This is the way: a Unique Course Recommendation Approach

Erdem Korhan Erdem   Adem Baran Orhan

Abstract

Becoming a developer and trying to find a job in top companies are getting more complex every day. To become more capable developers, people need specialized courses. Today, there are several course recommendation systems to help people develop themselves in specific fields. With a new course recommendation system approach that uses computer engineers' skills and online computer science-related course outcomes as its primary data source, this path is becoming more accessible and easier. To make a meaningful recommendation, we extracted the skills from course descriptions and then compared them with the engineers' skills. Several methods exist to extract entities from texts and several other methods to compare the semantic similarity between pairs of words. In this paper, we will try to bring a new point of view to the course recommendation systems and examine all the required intermediate steps to complete this task.

1. Introduction

Many websites are providing online courses on the internet. Students face many advertisements and course recommendations from college communities, e-mails or other social networks from these websites. Most of the time, these courses are separate from a person's interests. Sometimes courses are recommended by the user-rating rank, which may not be reliable enough since anyone can use bots to get higher course rates. This project's primary purpose is to help anyone who wants to improve in a specific computer science area and is looking for a suitable course on Udemy[1]. The most important and different from any other recommendation system is using the up-to-date dataset from developers from specified areas. Using LinkedIn[2] for creating skill datasets with six different areas. Machine Learning, Back-end, Front-end, DevOps, and QA were chosen initially. Our recommendation approach begins with extracting the skill entities from the course outcome section of the online courses first, then computing the similarities between these skill entities and the existing skill list of engineers currently working in user-specified fields. We have decided to use a natural language processing technique called Named Entity Recognition (NER) to extract skill entities out of phrases explaining courses' outcomes.

Named entity recognition (NER) is a widely studied problem in natural language processing (NLP), with numerous approaches proposed in the literature. Early NER approaches typically relied on hand-crafted rules and dictionaries. Still, more recent approaches have employed machine learning techniques such as hidden Markov models, maximum entropy models, and conditional random fields.

There are various approaches to performing NER, but one common method is to use machine learning algorithms to identify named entities in text. This typically involves training a machine learning model on a large dataset of annotated text, where the named entities have been identified and labeled. The model can then predict the named entities in new, unseen text. Below, you can see a shallow example of how NER is used.

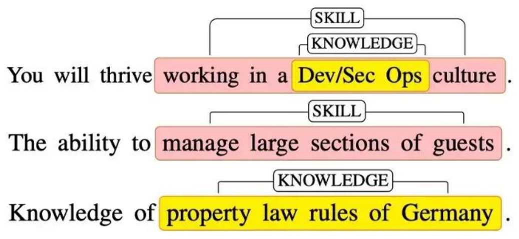

SKILL
KNOWLEDGE
You will thrive working in a Dev/Sec Ops culture .
SKILL
The ability to manage large sections of guests .
KNOWLEDGE
Knowledge of property law rules of Germany .

Figure 1. A basic sample about NER usage

Later on, extracting skill entities, what we need to do is to compute the semantic similarities between extracted skills and already gathered field-specific engineer skills. To fulfil this task, another natural language processing approach exists, word embeddings.

Word embeddings are an important technique in natural language processing representing words as dense vectors in a low-dimensional continuous space. These embeddings capture the semantic relationships between words, allowing for word similarity and analogy-solving tasks.

A simple visualization of the word embedding approach is below.

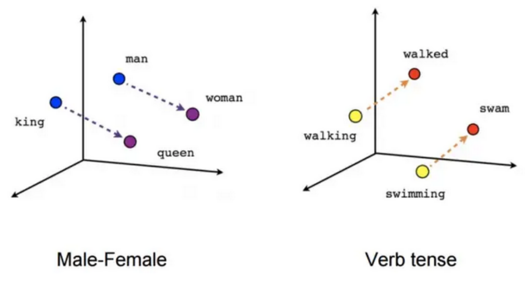


Figure 2. Visualize word vectors

The motivation behind our approach is that by studying the courses that have the potential to provide skills of successful professionals in a particular field, users can gain valuable insights into the knowledge and skills required to succeed in that field. This can be especially useful for individuals who are seeking to transition into a new career or are just starting their professional journey. By recommending courses related to successful professionals' skills, our system can help users build a solid foundation and better prepare themselves for a successful career.

To develop our course recommendation system, we collected data on the online courses and skills of employees in computer science fields. We then used this data to create an algorithm that suggests courses to users based on their career aspirations and the skills of successful professionals in those fields.

The remainder of this paper is organized as follows: In Section 2, we review related work in the field. In Section 3, we describe our methodology, including the research design, data collection, and analysis methods. In Section 4, we present the results of our experimental evaluation, including a discussion of the findings and their implications.

2. Related Work

There has been a significant amount of research in the field of course recommendation, with a focus on personalizing the learning experience for individual students. One approach to course recommendation is to use collaborative filtering, where the recommendations are based on similar users' past behaviours and preferences. For example, [3] proposed a hybrid recommendation system that combined collaborative filtering with content-based filtering, in which the recommendations are based on the characteristics of the courses themselves. Another approach is to use content-based filtering, where the recommendations are based on the characteristics of the courses and the student's past behaviours and preferences. [4] developed a content-based recommendation system that used Latent Semantic Analysis to identify the relationships between courses and to make recommendations based on these relationships.

There is no NER utilization in the context of course recommendation systems in the literature. However, NER is used for various other purposes: identifying and classifying named entities, such as people, organizations, and locations, in unstructured text. There have been numerous approaches to NER, ranging from rule-based methods to machine learning-based methods. Rule-based methods for NER rely on hand-crafted rules and dictionaries to identify and classify named entities. These methods are often language-specific and require significant manual effort to develop and maintain. [5] presented a rule-based approach for NER in Spanish, using a combination of regular expressions and a manually compiled list of named entities. [6] proposed a rule-based approach for NER in Chinese, using a combination of dictionaries and patterns.

Machine learning-based methods for NER do not require hand-crafted rules and dictionaries but instead, learn to identify and classify named entities from a labeled training dataset. [7] presented a machine learning-based approach for NER in English, using a Conditional Random Field (CRF) model trained on the CoNLL 2003 dataset. [8] proposed a machine learning-based approach for NER in French, using a CRF model trained on a manually annotated dataset.

Recent approaches have also explored the use of deep learning for NER. [9] introduced a convolutional neural network (CNN)-based approach for NER in English, using a combination of word embeddings and character-level CNNs. [10] proposed a bidirectional long short-term memory (BiLSTM)-based approach for NER in German, using pre-trained word embeddings and a BiLSTM model.

One of the most widely used deep learning models for NER is BERT (Bidirectional Encoder Representations from Transformers). BERT is a transformer-based model that has achieved state-of-the-art results on a wide range of NLP tasks, including NER. [11] showed that fine-tuning a BERT model on the CoNLL 2003 NER dataset resulted in significant improvements over previous state-of-the-art results. [12] demonstrated that using BERT for NER in a zero-shot setting, where the model is trained on one language and tested on another without additional fine-tuning, can still achieve competitive results.

In summary, there are various approaches to NER, ranging from rule-based methods to machine learning-based methods to deep learning-based methods. While rule-based methods are simple and language-specific, machine learning-based methods and deep learning-based methods are more flexible and can generalize to multiple languages. BERT is a particularly powerful deep learning-based approach that has achieved state-of-the-art results on various NER tasks.

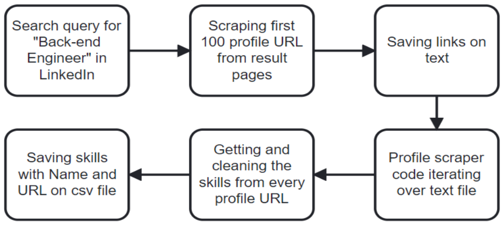
3. The Approach

3.1. Dataset

Since there does not exist a publicly available dataset suitable for our task, we have collected the required data ourselves. As explained, in this manner of recommendation system, we needed skills of engineers working in specific computer science fields: Machine Learning, Back-end, Front-end, DevOps, QA, and outcome sections of online courses. Udemy was our main platform preference for gathering the required course outcome data since it has a well-structured format of phrase-by-phrase course outcomes and due to the variety of courses it provides. To collect the skills of engineers, we decided to use the LinkedIn platform. LinkedIn provides a 'skills' section in personal profiles so that users can publicly share their skills with others.

To collect the required data accordingly, we took advantage of libraries Selenium and BeautifulSoup for web scraping. During the scraping, we stored the data as a dictionary with the person's name and a list of the skills. LinkedIn skills contain some words such as endorsements and approval by LinkedIn. During the scrap, we also cleaned the unwanted keywords. Starting with LinkedIn skills.

The process of getting skills from developers is as follows:

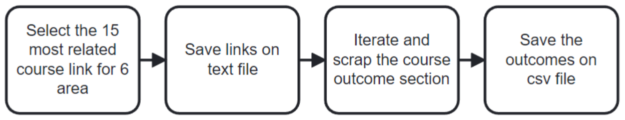


Search query for "Back-end Engineer" in LinkedIn
Scraping first 100 profile URL from result pages
Saving links on text
Profile scraper code iterating over text file
Getting and cleaning the skills from every profile URL
Saving skills with Name and URL on csv file

Figure 3. Diagram about scraping LinkedIn skills

Example from the Backend dataset: '4,'https://www.linkedin.com/in/jingning***/': 'name': 'Jing*****', 'skill': ['C++', 'Java', 'Matlab', 'Machine Learning', 'Python', 'PHP', 'Programming', 'Microsoft SQL Server', 'OpenGL', 'OpenCV', 'Analysis', 'Microsoft Office', 'HTML', 'Machine Learning', 'Programming', 'OpenCV', 'C++', 'Java', 'Matlab', 'Python', 'PHP', 'Microsoft SQL Server', 'OpenGL', 'Microsoft Office', 'HTML', 'Analysis']

The process of getting course outcomes from Udemy is as follows:



Select the 15 most related course link for 6 area
Save links on text file
Iterate and scrap the course outcome section
Save the outcomes on csv file

Figure 4. Diagram about scraping Course Outcomes

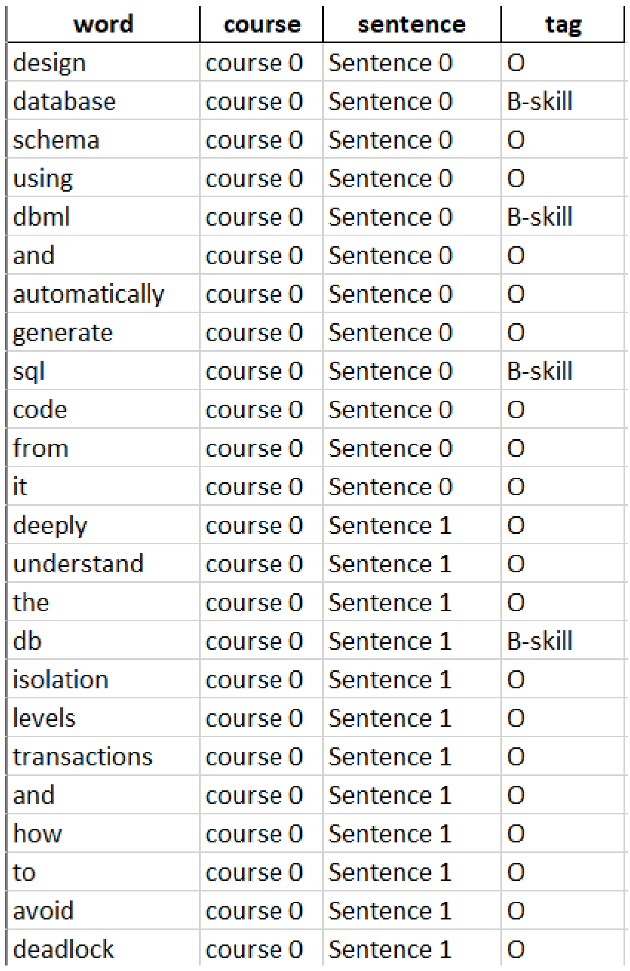
Example from one course in the Backend area:

0,'https://www.udemy.com/course/backend-master-class-golang-postgresql-kubernetes/': 'Outcomes': ['Design database schema using DBML and automatically generate SQL code from it', 'Deeply understand the DB isolation levels, transactions and how to avoid deadlock', 'Automatically generate Golang code to interact with the database', 'Develop a RESTful backend web service using the Gin framework', 'Secure the APIs with user authentication, JWT and PASETO', 'Write stronger test set with high coverage using interfaces and mocking', 'Build a minimal Docker image for deployment and use Docker-compose for development', 'Set up Github Action to automatically build and deploy the app to AWS Kubernetes cluster', 'Register a domain and config Kubernetes ingress to route traffic to the web service', "Enable automatic issue & renew TLS certificate for the domain with Let's Encrypt", 'Take your web service to the next level with gRPC and gRPC gateway', 'Run background workers to process tasks asynchronously with Redis and Asynq']

Since we need the datasets as labeled to perform Named Entity Recognition, we ended up tagging all the words from the outcomes of the courses. Before starting the tagging process, we cleaned the punctuation, extra spaces and lowered the words.

To label the data appropriately, we needed to tag all the words according to BIO tagging scheme. In natural language processing (NLP), the BIO tagging scheme is used to annotate text with information about the entities mentioned in the text. The BIO scheme stands for "beginning, inside, outside," and is used to distinguish between the beginning of an entity, the inside of an entity, and the text outside of an entity.

Later on, we separated each word by keeping the sentence and course information, then tagged them accordingly. Below, you can see a sample set of tagged words from our own dataset.



word	course	sentence	tag
design	course 0	Sentence 0	O
database	course 0	Sentence 0	B-skill
schema	course 0	Sentence 0	O
using	course 0	Sentence 0	O
dbml	course 0	Sentence 0	B-skill
and	course 0	Sentence 0	O
automatically	course 0	Sentence 0	O
generate	course 0	Sentence 0	O
sql	course 0	Sentence 0	B-skill
code	course 0	Sentence 0	O
from	course 0	Sentence 0	O
it	course 0	Sentence 0	O
deeply	course 0	Sentence 1	O
understand	course 0	Sentence 1	O
the	course 0	Sentence 1	O
db	course 0	Sentence 1	B-skill
isolation	course 0	Sentence 1	O
levels	course 0	Sentence 1	O
transactions	course 0	Sentence 1	O
and	course 0	Sentence 1	O
how	course 0	Sentence 1	O
to	course 0	Sentence 1	O
avoid	course 0	Sentence 1	O
deadlock	course 0	Sentence 1	O

Figure 5. Tagged example from Backend dataset

In our course recommendation system, we only deal with the ’skill’ entities inside the phrases. Therefore, we have only labeled the skills inside the texts. For instance, the word ’design’ in the figure above may refer to one of the other various entities. However, it is labeled as ’O’ since the skills are our main interest.

To extract computer science-related skills from raw phrases using NER, there are several possible model architectures, such as RNNs, GRUs, LSTMs, and Transformer based models. Some of the approaches that we mainly used are listed in the next section.

3.2. LSTM

Long Short-Term Memory (LSTM) networks are a type of recurrent neural network that is particularly well-suited to modelling long-term dependencies in data. They were introduced by Hochreiter and Schmidhuber in 1997, and have since become a popular choice for a wide range of tasks, including natural language processing, speech recognition, and time series forecasting.

At a high level, LSTM networks consist of a series of ”memory cells” that are connected through a series of ”gates.” The gates, which are implemented using logistic regression units, control the flow of information into and out of the memory cells, allowing the network to selectively store or forget information as needed.

To compute the output of an LSTM cell, the following steps are performed:

1. The input, forget, and output gates are activated using the current input and the previous output of the cell.

2. The input and forget gates are used to update the cell state, which is a weighted sum of the previous cell state and the current input, with the input gate determining the weight of the current input and the forget gate determining the weight of the previous cell state.

3. The output gate is used to compute the output of the cell, which is a weighted sum of the cell state and the current input, with the output gate determining the weight of the cell state.

LSTM networks are trained using backpropagation through time, which involves unrolling the network into a feedforward network and applying the standard backpropagation algorithm. This allows the network to learn long-term dependencies by adjusting the weights of the gates over multiple time steps.

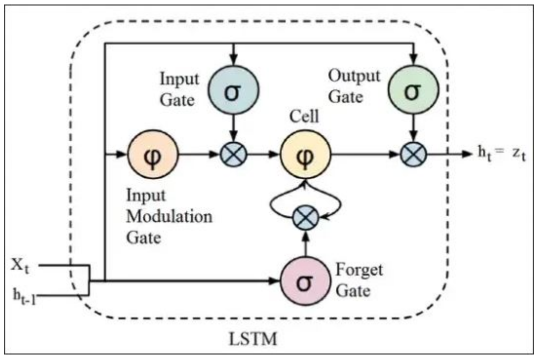


Figure 6. LSTM cell, from https://medium.com/@kangeugine

3.3. Bidirectional LSTM

Bidirectional Long Short-Term Memory (LSTM) networks are a variant of LSTM networks that are able to take into account both past and future context. They are particularly useful for tasks such as natural language processing, where the meaning of a word can depend on the context in which it appears.

A bidirectional LSTM network consists of two separate LSTM networks, one processing the input sequence in forward order and the other processing the input sequence in reverse order. The output of the two LSTMs is then combined at each time step, allowing the network to take into account both past and future context.

Bidirectional LSTMs can be trained using the same techniques as regular LSTMs, including backpropagation through time. However, due to the additional complexity of the network, they may be more computationally expensive to train and may require more data to achieve good performance.

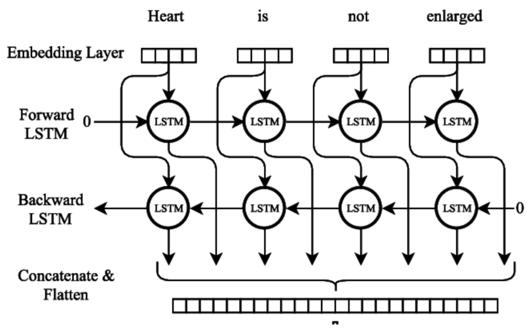
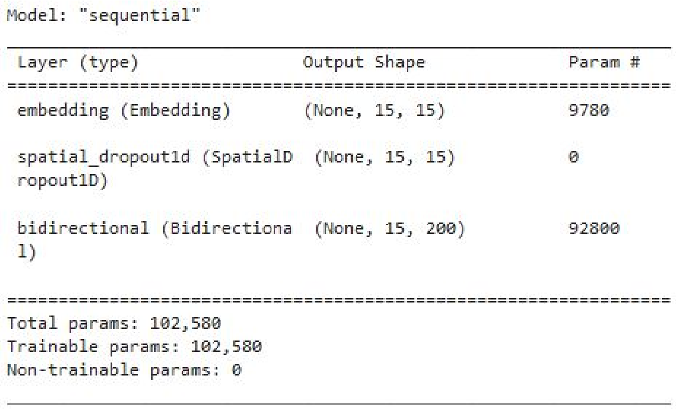


Figure 7. Bidirectional LSTM [13]

As our first approach to fulfil the Named Entity Recognition task, we decided to implement the biLSTM-based model architecture. To make our data suitable for training a biLSTM model, we have first constructed a corpus using only Machine Learning related courses' outcomes. By using this corpus, our raw phrases had been represented with numbers instead of words. Besides that, we have also constructed a corpus for our tags. After these steps, we tried to train a shallow model to extract Machine Learning related skills from Machine Learning course descriptions.

The general structure of the biLSTM-based NER model's architecture is as follows: an Embedding layer, followed by a one-dimensional spatial dropout layer and a bidirectional LSTM layer with 256 units.

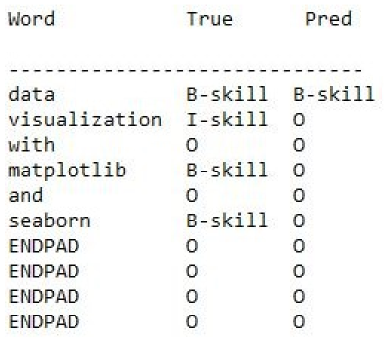
We chose Adam as the optimizer by its default learning rate, and sparse-categorical-crossentropy as our loss function.



```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding (Embedding)       (None, 15, 15)            9780

 spatial_dropout1d (SpatialD  (None, 15, 15)           0
 ropout1D)

 bidirectional (Bidirectiona  (None, 15, 200)          92800
 l)

=================================================================
Total params: 102,580
Trainable params: 102,580
Non-trainable params: 0
_________________________________________________________________
```

Figure 8. Summary of the biLSTM model

We have used the 0.75 part of our text corpus as a train set, 0.15 as validation, and 0.1 part for testing the model's performance. We have trained the model for 3 epochs, and have reached around 0.9 validation accuracy. It seemed successful at first, however, the fact we faced when we manually test the model was predicting almost all the words with tag-O (outside of an entity), and getting high validation accuracies since most of the sentences in the dataset include 3-4 skill entities at most. Here is an example phrase in our dataset, predicted by the shallow model:



```
Word           True      Pred

------------------------------
data           B-skill  B-skill
visualization  I-skill  O
with           O        O
matplotlib     B-skill  O
and            O        O
seaborn        B-skill  O
ENDPAD         O        O
ENDPAD         O        O
ENDPAD         O        O
ENDPAD         O        O
```

Figure 9. Sample phrase predicted by the model

Above, there exist words named 'ENDPAD', and their main purpose of usage is to set all the phrases in the dataset to the same length. Setting a maximum length for the phrases is a design choice and depends on the phrases inside the dataset. In this sample, the length of the phrases has been set to 10 at maximum, therefore a phrase that has words less than 10 is basically set to a length of 10 by adding ENDPAD words at the end.

3.4. BERT

Bidirectional Encoder Representations from Transformers (BERT) is a state-of-the-art NLP model developed by Google. It is based on the transformer architecture, which uses self-attention mechanisms to capture dependencies between words in a sentence. BERT uses a multi-layer transformer encoder to generate a fixed-length representation of a variable-length input sequence. It can be fine-tuned for a wide range of NLP tasks, including named entity recognition.

In the context of NER, BERT can be fine-tuned to classify each word in a sentence as a specific entity type. This is typically done by adding a classification layer on top of the BERT model and training it to predict the entity labels for each word. Because BERT is pre-trained on a large dataset, it can often achieve superior performance on NER tasks compared to models that are trained from scratch.

One of the pre-trained models of BERT is best-base-uncased model. The ”base” version of BERT has 12 layers and 110 million parameters, while the ”large” version has 24 layers and 340 million parameters. The ”uncased” version of BERT has lowercase all the words in the vocabulary, while the ”cased” version preserves the case of the words.

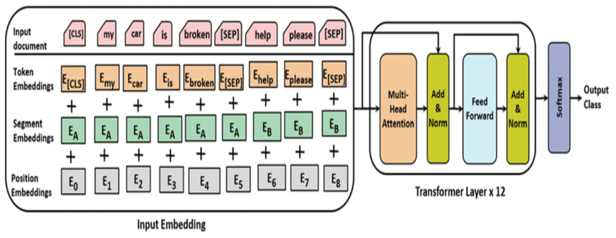


Figure 10. BERT-base model architecture

Above, you can see the basic architecture of BERT's base classifier model. By default, BERT does not provide a decoder and only exists encoders. Decoders are explicitly needed in text-to-text generations. In our case, we take advantage of the BERT base model for the Named Entity Recognition task. In other words, NER is also named 'Token Classification'. Since we do not generate new text using existing texts in our dataset but only classify them, we do not need to add an explicit decoder in the fine-tuning process. The only step needed in fine-tuning process for Token Classification is adding a classification layer at the end of the model, as seen at the rightmost of figure-10.

Our BERT-base model for the Token Classification task was trained with different sets of hyperparameters. Some hyperparameter options we have made experiments on are optimizers, learning rate, batch size and the number of epochs. Details of the experiments and the final obtained metrics will be discussed in section 4.

3.5. Word2vec

Later on, extracting skills using a NER model, we needed to compute the similarities between extracted skills and the pre-collected skills list. For this purpose, we have taken advantage of Word2vec word embedding models.

Word2vec is a popular technique for generating word embeddings, which are low-dimensional representations of words in a text corpus. These embeddings capture the context in which words appear, and they can be used to identify relationships between words and perform various natural language processing tasks. Word2vec captures semantic relationships between words by using the context in which words appear to learn their meanings. In the context of this approach, if two words often appear in similar contexts (i.e., they are often used in the same sentences), then it is likely that they have similar meanings. For instance, the words 'machine' and 'learning' are highly likely to appear together in the course outcomes of artificial intelligence-related courses. Thus, it will lead the Word2vec model to set the vectors of these words similarly.

Word2vec has been widely used for various natural language processing tasks, and it has been shown to produce high-quality word embeddings that capture the semantic and syntactic relationships between words.

Again, the Word2vec model we have created was trained with various sets of hyperparameters. In the experiments, we construct the optimal model by making experiments on the hyperparameters: size, window, and minimum count. These parameters refer to the length of a vector in the vector space of the model, the number of words to look at both before and after a target word in the training step to find the relations and minimum frequency required for a word to be included in the model corpus, respectively. Details of the experiments and final results will be discussed in section 4.

3.6. Recommendations

After extracting skill entities out of raw course outcomes by using a BERT-based NER model and then constructing a Word2vec model to compute similarities between the extracted skills and pre-collected skills, the general structure of our recommendation system was built. We have matched the extracted skills with each course by extracting skill entities out of courses' outcomes. Finally, we ended up with a dictionary with keys are courses, whereas values are the list of skill entities belonging to its key, which is a course. Later on, we aimed to compute the similarities between each course in this dictionary and already gathered employee skills. To find the most satisfactory results, there were several options we could try while performing the similarity computation task. Details of these experiments and final results will be discussed in section 4.

4. Experimental Evaluation

4.1. Bidirectional LSTM Model

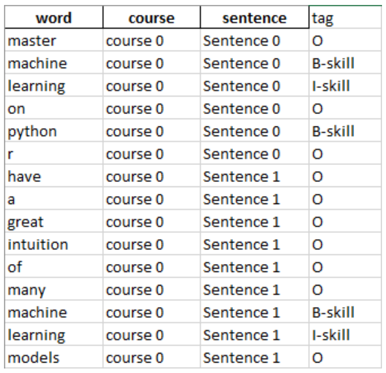
In section 3.3, we have explained the details of general architecture and some parameter options for the bidirectional LSTM model constructed for the NER task. Normally, this type of model architecture is capable of capturing entities from texts. However, in our case, the data which the model was trained was not large enough.

As a result of this experiment, we have concluded that our current data may not be sufficient to train a model from scratch, which can give us satisfying results in the Named Entity Recognition process.

4.2. Fine-Tuned BERT-base-uncased

Later on, deciding from-scratch biLSTM model was not capable enough to fulfil the NER task, we decided to fine-tune the pre-trained best-base-uncased model.

After formatting the dataset, shown in figure 11,



word	course	sentence	tag
master	course 0	Sentence 0	O
machine	course 0	Sentence 0	B-skill
learning	course 0	Sentence 0	I-skill
on	course 0	Sentence 0	O
python	course 0	Sentence 0	B-skill
r	course 0	Sentence 0	O
have	course 0	Sentence 1	O
a	course 0	Sentence 1	O
great	course 0	Sentence 1	O
intuition	course 0	Sentence 1	O
of	course 0	Sentence 1	O
many	course 0	Sentence 1	O
machine	course 0	Sentence 1	B-skill
learning	course 0	Sentence 1	I-skill
models	course 0	Sentence 1	O

Figure 11. ML courses dataset for NER

We split the data to train, validation and test sets with the ratio 75-15-10. For the Named Entity Recognition model, we tried different combinations of hyperparameters:

num_train_epochs = [4, 6, 8]

learning_rates = [0.01,0.0001, 0.00001]

batch_sizes = [16, 32, 64].

And get the best results for 8 epochs, 0.0001 learning rate with 32 batch sizes. For these optimal hyperparameters, obtained results are:

- Evaluation Loss: 0.3321927636861801
- Precision: 0.84375
- Recall: 0.6428571428571429
- F-1 Score: 0.7297297297297298

Eventually, hyperparameters (8 epochs, 0.0001 learning rate with 32 batch sizes) were used for training BERT-base-uncased.

After testing the model on unseen test data, we obtained satisfactory results. Here is an example of predicting skill entities for a random sentence.

[['master': 'O', 'machine': 'B-SKILL', 'learning': 'I-SKILL', 'on': 'O', 'sql': 'B-SKILL']]

4.3. Word2Vec Model

In section 3.5, we have explained why we preferred Word2Vec for the word embedding task. This section will give more details about the experiments and how we constructed the optimal model. Before constructing a word2vec model, we applied some preprocessing and formatting steps to use word2vec. Below you can see the first row of the dataset with the first sentence of outcomes. This is before the preprocessing and formatting of the data.

0,'https://www.udemy.com/course/machinelearning/': 'Outcomes': ['Master Machine Learning on Python & R',...

During the preprocessing, we lowered the sentences, removed stopwords, punctations and lemmatized the words. Lemmatizing words are preferred to stemmers for getting more clean course outcomes. For instance, if we have "microprocessors" in the dataset, we are converting it to "microprocessor" by lemmatising. This gives better results when we compare it with the skills since skills are mostly not plural. Before training the model, we created a corpus from the updated course outcomes. We have done something creative in selecting the hyperparameters of the word2vec model. For optimal hyperparameters, first, we found the most frequent pairs in the course outcomes dataset. We are interested in the pairs because we will use them by calculating pair similarities and getting the best hyperparameters values, giving the best similarity scores for pairs.

Most frequent pairs in the dataset: [(('machine', 'learning'), 37), (('neural', 'network'), 16), (('data', 'science'), 11), (('learning', 'algorithm'), 9), (('deep', 'learning'), 8)]

More clearly, we give the "machine" and "learning" to our function, which is finding the best similarity score for these 2 words by starting from min_size=50 to max_size=100, min_window=1 to max_window=10 with min_count=1, workers=4. And we have done this with machine learning, neural network, and data science pairs.

In the end, we selected the hyperparameters size=99, window=10, min_count=1, workers=4 and 15 epochs, which gives optimal similarity scores for our pairs. Figure 12 shows the pair score and gives a reasonable similarity score for words.

4.4. Results

After finding the entities from course descriptions, we get the top 15 frequent engineer skills for the specific area. We only considered Machine Learning courses and Machine Learning engineer skills for proof-of-concept. For engineers, here are the top 15 frequent skills of engineers. ['python', 'java', 'c++', 'machine', 'learning', 'sql', 'c', 'deep', 'learning', 'matlab', 'data', 'analysis', 'r', 'tensorflow', 'algorithms', 'data', 'mining', 'linux', 'mysql']

We used two different methods for getting the most similar and non-similar courses. Firstly, we compared the entities of courses with engineers' skills by averaging their similarities. For similarity, wv.similarity function is used, calculating the cosine similarity between pairs of words on the background. Figure 14 shows the result of this first computation. Secondly, we compared the entities of courses with the skills of engineers by finding minimum and maximum similarities. Initially, set the min score to 1.0 and the max score to 0.0, then get the similarity with wv.similarity function. In the end, get the minimum of the min score and similarity and get the maximum of the max score and similarity. Figure 12 shows the results of this second computation. We just used min scores on the graph since our max values become 1 because the same words make the maximum similarity score equal to 1. After looking at the plot, we can say that the minimum and maximum similarities are relatively high; we already know the maximum score is high for all the courses. Like in course 4, it could indicate that the words in the course are generally more similar to the skills.

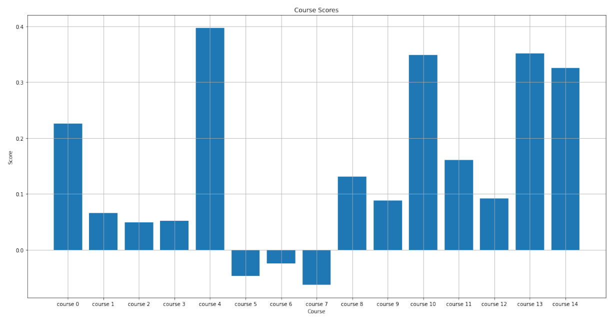


Figure 12. Results of the course scores by min-max similarities. y-axis similarity score and x-axis course index

4.5. Evaluation Metric

To examine the final capability of our recommendation approach, we decided to evaluate our recommendations by a ranking evaluation metric, Mean Average Precision (MAP).

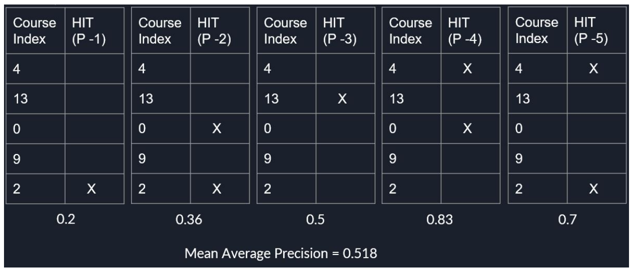
Mean Average Precision (MAP) is a commonly used evaluation metric for recommendation systems. It measures the average precision of the recommendations made by the system at different cut-off points.

In recommendation systems, precision is defined as the number of relevant items (items the user likes) in the recommended list divided by the total number of items in the recommended list. Mean Average Precision (MAP) is a measure of the overall precision of a recommendation system, calculated as the mean of the precision scores at different cut-off points.

For example, a recommendation system recommends 10 items to a user. If 4 of the recommended items are relevant to the user, the precision at the first cut-off point (top-1) is 0.4 (4 relevant items / 10 recommended items). If the next relevant item is the 7th item in the list, the precision at the second cut-off point (top-2) is (4 relevant items + 1 relevant item) / 7 recommended items = 0.57. The MAP is calculated by taking the mean of the precision scores at all cut-off points.

MAP is a commonly used evaluation metric for recommendation systems because it considers the ranking of the recommended items and not just their relevance. It is a more informative metric than precision alone, which only considers the relevance of the top-k recommended items.

Since our system reflects a new recommendation approach and is currently not in use, we were not able to collect exact feedback from actual users. However, to evaluate the recommendation performance of our system in an abstract manner, we decided to survey 5 computer engineering students who have pre-knowledge on Machine Learning to rank the first five courses from our course dataset.



Course Index	HIT (P -1)	Course Index	HIT (P -2)	Course Index	HIT (P -3)	Course Index	HIT (P -4)	Course Index	HIT (P -5)
4		4		4		4	X	4	X
13		13		13	X	13		13	
0		0	X	0		0	X	0	
9		9		9		9		9	
2	X	2	X	2		2		2	X
0.2		0.36		0.5		0.83		0.7	

Mean Average Precision = 0.518

Figure 13. Results of the survey

The results of our survey can be seen in Figure 14. In this figure, each table represents the preferences of each distinct student who participated in the survey. The left part of the table represents the recommendations, while the right part represents the intersection between users' top-5 preferences and our recommendations. After calculating the average precision for each student and computing the mean of all, we obtained 0.518 as the MAP value of our recommendation system, which can tell us that the system can make average-level recommendations.

4.6. Limitations of Approches

One limitation of our approach is that if a course description contains only a single skill entity, such as ”machine learning”, our system might lead to a bias towards that single entity. In such cases, courses with more entities would give more generalized similarity scores. One way to avoid this type of bias is to set a minimum number of entities required for a course.

Figure 14, for the Machine Learning courses, shows the highest score for course 4. After checking manually, we saw that course 4 had a single entity, ”machine learning”.

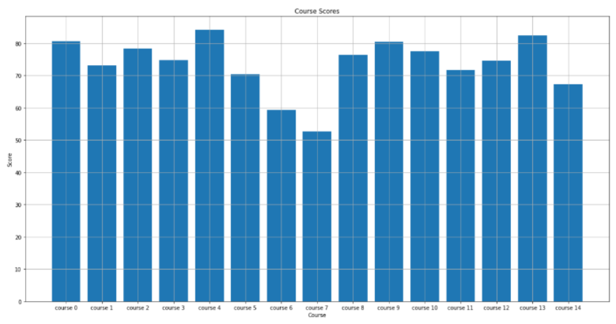


Figure 14. Results of the course scores by averaging sum of scores. y-axis similarity score and x-axis course index

The second limitation of our recommendation system is that it assumes consistency between course descriptions and course materials. In a case where an inconsistency occurs between course outcomes and materials, recommendations will get highly likely be irrelevant.

5. Conclusion

While the use of BERT significantly improved the performance of our skill recommendation system, there is still room for further improvement. We only use the skill entities extracted from course descriptions as a basis for recommendations. However, incorporating other sources of information, such as engineers' job titles, work experience, and education, could provide more personalized and accurate recommendations. You can check project details from Github repository.[1]

5.1. Future Direction

In this research, we tried to prove the concept of a recommendation system that uses field experts' skills as a reference. We used only the skills of engineers working in the Machine Learning area and Machine Learning related courses. However, this recommendation approach is not limited to only one field. As long as the data required for a field is accessible, this recommendation system can be applied to any domain. Since we have the web scraper code for both Udemy and LinkedIn it will not take much time for us to extend the system. For instance, we want to make recommendations in the Finance area. All we need is finance professionals skills from LinkedIn and Finance related courses.

The second work for the future is applying Named Entity Normalization(NEN). Named Entity Normalization (NEN) converts named entities, phrases containing a specific person, place, or thing, into a standard form. There are several approaches to NEN, including:

- Dictionary-based: This approach uses a pre-defined dictionary of named entities and their corresponding normalization forms. For example, if the dictionary contains the entry ”New York City = NY”, then all occurrences of ”New York City” in the text would be replaced with ”NY”.

- Rule-based: This approach uses predefined rules to transform named entities into their normalized form. For example, a rule might state that ”Dr.” occurrences should be replaced with ”Doctor”.

- Machine learning-based: This approach involves training a machine learning model on a dataset of named entities and their corresponding normalization forms. The model can then predict the normalization form for named entities in new texts.

With NEN, we can improve the accuracy and effectiveness of the system. In our case, ML and AI can be normalized to get better results.

[1]https://github.com/a-baran-orhan/AIN311_Project

6. Citations and References

[1] Udemy. (n.d.). Online courses - learn anything, on your schedule. Retrieved December 17, 2022, from https://www.udemy.com/

[2] LinkedIn. (n.d.). Retrieved December 17, 2022, from https://www.linkedin.com/

[3] Yoon, K. Y., & Kwon, H. K. (2009). A hybrid recommendation system for e-learning. Expert Systems with Applications, 36, 9495-9503.

[4] Liu, Y., Liu, H. (2011). A content-based recommendation system for e-learning. Educational Technology & Society, 14, 170-180.

[5] Vila, J. M., Martínez, M. A., & Martínez, J. A. (2004). A rule-based approach for named entity recognition in Spanish. In Proceedings of the International Conference on Natural Language Processing and Information Systems (pp. 171-176).

[6] Li, Z. C., Lu, Y. R., & Li, Y. F. (2004). A rule-based approach for named entity recognition in Chinese. In Proceedings of the International Conference on Natural Language Processing and Information Systems (pp. 183-188).

[7] Miwa, S., & Matsuo, Y. B. (2014). Named entity recognition using a feature-rich conditional random field. In Proceedings of the Conference on Empirical Methods in Natural Language Processing (pp. 964-972).

[8] Bontcheva, K. K., & Maynard, D. (2015). Named entity recognition using a linear chain CRF and a rich feature set. In Proceedings of the Conference on Computational Linguistics and Intelligent Text Processing (pp. 97-108).

[9] Kim, Y. (2014). Convolutional neural networks for sentence classification. In Proceedings of the Conference on Empirical Methods in Natural Language Processing (pp. 1746-1751).

[10] Maas, A. L., Ng, A. Y., & Potts, B. P. (2011). Learning word vectors for sentiment analysis. In Proceedings of the Conference on Computational Natural Language Learning (pp. 142-150).

[11] Devlin, D., Chang, M.-W., Lee, K., & Toutanova, K. (2019). BERT: Pre-training of deep bidirectional transformers for language understanding. In Proceedings of the Conference on North American Chapter of the Association for Computational Linguistics: Human Language Technologies (pp. 4171-4186).

[12] Conneau, A., Kratzwald, G., Schwenk, H., & Lecun, Y. (2020). XLM: Unsupervised cross-lingual representation learning at scale. In Proceedings of the Conference on International Conference on Learning Representations.

[13] Cornegruta, S., Bakewell, R., Withey, S., & Montana, G. (2016). Modelling Radiological Language with Bidirectional Long Short-Term Memory Networks. arXiv.org. https://doi.org/10.48550/arXiv.1609.08409