'programming_language', '.'], 'cs106a'),
 ...]
```

Each of these labeled sentences is one training example, and our training set consists of 73 thousand such labeled sentences.

4 Methods

We use a composite model, averaging the results of two models: A course vector model and a naive Bayes word vector model.

4.1 Training

4.1.1 Word and course vectors

Both models use word and course vectors to obtain query results, so we train a PV-DM model according to [4]. Our PV-DM model uses a NNLM with 1 hidden layer of 300 nodes, and we use Huffman tree hierarchal softmax to efficiently represent our sentences as in [1]. This model produces 300-dimensional vectors, to match the dimensionality of the pretrained vectors.

We begin training by initializing the word vectors to pretrained values, or to random values if unknown. Pretrained words learn at 20% of the normal rate to reflect the difference in how much they have to learn. We train with stochastic gradient descent for 100 epochs, tracking the total absolute difference in vector weights. If the absolute difference does not decrease over 3 epochs, we decrease the learning rate by 25% to improve convergence.

4.1.2 Word-course probabilities

For our naive bayes word vector model, we compute a mapping of words to their most related courses. We make the naive independence assumption (each word in a course's information is

independent of the others), allowing us to use Bayes' theorem to compute the probability that a particular query word w came from a particular course c:

$$P(c \mid w) = \frac{P(w \mid c)P(c)}{P(w)}$$

We compute this probability for every word-course pair and construct a mapping as follows:

```
Java: (cs106a 0.27), (engr70a 0.27), (cs108 0.18),
      (ms&e223 0.12), (cs193a 0.11),  ...
```

4.2 Predicting

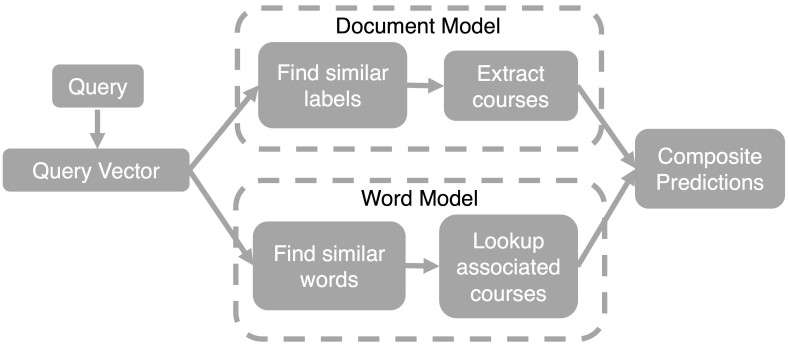

Figure 1: Flow chart illustrating the prediction process.

Upon receiving a query string, we convert the string into a query vector by averaging the word vectors (using the PV-DM model to predict a label vector gives highly variable results). We feed this vector into the two models, described below, obtaining two lists of course suggestions with the confidence for each suggestion. We compute the final list of suggested courses by averaging the confidences for each suggestion. This process is outlined in Figure 1.

4.2.1 Course vector model

Given a query vector v and a requested number of results n, we find the n most similar labels (corse codes) by cosine similarity. We choose cosine similarity over Euclidean distance because cosine similarity accounts more for differences in vector direction. A label's cosine similarity to the query vector represents the model's confidence in that label. We pair each course with its cosine similarity and return the n (course, similarity) tuples.

4.2.2 Naive Bayes word vector model

This model also takes a query vector v and a requested number of results n. It begins by finding the 10 most similar words (synonyms) to the query vector by cosine similarity. We treat the cosine similarity of a synonym s_i to the query vector v as the probability $P(s_i)$ that the user would be satisfied with searching for that synonym. This allows us to compute the probability that the user queried for a particular course c given the query vector v as:

$$P(c \mid v) = 1 - \prod_i (1 - P(s_i)P(c \mid s_i))$$

We take the n most likely courses by this metric, with each course's score given by

$$Score = \frac{1}{1 - P(c \mid v)}$$

representing the model's confidence in the course. We return the list of n (course, score) tuples.

5 Results

In both qualitative and quantitative tests, our model outperforms ExploreCourses. In no cases does our model perform worse.

We seek to increase the relevance of course search results, so we measure success subjectively by analyzing sample queries and conducting user tests. It is often the case that students query ExploreCourses without a particular course in mind; they are using ExploreCourses to find courses that might interest them. Given this behavior, it doesn't make sense to evaluate our model objectively with a test set.

Qualitative evaluation

By inspection, our model outperforms ExploreCourses for a plethora of sample queries. We present three searches that demonstrate some of the different types of queries that students make (Table 1). These queries demonstrate how our model outperforms ExploreCourses in a variety of ways.

Table 1: Sample queries illustrating the difference in top three results generated by ExploreCourses (middle) and our composite model (right)

Query	ExploreCourses Results	Composite Model Results
Computer Programming	Bio-physics of Multi-cellular Systems Physics-Based Simulation Computational Genomics	Introduction to Computing Principles Programming Service Project Introduction to Computers
Violin Lessons	None	Introductory Violin Class Violin Advanced Violin
Linear Algebra	Advanced Feedback Control Design Numerical Methods in Engineering Mechanical Vibrations	Calculus Modern Algebra Linear Algebra and Differential Calculus

The first query, "computer programming" should return introductory computer science classes. However, ExploreCourses returns several bioengineering classes that happen to contain both 'computer' and 'programming' in their course information. In contrast, our model finds relevant courses - CS101, CS192, and CS105. Other relevant results, CS106A, CS106B, and CS106X also appear on the first page of results.

The second query illustrates a problem with ExploreCourses in that both search terms must appear in a course's information in order for it to be found. No courses exist that contain both 'violin' and 'lessons', so ExploreCourses generates zero results. In this case, our model is able to find Stanford's three violin classes because it also looks for synonyms of 'lessons'.

The third query demonstrates searching for a specific topic. It seems fair to assume that a student searching for 'linear algebra' expects to find classes about linear algebra, not classes which merely use it. However, ExploreCourses doesn't differentiate between these two concepts, and in this case it returns the alphabetically-first classes for which knowledge of 'linear algebra' appears as a prerequisite. Our model again outperforms ExploreCourses by finding math classes which teach linear algebra or are highly relevant to classes that do.

Quantitative evaluation

We asked 20 students to assess the performance of our models against ExploreCourses across three query categories. Each student generated 10 queries in a given category before giving an overall relevance score from 1 (not relevant at all) to 5 (extremely relevant) to a specific model in a specific category. In increasing order of generality, the three categories were (i) searching for a specific

course in a particular subject area, (ii) finding any courses on a specific topic or within a specific department, and (iii) finding any course on a general topic. Results are shown in Table 2.

Table 2: Average scores (out of 5) given by n=20 students assessing performance of different models across a variety of search categories

Category	ExploreCourses	Course Vector	NB Word Vector	Composite
Specific course	3.4	2.5	4.2	4.4
Specific topic	2.2	3.1	3.8	4.0
General topic	1.3	3.6	3.2	3.9
Total	**2.3**	**3.1**	**3.7**	**4.1**

Unsurprisingly, ExploreCourses performs poorly across the board. The ExploreCourses results get worse as the queries become more abstract, because of its reliance on exact substring matches. Interestingly, the Course Vector model actually performs better on generic queries, because its course vectors capture abstract, high-context meaning. The Naive Bayes word vector model does extremely well on specific searches, especially because it builds synonymous queries to ensure that any courses close to the query string are produced. As expected, NB Word Vector model is weaker for generic searches, because there are fewer specific search phrases to work with.

Our composite model yields not only the highest relevance results overall, but also the most stable results across the three categories. By combining the strengths of the Course Vector model and the NB Word Vector model, the composite model is able to accurate find good courses whether students search for something as specific as a single course or something as vague as a general topic.

Caveats

As always, we need to be cognizant of overfitting to our training set. However, for this project our training set (the entire Stanford course catalog) represents the entire universe of discourse. Therefore, overfitting isn't a serious problem, because we will never need to generate courses that aren't in our training set. While it's possible that new courses could be added, or old course information changed, this happens at most four times per year, and the model can easily be retrained that frequently.

6 Conclusion

In conclusion, we have built a model that generates a list of highly-relevant courses given a search query by combining two existing models trained on the entire corpus of existing Stanford courses. One component model uses cosign similarity on a query vector to find similar courses, and works very well on general queries. The other component uses a combination of word2vec and Naive Bayes to find synonymous queries and associated courses and works very well on specific queries. These results are then aggregated, weighted by their relative confidences, to yield final results. Overall, the suggested courses have much higher relevance in a number of search categories, and seems to be a qualitative and quantitative improvement over the existing ExploreCourse implementation.

In the future, we hope to integrate our search tool with ExploreCourses in production so we can obtain vital statistics about usage patterns which we can then use to improve the model. Ideally, we would A/B test different parameter choices at scale in order to gain a much more accurate idea of what the 'best', most-relevant courses are for common queries. Additionally, if we replaced our NNLM with an recurrent NNLM, we may see slightly better performance.

References

1. Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. ICLR Workshop, 2013.

2. Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeffrey Dean. Distributed Representations of Words and Phrases and their Compositionality. Advances in neural information processing systems. p3111-3119. 2013.

3. Y. Bengio, R. Ducharme, P. Vincent. A neural probabilistic language model. Journal of Machine Learning Research, 3:1137-1155, 2003.

4. Quoc V Le and Tomas Mikolov. Distributed Representations of Sentences and Documents. 2014.

5. http://www.puttypeg.net/papers/vector-chapter.pdf

6. Wiener, Erik, Jan O. Pedersen, and Andreas S. Weigend. "A neural network approach to topic spotting." Proceedings of SDAIR-95, 4th annual symposium on document analysis and information retrieval. Vol. 317. 1995.

7. https://code.google.com/p/word2vec/

8. http://explorecourses.stanford.edu/

9. Kiss, Tibor and Strunk, Jan (2006): Unsupervised Multilingual Sentence Boundary Detection. Computational Linguistics 32: 485-525

10. Manning, Christopher D., and Hinrich Schtze. Foundations of statistical natural language processing. MIT press, 1999.

11. GenSim: Topic Modelling for Humans. Radim Rehurek. 2009.