Západočeská univerzita v Plzni
Fakulta aplikovaných věd
Katedra informatiky a výpočetní techniky

Diplomová práce

Vylepšení systému podpory rozhodování o přenositelnosti kreditů mezi předměty

Plzeň 2018

Bc. David Herman

University of West Bohemia
Faculty of Applied Sciences
Department of Computer Science and Engineering

Master's thesis

Improvement of a Computer Aided System for Course Articulation

Pilsen 2018

Bc. David Herman

Místo této strany bude zadání práce.

Declaration

I hereby declare that this master's thesis is completely my own work and that I used only the cited sources.

Pilsen, 14th May 2018

Bc. David Herman

Abstract

This thesis follows the cooperation between the University of West Bohemia and an American company Owen Software Development Company. During the cooperation, the research teams have developed the system for automatic course transferability decision making. The primary goal of this thesis is to analyze and improve upon the existing transferability decision system. The thesis describes multiple improvements that have been implemented to the existing system. The essence of implemented improvements is to capture the semantic information from the descriptive attributes of the courses. The results of the designed experiments have proved that we successfully increased the existing system performance. We have achieved the best results by employing the artificial neural networks; we have increased accuracy by 11.2%.

Abstrakt

Tato práce navazuje na spolupráci mezi Západočeskou Univerzitou a Americkou společností Owen Software Development Company. V rámci této spolupráce byl vytvořen systém pro automatické rozhodování o přenositelnosti kreditů mezi dvěma předměty. Hlavním cílem této práce je analýza stávajícího systému, návrh a implementace jeho vylepšení. V práci je popsáno a implementováno několik možných vylepšení stávajícího systému. Při návrhu vylepšení je hlavní důraz kladen na zachycení sémantické podobnosti názvu a popisu kurzů. Pro ověření kvality jednotlivých implementovaných vylepšení je v práci provedeno několik experimentů, které prokázaly, že se stávající systém podařilo vylepšit. Nejlepších výsledků je dosaženo za použití umělých neuronových sítí, kdy se stávající systém podařilo vylepšit o 11.2%

Acknowledgment

I would like to thank my thesis advisor, Ing. Miloslav Konopík, Ph.D, for the patient guidance, encouragement and advice he has provided throughout my work on this thesis.

Contents

1 Introduction

This introductory chapter presents an opening to the task, and challenges which are related to this thesis. Also, we summarize the thesis structure at the end of this chapter.

1.1 Task Introduction

Many postsecondary institutions such as colleges or universities offer various courses similar in terms of content. However, while the topic of the courses might be similar; its name, description, and other attributes may vary. The process of deciding whether credits can be transferred between the courses despite different attributes of the both courses is called *Course Articulation.* The courses are usually from different institutions, that way the *Course Articulation* task decides whether the knowledge and experience obtained during completing some course on some institution meets the same level as when completing another course at another institution.

Universities have developed databases for exploring course transferability. These databases are created manually and they are often used by students who are transferring between the institutions to know what courses they will not have to retake on the institution they are transferring to. Development and research of an automatic system for *Course Articulation* is the primary goal of this thesis.

Many courses are asymmetrical by nature; in other words, courses are often transferable in only one way. This is because the courses have different difficulty and expertise level and thus the content of one the courses may be just a subset of another course's content. The asymmetricity of the courses is the main challenge when creating the automatic system for *Course Articulation.* However, setting the symmetrical assumption for the courses should not significantly influence the overall task results but only slightly increase a *false-positive* metrics. In this thesis, we assume symmetrical courses.

The UWB[1] developed an automatic transferability decision system as a result of cooperation between the UWB and an American company *Owen Software Development Company* [45]. In this thesis, the UWB research [18, 19, 45] is replicated and extended using the approaches for the short text similarity task, which is the task related to the *Course Articulation.*

[1]University of West Bohemia

1.2 Thesis Structure

The thesis is structured as follows. The chapter 2 provides the necessary theoretical background for the *Course Articulation* task and especially for the *Short text similarity* task. The chapter 3 describes the existing system developed by the UWB. Chapter 4 describes the improvements made to the UWB system. The experimental results are detailed in chapter 5, and in chapter 6, we conclude the thesis.

2 Theoretical Background

In this thesis, we treat the *Course Articulation* task as the *Short text similarity* task meaning that we decide the course pair transferability solely on a semantic relationship between the descriptive attributes of the courses – its name and description. This chapter contains a theoretical background related to the methods used when solving the task.

2.1 Preprocessing

Most of the tasks related to NLP[1] (including text similarity) requires input data preprocessing. The preprocessing phase includes *text cleaning*, *tokenization*, *stopwords removal*, *stemming*, and *lemmatization*. The figure (2.1) visualizes the usual workflow of a preprocessing phase. We explain each step in the following subsections. The steps done during the preprocessing phase are usually language dependent. In other words, the preprocessing workflow requires to be tailored for the particular task and language.

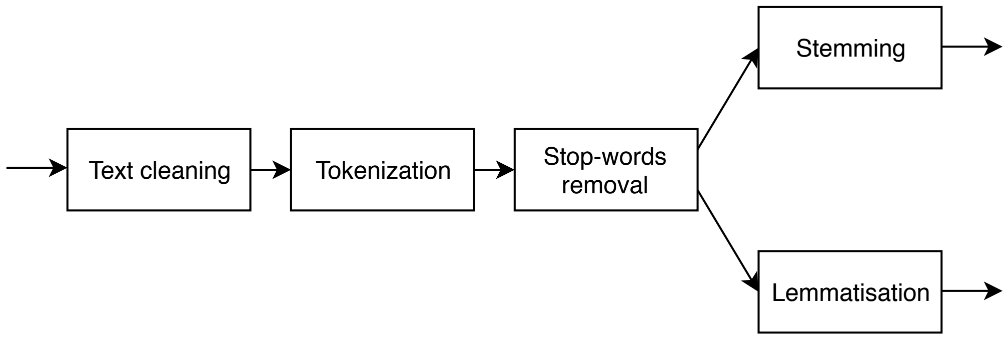

Figure 2.1: Visualisation of a preprocessing workflow

2.1.1 Text Cleaning

Usually, the first step during the preprocessing phase is the *text cleaning*. To clean the text is to remove any symbols and characters that are not related to the content but have more of a syntactical meaning. Such symbols can be emoticons, punctuations, or any forms of a markup language (e.g. HTML[2])

[1]Natural Language Processing
[2]Hypertext Markup Language

2.1.2 Tokenization

Tokenization [24] is the process of breaking a continuous character sequence into a sequence of entities called tokens. A program that performs tokenization is called *tokenizer*. Typically, tokens are understood as words, thus delimited by spaces. However, the *tokenizer* needs to be modified for the input language since some languages do not identify words by spaces (e.g., *Chinese*, or *Korean*).

2.1.3 Spellchecking

Spellchecking [24] is the process of identifying and correcting the misspelled words in the text. We differentiate between the *isolated-term* correction and *context-sensitive* correction.

The *isolated-term* correction treats the input text as a separate sequence of words. This approach can correct only the spelling errors that result into a character sequence that is not in the vocabulary of the spellchecker. The *context-sensitive* spelling correction provides more advanced detection of the spelling errors. For example, the sentence *"I am form Czechia"* would pass the *isolated-term* based spellchecker without errors. However, is it clear that the word *"form"* should be corrected to *"from"*. These types of spelling errors are caught by the *context-sensitive* correction.

Basic spellcheckers use the *Levenstein* distance (or *edit* distance) to detect the spelling errors. More advanced spellcheckers are using the probabilistic models to identify misspelled words [46].

2.1.4 Phrase Detection

The natural language consists of many phrases that are composed of multiple words. An example of a multi-word phrase is *"Information technology"*. While it is not exactly a mistake to interpret this entity as two tokens: *"Information"* and *"technology"*, it is clear that the individual words do not express the true nature of the phrase. Identifying the phrases in the input documents can significantly improve the overall performance as it brings a more in-depth insight on the text.

The approaches for the phrase detection are using token co-occurrence frequency [28]. Another naive, but efficient approach is to download an existing vocabulary of common phrases in the natural language and try to find such phrases in the input document collection.

2.1.5 Stopwords Removal

In the area of IR[3] and NLP, the term *stopword* [24] refers to a so commonly used word in a particular language that it does provide little to no information about the content of a document. For the English language an example *stopword* is the word *"the"*; knowing that some document contains such word does not provide any insight on the document. On the other hand, knowing that the document contains many occurrences of the word *"football"* undoubtedly tells us some information regarding the document content.

Keeping the stopwords in the text can confuse some of the methods and thus can lead to significantly worse performance results.

For most of the languages that researchers work with, a list containing *stopwords* for these particular languages exist. These lists are called *stoplists*, and the words from these lists are commonly removed from the texts to improve the performance and also to reduce the vocabulary size.

2.1.6 Stemming

The purpose of the stemming [24] step is to find the approximation of the root (*stem*) form of a word. The [20] defines the word *stem* as a part of the word that contains no inflectional morphology. The resulting entity of the stemming does not necessarily have to match the morphological root of the word nor even have to match any valid word from the language. The critical fact is that syntactically related words, plural and singular forms for example, are mapped to the same *stem*. This enables treating these words as synonyms and thus significantly reduces the overall vocabulary size.

The main advantage of this step is that the stemming algorithm does not require any other input than the words to process. The disadvantage is the fact that lexically related words are not guaranteed to be semantically related. The stemming algorithm needs to be carefully selected for the particular task since an inappropriate stemming algorithm can result in the overstemming problem where the words are confused as related while in fact they are not lexically related at all. Stemming is a language and task dependent, e.g., for the English language a stemming algorithm called *Porter stemmer* [36] exists. It is a rule-based stemming algorithm, and all of the rules were created manually.

[3]Information retrieval

2.1.7 Lemmatization

Lemmatization [24] is mapping inflected forms of a word to a single primal form (*lemma*). This process allows considering the inflected forms of words as the same entity. The difference between the lemmatization and stemming steps is that the lemmatization takes morphological information about the word into account. The *lemmas* for the words are usually obtained from a lexical database. Lemmatization step is done by a program called *lemmatizer*. The lemmatization step can also do the disambiguation. For example, the word `bank` can be either *noun* or *verb*, the *lemmatizer* would then differentiate between the different meanings and map these words to different *lemmas* (e.g., `bank_1`, and `bank_2`).

2.1.8 WordNet

WordNet [29] is a lexical database of English words capturing semantic relations between the words. *Nouns*, *verbs*, *adjectives*, and *adverbs* are grouped into cognitive sets of synonyms called *synsets*. *Synsets* are then linked through semantic and lexical relations. WordNet currently contains the information about $117,000$ English words.

2.2 Lexical-based Text Similarity

Naive and straightforward methods can measure the similarity between multiple documents at the lexical level. The lexical-similarity based methods which are relevant for this thesis are examined in the following subsections. Although the lexical-based text similarity does not take the semantic relationship between the entities into account, the results can still be sufficient for some of the use-cases.

Lexical-based similarity methods are widely used in *search engines*. Sometimes the lexical-based methods are used in more complex systems as additional features.

The main advantage of lexical-based similarity scoring methods is that they are typically not language dependent in most of the cases. This is because these methods – as said – do not take the semantic information about entities into account. However, they often provide worse results compared to more sophisticated methods that can capture the semantic information in the text.

2.2.1 Word Overlap

Calculating the document similarity as a lexical overlap present the most straightforward scoring method for comparing the documents at the lexical level. The similarity score is calculated as shown in the equation 2.1.

$$score_{d_1,d_2} = \frac{len(d_1 \cap d_2)}{max(len(d_1), len(d_2))} \tag{2.1}$$

Variables d_1, d_2 represent the token sequences of the compared documents and function *len* returns a length of a given sequence. The results of this methods can be significantly improved by enhancing the preprocessing phase – using stemming or lemmatization in particular.

2.2.2 Vector Space Models

Documents can be represented as vectors of floating point numbers in a n-dimensional space. Applying appropriate transformation on documents will result in a state where similar documents will have the lesser distance in this space than unrelated documents. This is frequently used in search engines where such distance between query and document is calculated to return documents most similar to a presented query.

An angle between the document vectors can be interpreted as a similarity of these documents. The lesser the angle between vectors, the more are documents considered as similar. To calculate the angle between the vectors a cosine similarity is used. This metric is calculated as shown in the equation 2.2.

$$cos(d_1, d_2) = \frac{d_1 \cdot d_2}{||d_1|| \cdot ||d_2||} \tag{2.2}$$

The figure 2.2 shows a visualization of the equation 2.2. By nature, the cosine similarity does not consider the actual distance of the vectors. For example, calculating the cosine similarity between the two vectors that are highly similar in most of the dimensions and different in a few would lead to the same results as calculating the cosine similarity between the two vectors that slightly differ in most of the dimensions. This can be targeted by combining the cosine similarity and some other metric, e.g. Euclidean distance [16].

While the cosine similarity is a universal method to measure the similarity between multiple documents represented in vector space, the other metrics could be more suitable for particular document representations, a Hellinger distance for example [10].

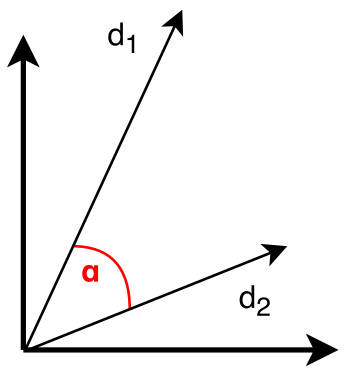

Figure 2.2: Vector space model

TF-IDF

TF-IDF [37] is a weighting scheme for documents represented in vector spaces and it was designed to capture the importance of the words regarding the input document collection. TF-IDF weighting translates the one-hot document vectors into a vector of floating point numbers. This weighting scheme was initially used in *search engines* as it easily allows to find a document with a query containing only its keywords (the most describing words for the document). The name of the method is derived from the two metrics that are used:

- *Term frequency* - term frequency in a particular document,
- *Inverse document frequency* - inversed count of documents in which the term occurs.

The resulting score is then calculated by multiplicating these two metrics.

Term frequency (TF) Term frequency is defined as a count of a word in a particular document. This raw frequency is often normalized by the length of a document which prevents overscoring longer documents. The normalized TF weight is calculated as shown in equation 2.3.

$$tf_{i,t} = \frac{f_{i,t}}{\sum_{j \epsilon T} f_{j,t}}, \tag{2.3}$$

the $f_{i,t}$ is the frequency of the i-th word in the document t. Many modifications of computing $tf_{i,t}$ exist, one of them is to use logarithmic weighting as described in [42].

Inverse document frequency (IDF) IDF [39] weight can be interpreted as the degree of uniqueness of the word, thus the importance of the word related to the whole document collection. The bigger the document count where the word occurs, the lesser the IDF weight is assigned. IDF weights for the words from the stoplists would result in scores near 0 value since they would appear in most of the documents; that is an example why stopwords are often filtered during the preprocessing phase. IDF weight is calculated as shown in the equation 2.4.

$$idf_i = \log \frac{N}{df(i)}, \tag{2.4}$$

the N is the size of the document collection and function $df(i)$ returns a document count in which the i-th word appears. Also, many modifications of estimation the IDF weight exist [33].

2.2.3 Keyword Extraction

In NLP, a *keyword* refers to a term that best describes the content of a document. A single document usually has a set of multiple keywords to be represented with. The automatic keyword extraction can significantly improve the performance of NLP systems as it can be used as an additional feature for measuring document similarity, or it can be helpful when solving the document summarization task.

A simple approach for the automatic keyword extraction task is to use TF-IDF weighting. TF-IDF weight is considered as the importance of a word in the document. Keywords are then selected as the words with the highest scores. This method, however, selects keywords that frequently appears in some document while not so frequently appearing in the rest of the collection, that does not correlate with the definition of the *keyword*, but this approach still provides sufficient results. It is quality, however, relies on the nature of the whole document collection.

The keyword extraction task is not related purely to the lexical-based approaches. We can extract the keywords using the semantic-based approaches. For example, the topic modeling approaches described in subsection 2.3.3 can be used for the keyword extraction. With the topic modeling, we can define a probability of the word in a particular document, that way we can consider the most probable words as keywords for the document. The word probability in a topic modeling is calculated by the equation 2.5.

$$P(w|d) = \sum_t P(w|t)P(t|d), \tag{2.5}$$

where w is the input word, d is a given document, and t is the particular topic. The combination of TF-IDF weight with the word probability might lead to better results when selecting the document keywords. The final keyword score would be then calculated as in the equation 2.6.

$$score(w) = P(w|d) \cdot tfidf(w, d). \tag{2.6}$$

More advanced methods are using term co-occurrence [25] and can work solely with a single document. However, such approaches are useful only on long continuous texts. On short texts, methods like TF-IDF weighting or topic modeling approaches are needed to be used since they do not require to compute the co-occurrence frequency.

2.3 Semantic-based Text Similarity

Estimating the semantic similarity at a document, sentence, or token level requires the use of more sophisticated methods. While the lexical-based approaches focus more on a syntactic relation of the compared entities, the semantic-based methods capture the deeper level of the entity relationships and thus providing better results in some cases.

In order to calculate a similarity metric between the two entities, a token-to-token similarity metric needs to be typically evaluated, this metric evaluates the semantic similarity between the two tokens. To capture the token-to-token similarity, a previously mentioned database called WordNet can be used or the similarity can be evaluated using the methods based on distributional hypothesis.

2.3.1 Distributional Hypothesis

In linguistics, the distributional hypothesis is a theory based on the insight of natural languages. This hypothesis is stated in many variations but all of them are equal. For example, [40] says that *"words which are similar in meaning occur in similar contexts"*. The [41] provides the multiple definitions of the distributional hypothesis.

Many methods for estimating the semantic similarity between the two entities are based on the distributional hypothesis. The distributional hypothesis plays a critical part when constructing *semantic embeddings* of the natural language entities.

2.3.2 Semantic Embeddings

A *semantic embedding* (sometimes called a *semantic vector*, or a *context vector*) is a representation of a natural language entity in a n-dimensional vector space. The semantic relationship between the particular entities is dependent on their *semantic embeddings* distance in an n-dimensional space. The semantic relationship and the distance of *semantic embeddings* are inversely proportional. The similarity between the embeddings is calculated using the cosine similarity metric described in the equation 2.2.

Word Embeddings

Semantic embeddings at the token level are called *word embeddings.* Typically, the *word embeddings* term is also used for embeddings of multi-word phrases since the phrase is often treated as a single token or word.

The less sophisticated methods for constructing the *word embeddings* are based on co-occurrence matrix [21, 23, 35], the more sophisticated methods are creating the embeddings using the neural networks (e.g., *Word2Vec* [28]).

The quality of the resulting embeddings is strongly dependent on the nature of the training document collection. The larger the and the more generic the document collection is, the more quality embeddings result.

Because the word embedding structure is universal (e.g, the word *queen* will most probably have similar embedding structure no matter the training document collection once it is large and universal enough), it is possible to download pre-trained embeddings that were created using a vast document collection. Using the pre-trained word embeddings spare researchers the need to construct the embeddings from the input document collection. Also, the pre-trained embeddings would – in most of the cases – provide a better insight on the semantic relationships between the entities because they are usually trained on a large document database and have highly optimized hyperparameters.

Sequence Embeddings

Semantic embeddings can also be evaluated at the sequence level, these embeddings are called *sequence embeddings* (also *sentence embeddings*, or *document embeddings* depending on the input sequence nature).

When computing the *sequence embeddings*, the *word embeddings* are typically have to be created first. Most of the methods for computing the *sequence embeddings* are based on the manipulation with the *word embeddings* of the words from the sequence.

The most basic and naive approach is to construct the *sequence embedding* as an average vector of corresponding *word embeddings* for the words that are contained in that particular sequence. It is clear that this method does not reflect the nature of natural language since not all words are equally important for the meaning of a sequence. This method can be combined with TF-IDF weighting in order to reflect the fact that some words contribute to semantic meaning more than others. The resulting *sequence embedding* is then constructed as a weighted average of corresponding *word embeddings* using the weights based on TF-IDF scores.

The approach described in the previous paragraph is not very useful for long sequences as the averaging introduces a significant error which would increase with the number of words in the text. More advanced methods for computing the *sequence embeddings* are described in [22] (*Doc2Vec* method derived from the *Word2Vec* method), or in [10].

2.3.3 Topic Modeling

In the context of NLP, the topic modeling refer to statistical approaches for uncovering the hidden topics in the document collection. A topic can be interpreted as a cluster of related words. To calculate the semantic relation of the words, the topic modeling approaches are using the distributional semantics. For example, the words: `NHL`, `break`, and `goal` are likely to be put on the same topic.

In topic models, a document is represented as an n-dimensional vector of floating point numbers where n is the fixed number of hidden topics to uncover. The i-th element of a resulting vector represents the probability of how likely is the i-th topic included in the particular document. The result is then a probabilistic distribution of topics. Estimating the document similarity with use of topic modeling is based on the premise that the similar documents are likely to contain proportionally the same topics.

In topic modeling, a cosine similarity metric is usually replaced with more suitable ones. To measure the similarity between the two probabilistic distributions a metric called *Hellinger distance* exists. The *Hellinger distance* metric is calculated as shown in the equation 2.7.

$$H(d_1, d_2) = \frac{1}{\sqrt{2}} \left\| \sqrt{d_1} - \sqrt{d_2} \right\|_2, \tag{2.7}$$

where the d_1 and d_2 are topic distributions of the compared documents.

LDA

LDA[4] [6] is a generative probabilistic model for document topics. In LDA, each document is modeled as a mixture of various latent topics where each topic is represented by a set of words. In theory, the topic distribution within the document is presumed to have a sparse *Dirichlet prior*. In other words, it is expected that the document will cover only the small set of topics and each topic will contain only a small set of words. LDA assumes the following generative process; for each document $D_m \epsilon D$:

- Choose $\Theta \sim Dirichlet(\alpha)$,
- Choose $\varphi_k \sim Dirichlet(\beta)$ for each topic K.
- For each word position i:
  - Choose a topic $z_i \sim Multinomial(\Theta)$.
  - Choose a word $w_i \sim Multinomial(\varphi_{z_i})$.

When representing the document, LDA forms a vector of floating point numbers in a K-dimensional space where the K is the number of hidden topics. The resulting vector reflects topic distribution in the document.

The training process of the LDA model is an unsupervised iterative process during which words are assigned to one of the K topics. Hyperparameters K, α, and β are tunable. Parameter α controls the topic density per document. A high α value means that documents are more likely to be a mixture of most of the topics, a low α value means that documents are related to just a few topics. Parameter β controls the per topic word distribution, setting β to a high value means that document is likely to contain most of the topic words, analogically for the low β value.

2.4 Neural Networks Approach

Models based on neural networks (ANNs[5]) are widely used for a various number of tasks. In computer science, a neural network is a mathematical apparatus inspired by the structures of biological neural networks. Models based on neural networks are very useful in pattern recognition, clustering, or classification. For this thesis, RNN[6] models are relevant and are described in the following subsections.

[4]Latent Dirichlet Allocation
[5]Artificial Neural Networks
[6]Recurrent Neural Network

2.4.1 Recurrent Neural Networks

RNN models are the particular case of ANN models. In RNN output from a single layer is passed to an identical copy of the same layer, using this approach enables the RNN models to take the input history into account when processing sequences, or lists. RNN models are widely used in NLP tasks since the processing of sequences is the essence of natural languages. Figure 2.3 shows the principle of RNN models.

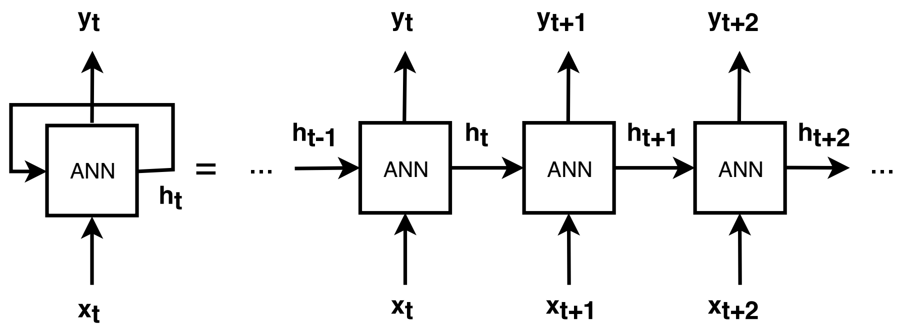

Figure 2.3: RNN model

A representative example of RNN models usage would be predicting the next video frame based on the current one. RNN models would be more effective compared to classic ANN models since the RNN models would be able to take the frame history (that led to current frame) into account.

Long Short Term Memory

LSTM[7] [14] is a type of RNN model. LSTM was designed to target the *Long-Term Dependency problem* [4] meaning that their default feature is the ability to remember information for a long period of time.

Essential building blocks for LSTM models are the *cells* and *gates*. In RNN, a *cell* refers to a single layer of the model, and the *gate* is a component that is responsible for the information flow through the network. LSTM network consists of three gates within a single cell. The architecture of LSTM *cells* is described through the five following equations:

$$\begin{aligned}
f_t &= \sigma(W_f x_t + U_f h_{t-1} + b_f), \\
i_t &= \sigma(W_i x_t + U_i h_{t-1} + b_i), \\
o_t &= \sigma(W_o x_t + U_o h_{t-1} + b_o), \\
c_t &= f_t \circ c_{t-1} + i_t \circ \tanh(W_c x_t + U_c h_{t-1} + b_c), \\
h_t &= o_t \circ \tanh(c_t).
\end{aligned} \tag{2.8}$$

[7]Long Short Term Memory

The variable x_t is the input vector in time t to the LSTM cell. Next, the f_t is the *forget gate*, this unit decides for how much information will be removed from the cell state. The i_t is called the *input gate* and controls how much information to remove from the *cell* input. The variable i_t is the *output gate*; this gate controls how much information will be removed from the output activation of the LSTM cell. The c_t is called the *cell state* and h_t is the *cell* output vector. The matrices W, and U are the weight matrices which are to be updated during the training process, the b is the bias vector and is also updated during the training process. Each vector h_t is a *sequence embedding* of the previous sequence of inputs. Many variations of LSTM models have been developed [12, 15].

Figure (2.4) shows the architecture of LSTM *cells*. It also provides a visual explanation of mathematical description in equation (2.8).

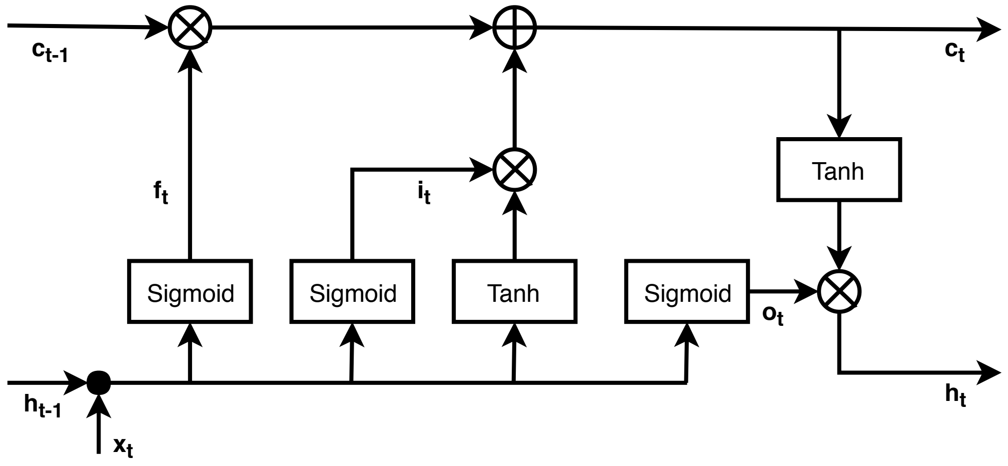

Figure 2.4: LSTM cells architecture

2.4.2 Stacking RNNs

When vertically stacking RNNs, the models that are not on the top of the stack must pass the aligned sequence of y_t states to the successor as shown in the figure (2.5). On this topic, the [34] says that *"building a deep RNN by stacking multiple recurrent hidden states on top of each other. This approach potentially allows the hidden state at each level to operate at different timescale"*. The stacked RNNs are used in speech recognition task [13].

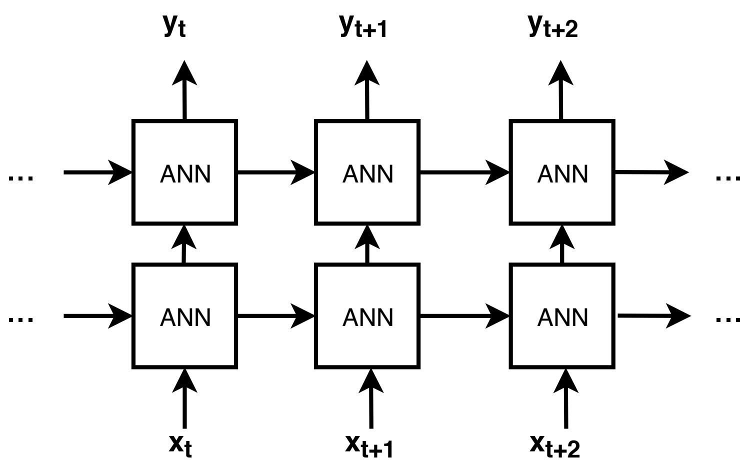

Figure 2.5: Stacked RNNs

2.4.3 Bi-directional RNNs

Bi-directional RNNs [43] were proposed to allow for the RNN to reach the future information from the current state. To do that, the Bi-directional RNNs are trained simultaneously in the positive and negative time direction.

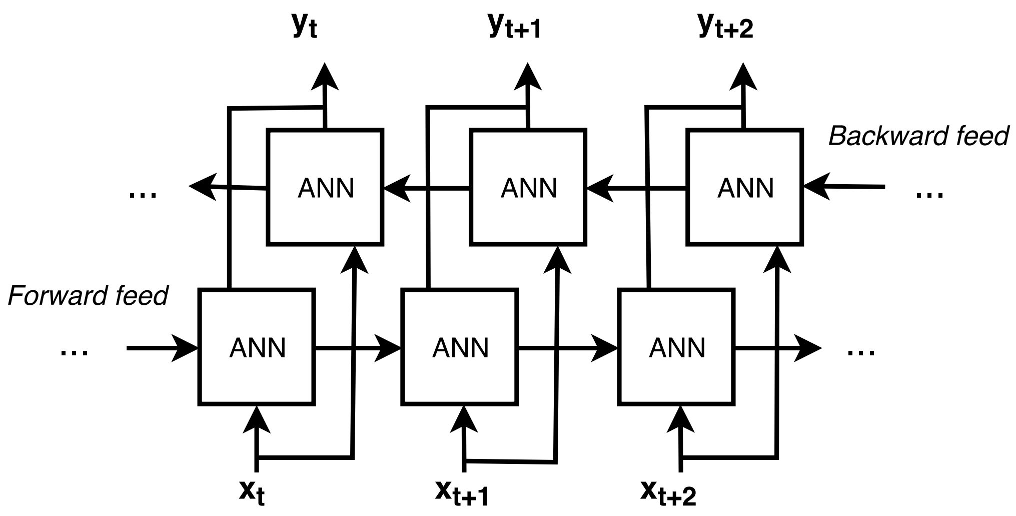

Figure 2.6: Bi-directional RNN

Bi-directional RNNs are used in e.g., NER[8], or automatic translation. In some cases, the Bi-directional RNN models do not provide better results compared to stacked classic RNN models; in NLP tasks, this could be because the natural language is composed in the way that previous information is further extended.

[8]Named entity recognition

2.4.4 Siamese Networks

Siamese networks are a special class of ANNs. The core design is that the siamese ANNs are composed of multiple copies of identical subnetworks. These subnetworks have the exact same hyperparameters set plus they share the weight matrices meaning that updating one of the subnetworks updates all of the subnetworks.

The architecture of siamese ANN is commonly used for the tasks where the task nature consists of determining a relationship between the multiple entities. In NLP the siamese ANNs are used e.g. in STS[9] task which is the task related to *Course Articulation* task. The figure (2.7) shows an example siamese ANN architecture.

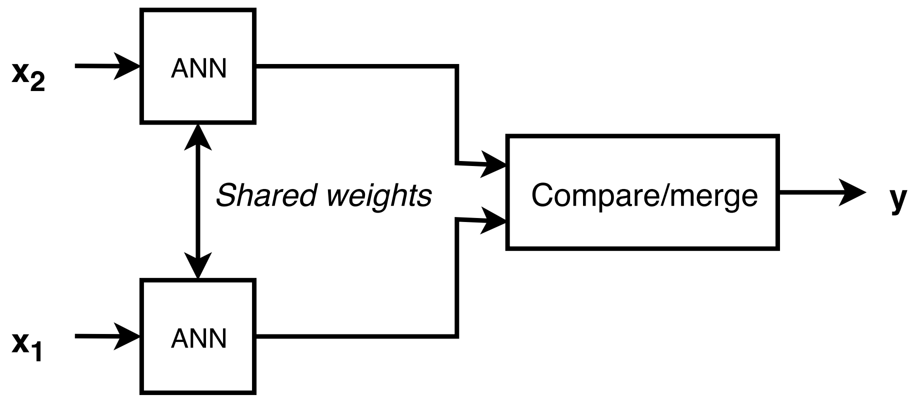

Figure 2.7: Siamese network architecture

2.5 Course Articulation

The primary topic of this thesis is to solve *Course Articulation* task as introduced in section 1.1. Transferability decision is a process that many institutions have adopted [31, 32, 47]. However, most of the current institutions use a large searchable database (up to $300,000$ transfers [19]) of manually annotated transfers. The current state is that the transferability decision is a manual process. These databases, however, typically do not include the negative results on the transferability decision. The goal of this thesis is to analyze and extend an existing system for automatic transferability decision which allows to classify previously unseen courses.

[9]Semantic Textual Similarity

2.5.1 Transferability Attributes

In order to create a course transferability database, a large course database must be collected first. Courses have many attributes that are taken into account whether the credits of this course are transferable for some other course. The basic list of course attributes is the following:

- *Institution*,
- *Department*,
- *Name*,
- *Description*,
- *Credits*,
- *Difficulty*,
- *Prerequisites.*

For the manual decision on the credit transferability, a person with knowledge of the course topics and also a person with experience of lecturing those particular topics is likely to be included.

When annotating the credit transferability, the annotators must take into account as much as possible attributes of both of the courses. This is time-consuming work and mostly also a reason why creating a large transferability database is a challenging task.

2.5.2 Current Approaches

Universities and colleges have adopted their internal procedures when deciding the course credit transferability. Currently, a universal database of the course transfers does not exist because universities and colleges have a different course model typically. This means that the course structure along with prerequisites can be fundamentally different to create such a universal database for the credit transferability.

While services providing an exploration for the course transferability exists, they are more like a course database with manually annotated credit transferability decisions. An example of credit transferability exploration system is *Transferology* [9]. To maintain the transferability database, *Transferology* tries to involve universities as well as lecturers. The results and decisions from universal systems like *Transferology* are not binding, and the universities do not have to respect the results from these systems.

2.6 Performance Measurement

Systems for *Course Articulation* or STS task are likely to be measured using multiple evaluation metrics. Using the evaluation metrics, the systems can be then compared to each other with regarding their quality. For this thesis, a few evaluation metrics are relevant: *accuracy*, *sensitivity*, and *specificity*.

2.6.1 Evaluation Metrics

This subsection describes the relevant evaluation metrics that are used in this thesis. All of the presented metrics are based on confusion matrix which is in table 2.1.

Actuall	*Predicted*	
	Positive	**Negative**
Positive	True positive (TP)	False negative (FN)
Negative	True negative (TN)	False positive (FP)

Table 2.1: Confusion matrix

Accuracy

The *accuracy* [2] is defined as in equation 2.9. It stands for overall classification accuracy on all of the classes.

$$accuracy = \frac{TP + FN}{TP + FP + FN + TN} \tag{2.9}$$

Sensitivity

In binary classification, the *sensitivity* [2] (*recall*) is defined as the overall true-positive rate. The equation 2.10 defines the *sensitivity* calculation.

$$sensitivity = \frac{TP}{TP + FN} \tag{2.10}$$

Specificity

The *specificity* [2] is defined as the overall false-positive rate of classification and is defined by the equation 2.11.

$$specificity = \frac{TN}{TN + FP} \tag{2.11}$$

3 Existing System Description

As introduced in section 1.1, the UWB have developed an automatic system for the *Course Articulation* task. During the system development and research, the UWB team has also created a course transferability database. This chapter describes both the created transferability database and the architecture of the developed system.

3.1 Transferability Database

To build a transferability database, we need the details about the courses. During the research, the UWB have collected a database containing the course details from many institutions. The database also contains the details about the institutions as well as the details the particular courses (see table 3.1 for statistics). Note that not all of the courses have all of the attributes. For example, not all courses have the credit information since such information was not always available.

Institutions	39
Courses	179, 113
Prerequisites	72, 848

Table 3.1: Course database statistics [45]

The course prerequisites were identified by applying NLP techniques on the descriptive attributes of the courses. The UWB says that they have used NLP techniques such as ER[1] to match course prerequisites [45]. Also, the SRT[2] was developed to find the corresponding syllabi online for each known course. The transferability records are the official transferability decisions of institutions. Each transferability record consists of three attributes:

- Course ID (transferring from),
- Course ID (transferring to),

[1]Entity recognition
[2]Syllabi retreival tool

- Transferability decision (*"YES"* or *"NO"*).

Both of the courses within a single transferability record come from different institutions. Only the *transferable* pairs have been downloaded from the official institution databases (the *non-transferable* records are not available as transferability pairs). Because of this, the *non-transferrable* pairs were artificially created as described in [45]. The table 3.2 describes the resulting course transferability database.

Institution	*Positive*	*Negative*
Michigan State University	1,268	26,063
Purdue University	329	0
University of Alabama	913	0
University of Maryland	2,078	3,588
University of Virginia	27	30
University of Washington	127	125
University of Wisconsin-Eau Claire	1,730	26
University of Wisconsin-Madison	2,232	2,128
University of Wisconsin-Milwauke	1,654	0
University of Wisconsin-Whitewater	1,598	4
Other	3,642	26,824
Total occurrence	15,598	58,788
Total transfers	7,895	29,301

Table 3.2: Transferrability database statistics [45]

3.1.1 Dataset Structure

The collected dataset is stored in a PostgreSQL [30] relational database. The database consists of the following 7 tables:

- `dd_institution` - information about particular institutions,
- `dd_course` - course attributes,
- `dd_syllabus` - syllabi details retreived with the SRT,
- `dd_transfer` - transferability records,
- `course_prerequisity` - links to course prerequisites,
- `course_transfer_from` - source transferability courses,

- `course_transfer_to` - target transferability courses.

The figure (3.1) visualizes the ERA[3] model for the created data collection.

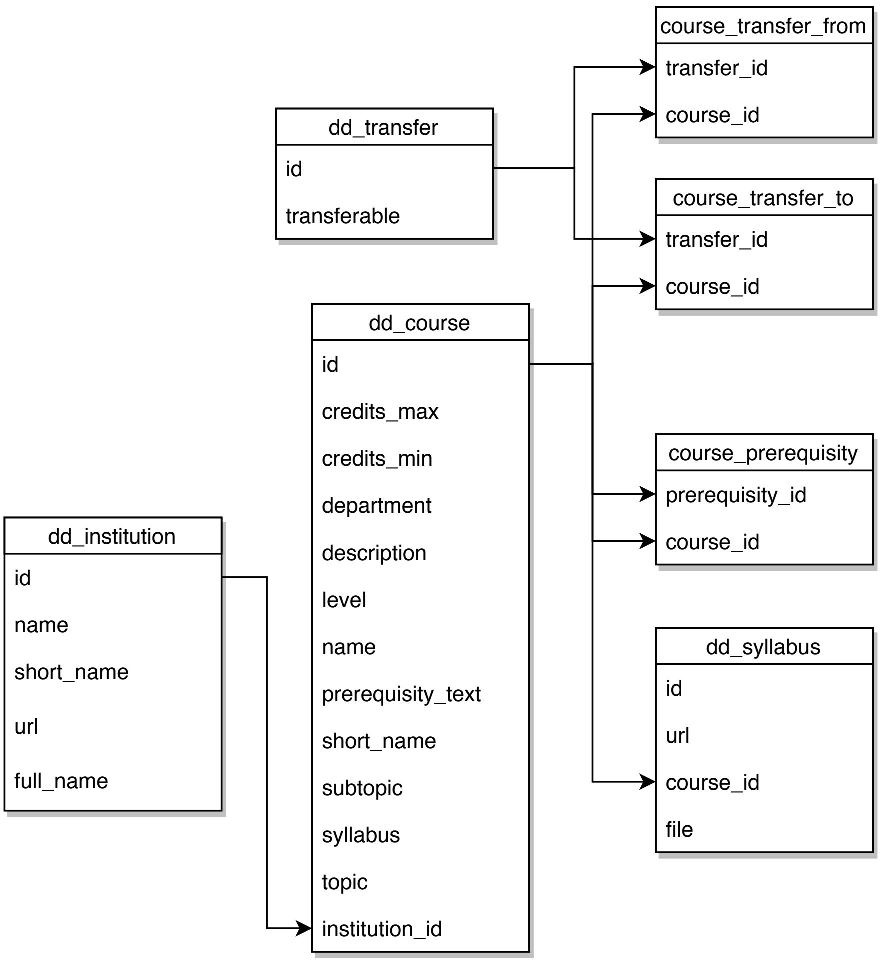

Figure 3.1: Dataset ERA model

3.2 Model Description

This section presents the UWB transferability decision model for the *Course Articulation* task. In particular, the section describes the employed feature set as well as the used classification algorithm.

The UWB model treats the *Course Articulation* task as the STS task meaning that the transferability decision is based on the semantic similarity of the descriptive course attributes – the `title` and `description`.

[3]Entity Relation Attribute

The [45] sees the transferability decision as a binary decision that takes the two courses and returns either *"YES"* or *"NO"* decision regarding the actuall course pair transferability. The UWB model does not take the asymmetric nature of the course pair transferability into account; this means that once the pair $A \rightarrow B$ is classified as transferable, the pair $B \rightarrow A$ must also be classified as transferable.

3.2.1 Feature Set

For the transferability decision, the UWB system employs the following set of features combined:

- The cosine similarity score between the topic probabilities from the LDA model trained to recognize 400 topics.
- The cosine similarity score between the vectors of TF-IDF scores for the documents.
- A number of words shared by courses in the `title` and `description` course attributes.
- A number of keywords shared by courses in the `title` and `description` course attributes.
- The cosine similarity between the vectors containing keyword probabilities for both of the courses.

The LDA and TF-IDF models are trained on the all courses from the created course database consisting of $179,113$ courses. The model uses the *Maximum Entropy* [5] classifier for the course pair transferability prediction.

3.2.2 Keyword extraction process

Keywords extraction presents a challenge when it comes to course description and title. The [45] says that the average number of words in a course description is about 30 words. Thus, for the keyword extraction, the UWB computes the word scores using both of the LDA and TF-IDF models as described in section 2.2.3. The words with highest scores are then selected as the keywords.

4 System Improvement

The purpose of this chapter is to analyze the UWB system and describe the improvements that have been done in this thesis. The first part contains the analysis and justification of the particular selected improvements. Second, we describe the technical details of some of the implemented improvements.

4.1 Problem Analysis

This section contains the analysis and decision steps that led to the selection of the particular implementations of the UWB system improvements. We choose to reimplement the system to set a baseline and to verify the performance results. To increase the original system performance, we have decided to implement new functionality at different levels of the system. Additionally, we have decided to create an alternative classifier for the course transferability. For that, we use ANN models.

4.1.1 System Reimplementation

The original system developed by the UWB is written in the Java [3] programming language; this is not suitable for us since one of our ideas is to use a neural networks approach to solve the *Course Articulation* task.

While there exists a deep learning library for Java – *Deeplearning4j* [44], we have decided that the Python language is the best fit for this work. One of the reasons for selecting the Python is the fact that most of the development and research in the area of ML[1] is currently done using this language, another reason is that we can use the deep learning libraries *Tensorflow* [1] and *Keras* [8] which are written in Python and currently tend to be one of the best libraries for the building ANN models.

4.1.2 Preprocessing Phase Improvement

The preprocessing phase in the UWB system is pretty straightforward and using only lowercasing, tokenization, and *stopwords* removal steps. There are many ways to improve the preprocessing phase. The proposed enhancements are analyzed and described further in this subsection.

[1]Machine learning

Stemming

Initially, we wanted to employ the stemmer in the preprocessing phase, but we have dropped this intention as we have decided that it would not bring any benefit. The reason for not including the stemming step into the preprocessing phase is that we aim to use pre-trained *word embeddings* for implementation of the semantic-based features (subsection 4.1.3) and the vocabulary of the embeddings models does not include stems for the words.

If we have used the stemming, we would have to train the *word embeddings* for ourselves, and we could not benefit from using a higher quality pre-trained embeddings.

Spellchecking

After manually browsing the course database, we have noticed that many courses have spelling typos in their descriptive attributes. Thus, we have decided to employ a spellchecker in the preprocessing phase. This brings two benefits; first, it will reduce the overall vocabulary size and second, it will map the misspelled words to a single entity and thus increase the performance of the employed features.

Implementation of a complex and sophisticated spellchecker would be a time-consuming work and since the spellchecking is just one of many considered enhancements to the UWB system, we have decided to implement a unigram-based spell checked using the *Damerau-Levenstein distance* [11]. This distance metric is based on a classical *Levenstein distance* (or *edit distance*) but it allows a transposition as the new edit operation. The transposition of characters is a very common case when making a spelling mistakes and adding a transposition as a possible spelling mistake would catch a greater set of spelling errors.

Phrase Detection

Extracting the multi-word expressions from the courses could improve the system performance. The preprocessing phase of the UWB system does not extract phrases in any way.

One of the possible approaches is to apply a phrase extracting algorithm for the available course database to extract phrases based on word co-occurrence. However, as mentioned in the previous subsection, we plan to use pre-trained *word embeddings* and it is essential to have the phrase embeddings available. Extracting the phrases by employing some sophisticated phrase extraction algorithm could result in the necessity to train the embeddings for the extracted phrases.

To target this, we use the vocabulary of pre-trained *word embeddings* that are containing phrases in the vocabulary. We download the 300-dimensional *Word2Vec embeddings* trained on about 100 billion of words from the *Google News* dataset[2]. The downloaded dataset contains $3,000,000$ *word embeddings* for the English words and phrases.

Using this approach – and not implementing any phrase extraction algorithm – have two benefits, first, we do not have to employ any sophisticated algorithm for phrase detection and second, we have the *word embeddings* for the phrases already trained.

4.1.3 Feature Set Extension

The features of the UWB model are mostly lexical-based, adding semantic-based features could significantly improve the overall system performance. We have decided to try to improve the UWB system by adding new features based on the *sequence embeddings*. In the implemented features, we will use the same *word embeddings* as for phrase extraction. This part is critical because we need the embeddings available for the phrases.

First, we train the *Doc2Vec* model. We have decided for the *Doc2Vec* because it is based on the same architecture as the *Word2Vec* (which we also use and which has excellent results for the word level embeddings) but it works on a sequence level.

Next, we want to experiment with another less sophisticated methods. We use the weighted *word embeddings* averaging by the TF-IDF weights. This approach is described in section 2.3.2.

4.1.4 Alternative Classifier

This thesis treats the *Course Articulation* task as the STS task, for this task ANN models are commonly used and are very efficient. Because of that, we have decided to create an ANN-based model as an alternative course pair transferability classifier.

The essence of the STS task is to compare two textual entities. We have decided to experiment with ANN models based on the siamese architecture (for siamese networks, see section 2.4.4) since the siamese networks were originally designed specifically for the comparison of multiple entities. We expect that this approach may significantly overcome the UWB system results.

[2]`https://code.google.com/archive/p/word2vec/`

Preprocessing for ANN Models

The preprocessing phase used in the UWB system is convenient for lexical-based features. For the alternative classifier using an ANN approach, we need to use different preprocessing steps. In particular, we do not remove the interpunction and the stopwords.

We keep the interpunction in the text because it still carries the semantic insight of the text and the RNN models can benefit from that. We identify interpunction as a separate token, but we do keep the interpunction joining the words (e.g., *”machine-learning”* is kept as a single token). Interpunction is removed in the UWB model preprocessing pipeline because the employed features in the UWB model are lexically-based and such features cannot capture the information provided by the interpunction.

Also, the *stopword* removal step is more of a technique related to the IR. Keeping the *stopwords* in the text can confuse the lexically-based features, but the semantic-based approaches (especially RNN based models) can have better results without removing the *stopwords*.

Figure 4.1 shows the difference between the UWB system and out modified preprocessing phases outputs. The first example shows the output of preprocessing phase used in the UWB system, and the second shows the output of preprocessing phase for ANN models.

"We had too many fumbles; we lost the game"

→ [*"fumbles", "lost", "game"*]

(a) Preprocessing result for the UWB model

"We had too many fumbles; we lost the game"

→ [*"We", "had", "too", "many", "fumbles", ";" ,"we", "lost", "the", "game"*]

(b) Preprocessing result for our ANN model

Figure 4.1: Difference between preprocessing phases

4.2 System Reimplementation

The UWB system was developed using several libraries that are not available for Python. The used libraries are *Brainy* [17] as the machine learning library and the *Mallet* library [26] for the LDA models. The lack of support for these libraries in Python was initially an issue as we needed to select suitable replacements for these libraries. As a result, we have decided to implement our own mechanism for operating with features and models, this mechanism is strongly inspired by the *Brainy* library. For the NLP models, we have used the *Gensim* [38] library. The *Gensim* also provides an interface for the models to work with the *word embeddings* and *sequence embeddings*.

The Python codebase developed during the work on this thesis could be used as the core for the *Course Articulation* systems, it allows building new models that are based on feature embeddings as well as creating the models for this task from scratch. We have recreated the UWB model using the developed mechanism for the features and the *Gensim* library.

4.3 Implemented Extensions

This section describes the technical details of some of the implemented enhancements and features. The subsection 4.3.3 describes the architecture details of created alternative classifier.

4.3.1 Spellchecking

The implemented spellchecker builds the vocabulary of the words by analyzing a large document collection given on input. Also, each word in the vocabulary has also stored occurrence frequency. This allows selecting the most probable word in the case where multiple corrections are available by selecting the most probable word based on the stored occurrence frequency.

Once the spellchecker builds the vocabulary, it works in the following way. The algorithm takes the word W on the input and if W is in the stored vocabulary, the algorithm returns it without any modification. If W is not found in the vocabulary, the spellchecker generates a set of words S where each $W' \epsilon S$ has the *Damerau-Levenstein distance* equal to 1. Then, the most frequent word in the vocabulary that matches any word from S is returned. If none found, the spellchecker returns the input word W unmodified.

4.3.2 Phrase Detection

To detect phrases, we create the word bigrams on the input text and we check whether the vocabulary of the downloaded *word embeddings* contains any of the created bigrams. If we find a match, we join to corresponding words in the input text to a phrase.

We do not extract trigram or higher order n-gram phrases because such phrases in the downloaded *word embeddings* vocabulary are mostly not common phrases from the natural language but only a statistically frequently co-occurring words (e.g. a famous person names, or names of places).

4.3.3 ANN-based Classifier

In this thesis, we implement an alternative classifier for the course transferability decision. The classifier is based on the siamese ANN architecture. As described in subsection 4.1.4, we modify the preprocessing steps used for the alternative classifier.

Model Description

The implemented model consists of the two subnetworks where each one accepts one course from the transferability pair, the result of the siamese subnetworks is then processed and the output of the model is a *"YES"* or *"NO"* decision regarding the course pair transferability.

We set the model hyperparameters experimentally. The siamese subnetworks consist of 300-dimensional `Embeddings` layer and two 300-dimensional `LSTM` layers stacked on top of each other (for details on stacking RNNs, see subsection 2.4.2). Both of the stacked `LSTM` layers are using a 30% recurrent and output dropout. The output of the siamese subnetwork is interpreted as the encoded course information. The figure 4.2 visualizes the architecture of the siamese subnetworks.

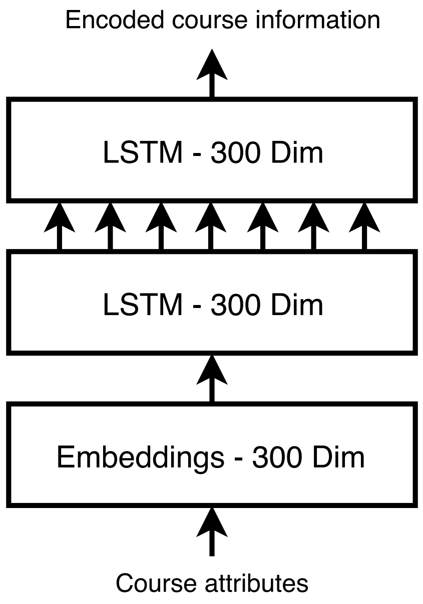

Figure 4.2: Implmeneted siamese subnetworks

The output of both of the siamese subnetworks is concatenated and sent to the two 900-dimensional densely connected layers with *RELU* activation stacked on top of each other. We add the 40% and 30% dropouts between the layers with *RELU* activation function.

The output from the densely connected layers is then sent to the 2-dimensional densely connected layer with *Softmax* activation. We choose the output number of neurons equal to the classification classes as it enables us to get scores per class and thus enables us to get more insight of the training and classification process.

The complete model is visualized in figure 4.3. All of the layers are trainable except the `Embeddings` layers. The model has total of $2,796,002$ parameters.

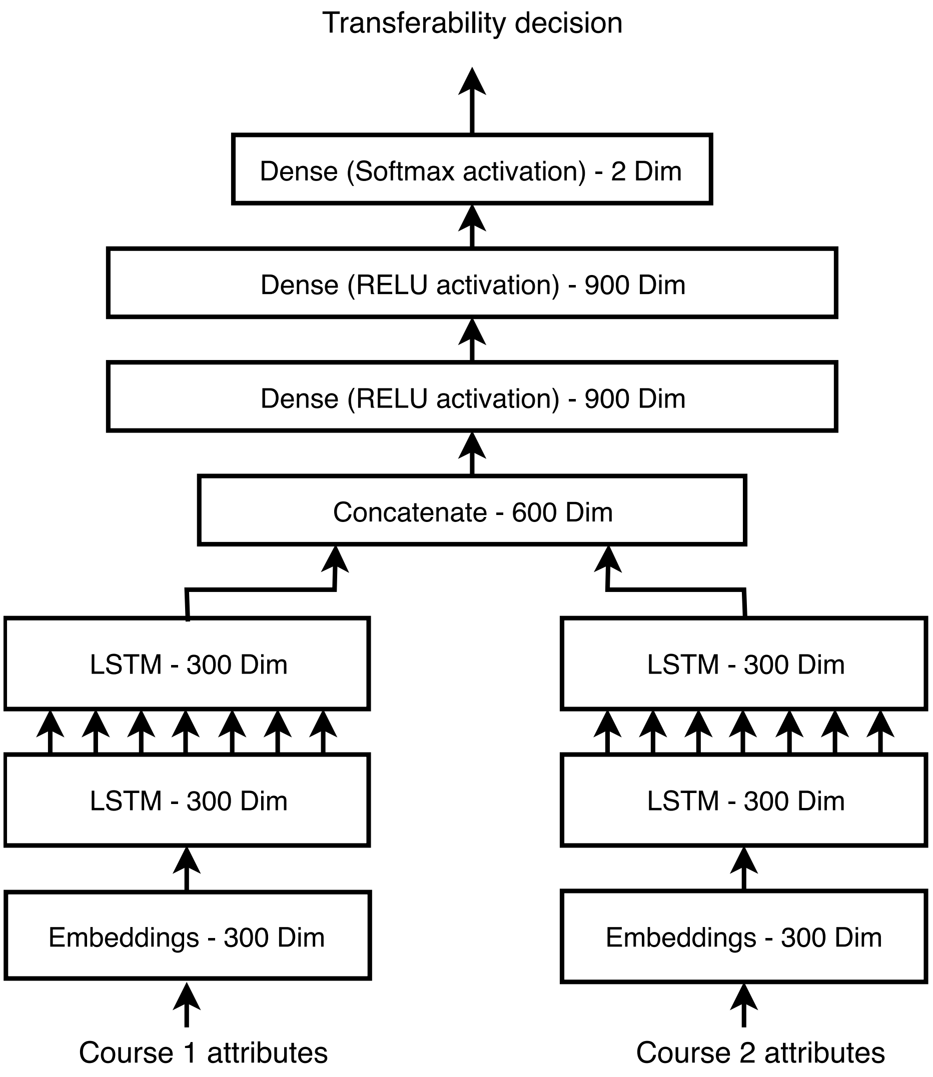

Figure 4.3: Architecture of created model

5 Experimental Results

In this chapter, we present the experimental results of the improvements that we have made to the UWB system. Section 5.1 describes the designed experiments, the results are then presented further in the chapter.

5.1 Designed Experiments

We have designed the series of experiments to validate the improvements to the UWB system. The first experiment validates the reimplementation of the existing system and sets a baseline for all of the following experiments. All other experiments are compared to the baseline and not to the original results of the existing system. The list of experiments is the following:

1. Validate the reimplementation of the existing system (section 5.4.2).
2. Evaluate the performance of the reimplemented system with improved preprocessing phase (section 5.5).
3. Evaluate the performance of the reimplemented system with semantic-based similarity features (section 5.6).
4. Evaluate the performance of the ANN-based classifier on the existing course transferability database (section 5.7.1).
5. Validate the created alternative classifier on a similar task (RTE[1] task using the SNLI[2] dataset [7]) (section 5.7.2).

The purpose of the last experiment is to check that the created alternative classifier provides meaningful results on the related task and that the results on the existing course database are not caused by overfitting of the created model.

The presented results in experiments 1-4 are the results of a 10-fold cross-validation on the whole course database. The results of experiment 5 are results on the SNLI dataset and are compared with the results of other published models on this dataset. In section 5.2.2, we explain why we have chosen the SNLI dataset for the validation of our model.

[1]Recognizing textual entailment

[2]Stanford natural language inference

5.2 Data Collections

This section provides the description of the datasets used in the experiments.

5.2.1 Existing Course Database

The existing course database is used as described in section 3.1. No changes have been made to this dataset for the experiments.

5.2.2 SNLI Dataset

The SNLI dataset is a collection of manually labeled English sentences for balanced 3-class classification. The output classes are *entailment*, *neutral*, and *contradiction*. The SNLI dataset consists of $570,000$ sentences and was created to support the RTE task.

We choose the RTE task for the model validation because of the similarity with our approach to the *Course Articulation* task. The SNLI dataset was picked for our experiments because it provides the consistent collection of training, development, and testing data parts. Additionally, the authors of the SNLI dataset provide the summary of published results meaning that we can compare our model performance to other published models.

When creating the dataset, the annotators were shown only the caption of a photography without showing the actual picture. The annotators were then asked to write the alternative captions. The literal wording of the instructions for writing alternative captions [7] is the following:

- Write one alternate caption that is **definitely** a **true** description of the photo. Example: For the caption *"Two dogs are running through a field."* you could write *"There are animals outdoors."*

- Write one alternate caption that **might be** a **true** description of the photo. Example: For the caption *"Two dogs are running through a field."* you could write *"Some puppies are running to catch a stick."*

- Write one alternate caption that is **definitely** a **false** description of the photo. Example: For the caption *"Two dogs are running through a field."* you could write *"The pets are sitting on a couch."* This is different from the maybe correct category because it is impossible for the dogs to be both running and sitting.

5.3 Evaluation Metrics

The experiments are evaluated using the evaluation metrics described in section 2.6.1. Specifically, the evaluation metrics are the following:

- *Accuracy*
- *Sensitivity*
- *Specificity*

Because of imbalance of the transferability decisions in the existing course database, the simple and naive classifier that would classify all course pairs as *non-transferable* would achieve high *accuracy*. To target this, we do not measure the quality of the classifiers only by the *accuracy* metric, but we consider other metrics as well. That way, we can consider the classifier with lower *accuracy* as better than other classifier which has higher *accuracy*.

5.4 Setting Baseline

To verify the UWB model implementation and to set a baseline for our experiments, we reimplement the original UWB system into the Python programming language. The explanation of this step is in the chapter 4.

The codebase of the existing system provided by the UWB does not contain the training details for the used models such as LDA for recognizing 400 topics. Only the serialized versions of trained models are provided. For the reimplemented system, we have used a slightly different set of features for our reimplemented system.

We optimized the hyperparameters of the models only on the last fold of the cross-validation, the remaining 9 folds were the heldout data. We were able to extract hyperparameters and thus successfully train only the LDA on 100 topics. When we used our LDA trained to recognize 100 topics in the keyword extractor with the TF-IDF model, we got worse results than using TF-IDF alone. Because of this, we have decided to use only the TF-IDF model for keyword extraction.

5.4.1 Models Parameters

This section contains the training details for the used models during the reimplementation. The models are trained using the *Gensim* library. We mention only the parameters that we change from the default values that are used by the *Gensim* library.

LDA

We trained the LDA with the following parameters:

Topics	100
Iterations	1000
Alpha	50.0
Eta	0.1
Passes	5

5.4.2 System Reimplementation

The first experiment is to verify the reimplementation of the existing system and to set a baseline for other experiments. This subsection contains the results achieved by running the Java codebase given by the UWB and results from our reimplemented Python codebase.

Results

The results of the experiment are in the table 5.1.

	Accuracy	**Sensitivity**	**Specificity**
Java codebase	86.4%	52.1%	95.6%
Python codebase	86.7%	51.4%	96.1%

Table 5.1: Results of system reimplementation experiment

Discussion

We achieve almost the same results as the existing UWB system despite using a slightly modified feature set. We do not use the same feature set because we do not have the training details for some models (for an explanation, see subsection 5.4). Specifically, we use LDA recognizing 100 topics instead of LDA recognizing 400 topics and we did not use LDA as the part of the keyword extractor.

We explain the similar results despite different features by the fact that *Gensim* library implements some optimizations for the LDA and TF-IDF models. We can say that we have successfully reimplemented the existing UWB system and thus set a valid baseline for other implemented improvements.

5.5 Improved Preprocessing

In this section, we describe experiments that are related to the preprocessing phase of the UWB system. We present separate experiments measuring the performance gain by employing the spellchecking in the preprocessing phase and by extracting the phrases from the courses.

5.5.1 Spellchecking

In this experiment, we employ the spellchecker in preprocessing. We experimentally create the spellchecker vocabulary on the content of [24].

Results

After manually checking the spellchecker results, we have decided not to measure the system performance with the spellchecked courses. This was because the implemented spellchecker on the available course database resulted in many correction errors. We tried to reduce the error rate by modifying the *Damerau-Levenstein distance* to permit only the transpose operation, but still, a lot of words were corrected incorrectly (e.g., *"GPA"* was corrected to *"gap"*, or *"DNA"* was corrected to *"and"*).

Discussion

We did not measure the system performance with the spellchecker included. The unsatisfactory results are caused by the simplicity of the implemented spellchecker. We could target this issue by selecting a better dataset for creating a spellchecker vocabulary. However, the existing course database includes courses for various topics and domains, we would need to initialize the spellchecker vocabulary on a universal and large document collection, or we would need to employ a more sophisticated spellchecking algorithm. As the spellchecking is only one of the many improvements planned to implement, we have decided not to tune the spellchecker.

5.5.2 Phrase Detection

The second made experiment related to the preprocessing phase is adding the phrase extraction. The phrase extraction process is implemented as described in subsection 4.1.2.

The purpose of this experiment is to validate that identifying phrases in the descriptive attributes of the courses will result in better performance of the UWB system.

Results

Using the described approach, we have detected the total of 7786 phrases in the available course database. A few examples of the detected phrases are *"biological sciences"*, *"supercritical fluid"*, *"civil rights"*, or *"liquid fuels"*.

Results of the experiment are in table 5.2. For this experiment, we did not tune any of the hyperparameters.

	Accuracy	**Sensitivity**	**Specificity**
Baseline	86.7%	51.4%	96.1%
Baseline + phrases	86.4%	49.9%	96.1%

Table 5.2: Results of phrase extraction experiment

Discussion

Adding a phrase extraction step into the preprocessing phase of the system lead to worse results. While the overall system *accuracy* decreases by only the 0.3%, the *sensitivity* metric decreases by 1.5%. The *specificity* does not change. The fact that significant change of *sensitivity* metric does only slightly affect the overall system *accuracy* points at the imbalance of the classification classes.

We expected increased *accuracy* by adding the phrase extraction step. It is because the natural language contains some phrases and considering such phrases as a single entity better describes the meaning of the text. We explain the results of the experiment by the inappropriate hyperparameters of the used models. Optimizing hyperparameters on the modified preprocessing phase might lead to better results.

5.6 Semantic-based Features

In this section, we present the results of experiments with adding a *sequence embeddings* features to the feature set of the UWB model.

Also, because the phrases have more of semantic meaning, we run the experiments both with and without the phrase extraction in the preprocessing phase. We expect for the phrase extraction to provide better results when combined with the semantic-based features.

5.6.1 Doc2Vec Model

In the third made experiment, we train the *Doc2Vec* model on the available course database and use it as a feature in the existing feature set. We train the *Doc2Vec* model in *Gensim* with the following parameters:

Vector size	300
Window	10
Min. count	5
Epochs	10

Results

Results of the experiment are in table 5.3.

	Accuracy	**Sensitivity**	**Specificity**
Baseline (BL)	86.7%	51.4%	96.1%
BL + Doc2Vec	86.7%	51.3%	96.2%
BL + phrases + Doc2Vec	86.8%	51.8%	96.1%

Table 5.3: Results of experiments with *Doc2Vec* model

Discussion

Adding a *Doc2Vec* model to existing feature set without phrase detection does not significantly change the performance of the system. The overall system accuracy does not change, but the *sensitivity* decreases by 0.1% and *specificity* increases by 0.1%. Despite the consistent *accuracy*, we consider this system as providing worse results for two reasons; first, the *Doc2Vec* feature clearly could not capture the semantic relations better than existing feature set combined, and second, the *sensitivity* metric is slightly worse.

By combining a phrase extraction with the *Doc2Vec* model, the system provides better results than the *baseline*. Compared to the baseline, the *Doc2Vec* with phrase extraction increases the overall *accuracy* by 0.1% and increases *sensitivity* by 0.4%. The *specificity* remains unchanged. We explain better results by the fact that phrases provide more of a semantic insight on the text. The *Doc2Vec* model is a semantic-based feature and thus can benefit from having the phrases identified, the existing feature set could not do that since it consists of mostly lexically-based features.

5.6.2 Weighted Word2Vec Averaging

In the next experiment, we try a more straightforward method to measure the course similarity. In particular, we use a weighted *word embeddings* averaging using the TF-IDF weights as described in subsection 2.3.2.

Results

Results of the experiment are presented in table 5.4.

	Accuracy	**Sensitivity**	**Specificity**
Baseline (BL)	86.7%	51.4%	96.1%
BL + Doc2Vec	86.7%	51.3%	96.2%
BL + phrases + Doc2Vec	86.8%	51.8%	96.1%
BL + V2W avg	87.1%	52.6%	96.2%
BL + phrases + V2W avg	87.0%	51.8%	96.3%

Table 5.4: Results of experiments with *Word2Vec* averaging

Discussion

Applying the feature based on the weighted *Word2Vec* averaging without extracting the phrases in the preprocessing outperforms the previous experiments. The *Word2Vec* averaging increases *accuracy* by 0.4% compared to to the baseline and 0.3% compared to the *Doc2Vec* feature with phrase extraction.

On the other hand, adding a phrase extraction to the *Word2Vec* weighted average feature does not lead to better performance results as expected by the outcome of the previous experiments. With the phrase extraction added, the weighted *Word2Vec* averaging feature performs only slightly better than the *Doc2Vec* feature with phrase extraction step.

We explain the better results using the simple weighted *word embeddings* averaging without the phrase extraction by the fact that we have used a high quality pre-trained *word embeddings.* This simple method, however, was not able to capture the additional semantic information provided by phrase extraction step.

5.7 ANN-based Classifier

We have created an alternative classifier based on the ANNs. The created model is described in subsection 4.3.3. We have applied to model on the SNLI dataset to verify the model performance on the task related to our approach for the *Course Articulation* task.

The training details for the ANN-based classifier are described in the subsections related to the particular experiments.

5.7.1 Existing Course Database

The first experiment with the alternative classifier was done on the available course transferability database.

When training the model we monitor the validation *accuracy* metric to detect when the model starts overfitting, and we always use the best-trained model. We have trained the model using the following configuration:

Max. epochs	25
Batch size	32
Validation split	0.1
Optimizer	*ADAM*
Loss function	*categorical crossentropy*

Results

The results of the experiment are presented in table 5.5. Figures 5.1, and 5.2 shows the development of the *accuracy* and *loss* metrics through the training process (both of the metrics are averaged over all epochs).

	Accuracy	**Sensitivity**	**Specificity**
Baseline (BL)	86.7%	51.4%	96.1%
BL + phrases + Doc2Vec	86.8%	51.8%	96.1%
BL + V2W avg	87.1%	52.6%	96.2%
ANN classifier	98.3%	93.5%	99.5%
ANN classifier + phrases	98.3%	93.5%	99.4%

Table 5.5: Results of ANN-based model experiments

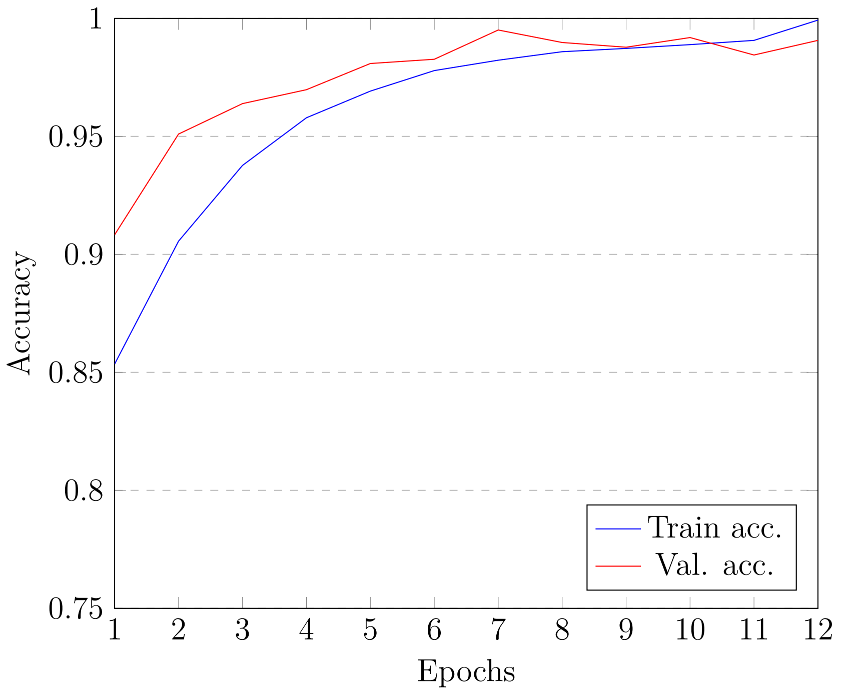

Figure 5.1: Training accuracy vs. Validation accuracy

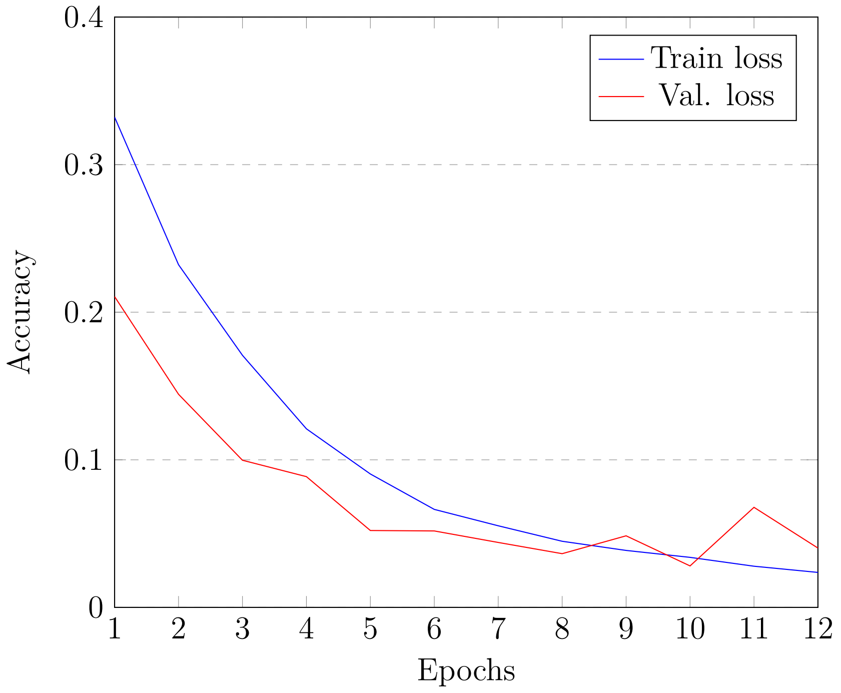

Figure 5.2: Training loss vs. Validation loss

Discussion

As expected, the ANN-based transferability classifier performs significantly better than the original UWB system and also outperforms all of the implemented improvements made to this system. We have tried to run the ANN classifier both with and without the phrase extraction, and the results remained the same for both cases.

The near-perfect results can be caused by the overfitting of the classifier. However, based on the figures 5.1 and 5.2, we can say that overfitting is the most probably not our case. We explain the results with the complexity of the ANN model, and thus that the model is able to capture much more information about the courses from the provided attributes than the lexical-based features employed in the UWB system.

5.7.2 SNLI Dataset

To verify the quality and meaningfulness of the created ANN-based transferability classifier, we have used the classifier for the RTE task on the SNLI dataset which is described in subsection 5.2.2.

When training the model we monitor the validation *accuracy* metric to detect when the model starts overfitting, and we always used the best-trained model. The model training have been done using the following configuration:

Max. epochs	50
Batch size	256
Optimizer	*ADAM*
Loss function	*categorical crossentropy*

Results

Table 5.6 shows our results on SNLI dataset compared to other published models. The last entry in table 5.6 are the best results from *sentence encoding* models. The complete table of published results on the SNLI dataset is available at `https://nlp.stanford.edu/projects/snli/`. We do not aim to achieve best results compared to other published models, the purpose of this experiment is to validate the quality of created ANN-based classifier.

Figures 5.3, and 5.4 visualizes the training process. Specifically, they show the development of the training *accuracy* and validation *accuracy* throughout the model training process.

Model (year of publication)	**Params**	**Train acc.**	**Test acc.**
⋮	⋮	⋮	⋮
100D LSTM encoders (15)	220k	84.8%	77.6%
Our implemented model	**2.8m**	**83.2%**	**81.9%**
300D CNN encoders (15)	3.5m	83.3%	82.1%
300D SPINN-PI encoders (16)	3.7m	89.2%	83.2%
⋮	⋮	⋮	⋮
300D Self-Att. Network (18)	3.1m	92.6%	86.3%

Table 5.6: Results on the SNLI dataset [7]

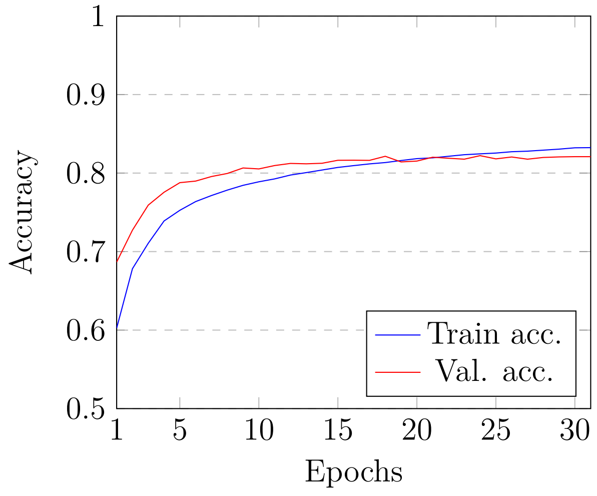

Figure 5.3: Training accuracy vs. Validation accuracy

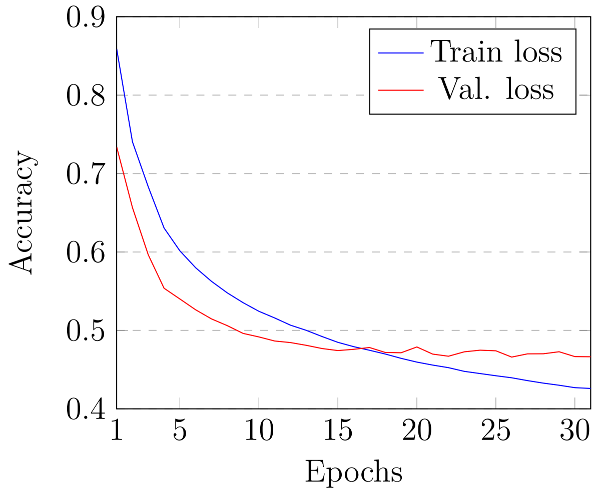

Figure 5.4: Training loss vs. Validation loss

Discussion

In this experiment, we use our ANN-based classifier to solve the RTE task using the SNLI dataset. The results have shown that our ANN-based model can solve the task related to the *Course Articulation* task. Although, the results on the SNLI dataset have shown that the ANN-based classifier does not match the state-of-the-art results.

By scoring results that are comparable to other models ensures us that our results on the existing course transferability database are not produced by overfitting the model.

6 Conclusion

In this chapter, we propose the possible improvements (section 6.1) to this thesis and we conclude the thesis (section 6.2).

6.1 Future Work

In this thesis, we have created a system for *Course Articulation* task. The future improvements to this work may include:

- Extend the existing course database and available course transferability records. When creating the transferability database, do not select the course pairs for annotation randomly, but employ a more sophisticated approaches to suggest the pairs for annotation – we believe the *Doc2Vec* model, or any other method based on *sequence embeddings* can be used for course pair suggestions.
- Develop a way how to reuse the course transferability predictions to improve the system. For example, select transferability predictions with the lowest confidence, annotate them and extend the training dataset with such course pairs.
- Employ more attributes of the courses when classifying the course pair transferability, if available. For example, consider the similarity of the prerequisites of the courses, credit information, etc.
- Target the possible asymmetricity of the transferability records. We believe that in order to achieve this, the descriptive attributes of the courses are not enough and more attributes would have to be considered when classifying the course pair transferability.
- Develop a more sophisticated spellchecking mechanism and use it in the preprocessing phase of the system.
- Create an online demo for the developed system. This would require to design and implement a graphical user interface for the web browser.

6.2 Conclusion

In this thesis, we have analyzed and improved the system for the *Course Articulation* task which was originally developed by the UWB. We have reimplemented the existing codebase from Java programming language to Python. Next, we have designed and implemented several improvements to the UWB system, and last, we have employed artificial neural network as an alternative classifier for the course transferability. In particular, the improvements to the UWB system consist of improving the preprocessing phase (*spellchecking* and *phrase extraction*), and implementing semantic-based features. The alternative classifier is based on siamese neural networks.

The reimplemented system provides comparable results when compared to the UWB system. Our experiments have shown that adding a phrase extraction to the UWB system does not lead to the performance improvement. On the other hand, our experiments have proved that combining phrase extraction with semantic-based features slightly improves the overall system *accuracy*. We explain such results with the fact that identifying phrases provides more of a semantic insight on the text and lexical-based features are not able to capture the additional information provided by the extracted phrases.

The results from the experiments with the created alternative classifier have shown, that results achieved by employing artificial neural networks significantly overcame the results achieved by the UWB. Specifically, the alternative classifier scores the 98.3% *accuracy* while the UWB system with our improvements scores only 87.1% *accuracy* on the existing course transferability database. To verify the created classifier, we have used it to solve the RTE task on the SNLI dataset where we achieved the *accuracy* of 81.9%.

List of Tables

List of Figures

Bibliography

[1] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. Tensorflow: A system for large-scale machine learning. In *OSDI*, volume 16, pages 265–283, 2016.

[2] Mehmet Fatih Akay. Support vector machines combined with feature selection for breast cancer diagnosis. *Expert systems with applications*, 36 (2):3240–3247, 2009.

[3] Ken Arnold, James Gosling, and David Holmes. *The Java programming language.* Addison Wesley Professional, 2005.

[4] Yoshua Bengio, Patrice Simard, and Paolo Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks*, 5(2):157–166, 1994.

[5] Adam L Berger, Vincent J Della Pietra, and Stephen A Della Pietra. A maximum entropy approach to natural language processing. *Computational linguistics*, 22(1):39–71, 1996.

[6] David M. Blei, Andrew Y. Ng, and Michael I. Jordan. Latent dirichlet allocation. *J. Mach. Learn. Res.*, 3:993–1022, March 2003. ISSN 1532-4435. URL `http://dl.acm.org/citation.cfm?id=944919.944937`.

[7] Samuel R Bowman, Gabor Angeli, Christopher Potts, and Christopher D Manning. A large annotated corpus for learning natural language inference. *arXiv preprint arXiv:1508.05326*, 2015.

[8] François Chollet et al. Keras. `https://keras.io`, 2015.

[9] Inc. College Source. Transferology, 2014. URL `https://www.transferology.com/`.

[10] Andrew M Dai, Christopher Olah, and Quoc V Le. Document embedding with paragraph vectors. *arXiv preprint arXiv:1507.07998*, 2015.

[11] Fred J Damerau. A technique for computer detection and correction of spelling errors. *Communications of the ACM*, 7(3):171–176, 1964.

[12] Felix A Gers and Jürgen Schmidhuber. Recurrent nets that time and count. In *Neural Networks, 2000. IJCNN 2000, Proceedings of the IEEE-INNS-ENNS International Joint Conference on*, volume 3, pages 189–194. IEEE, 2000.

[13] Alex Graves, Abdel-rahman Mohamed, and Geoffrey Hinton. Speech recognition with deep recurrent neural networks. In *Acoustics, speech and signal processing (icassp), 2013 ieee international conference on*, pages 6645–6649. IEEE, 2013.

[14] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[15] Nal Kalchbrenner, Ivo Danihelka, and Alex Graves. Grid long short-term memory. *arXiv preprint arXiv:1507.01526*, 2015.

[16] Tom Kenter and Maarten De Rijke. Short text similarity with word embeddings. In *Proceedings of the 24th ACM international on conference on information and knowledge management*, pages 1411–1420. ACM, 2015.

[17] Michal Konkol. Brainy: A machine learning library. In *International Conference on Artificial Intelligence and Soft Computing*, pages 490–499. Springer, 2014.

[18] Miloslav Konopík. Owen software - milestone 1 report. 2012.

[19] Miloslav Konopík. Owen software - milestone 2,3 report. 2013.

[20] P.R. Kroeger. *Analyzing Grammar: An Introduction.* Cambridge University Press, 2005. ISBN 9781139443517. URL `https://books.google.cz/books?id=rSglHbBaNyAC`.

[21] T.K. Landauer, P.W. Foltz, and D. Laham. An introduction to latent semantic analysis. *Discourse processes*, 25:259–284, 1998.

[22] Quoc Le and Tomas Mikolov. Distributed representations of sentences and documents. In *International Conference on Machine Learning*, pages 1188–1196, 2014.

[23] K. Lund and C. Burgess. Producing high-dimensional semantic spaces from lexical co-occurrence. *Behavior Research Methods Instruments and Computers*, 28(2):203–208, 1996.

[24] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. *Introduction to Information Retrieval.* Cambridge University Press, Cambridge, UK, 2008. ISBN 978-0-521-86571-5. URL `http://nlp.stanford.edu/IR-book/information-retrieval-book.html`.

[25] Yutaka Matsuo and Mitsuru Ishizuka. Keyword extraction from a single document using word co-occurrence statistical information. In *Proceedings of the Sixteenth International Florida Artificial Intelligence Research Society Conference*, pages 392–396. AAAI Press, 2003.

[26] Andrew Kachites McCallum. Mallet: A machine learning for language toolkit. 2002.

[27] Dirk Merkel. Docker: lightweight linux containers for consistent development and deployment. *Linux Journal*, 2014(239):2, 2014.

[28] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119, 2013.

[29] George A. Miller. Wordnet: A lexical database for english. *Commun. ACM*, 38(11):39–41, November 1995. ISSN 0001-0782. doi: 10.1145/219717.219748. URL `http://doi.acm.org/10.1145/219717.219748`.

[30] Regina Obe and Leo Hsu. *PostgreSQL: Up and Running*. O'Reilly Media, Inc., 2012. ISBN 1449326331, 9781449326333.

[31] UC Santa Barbara Office of Admissions. Guide to course transferability, 2016. URL `https://admissions.sa.ucsb.edu/docs/default-source/PDFs/guidetocoursetransferability.pdf`.

[32] University of Illinois at Chicago. Course transferability, 2016. URL `http://summer.uic.edu/uiuc-uis-studentsapplying/course-transferability`.

[33] Georgios Paltoglou and Mike Thelwall. A study of information retrieval weighting schemes for sentiment analysis. In *Proceedings of the 48th annual meeting of the association for computational linguistics*, pages 1386–1395. Association for Computational Linguistics, 2010.

[34] Razvan Pascanu, Çaglar Gülçehre, Kyunghyun Cho, and Yoshua Bengio. How to construct deep recurrent neural networks. *CoRR*, abs/1312.6026, 2013. URL `http://arxiv.org/abs/1312.6026`.

[35] Jeffrey Pennington, Richard Socher, and Christopher Manning. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543, 2014.

[36] Martin F Porter. An algorithm for suffix stripping. *Program*, 14(3): 130–137, 1980.

[37] Anand Rajaraman and Jeffrey David Ullman. *Mining of Massive Datasets*. Cambridge University Press, New York, NY, USA, 2011. ISBN 1107015359, 9781107015357.

[38] Radim Řehůřek and Petr Sojka. Software Framework for Topic Modelling with Large Corpora. In *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*, pages 45–50, Valletta, Malta, May 2010. ELRA. `http://is.muni.cz/publication/884893/en`.

[39] Stephen Robertson. Understanding inverse document frequency: On theoretical arguments for idf. *Journal of Documentation*, 60:2004, 2004. ISSN 0022-0418.

[40] Herbert Rubenstein and John B Goodenough. Contextual correlates of synonymy. *Communications of the ACM*, 8(10):627–633, 1965.

[41] Magnus Sahlgren. The distributional hypothesis. *Italian Journal of Disability Studies*, 20:33–53, 2008.

[42] Gerard Salton and Christopher Buckley. Term-weighting approaches in automatic text retrieval. *Inf. Process. Manage.*, 24(5):513–523, August 1988. ISSN 0306-4573. doi: 10.1016/0306-4573(88)90021-0. URL `http://dx.doi.org/10.1016/0306-4573(88)90021-0`.

[43] Mike Schuster and Kuldip K Paliwal. Bidirectional recurrent neural networks. *IEEE Transactions on Signal Processing*, 45(11):2673–2681, 1997.

[44] DJD Team. Deeplearning4j: Open-source distributed deep learning for the jvm. *Apache Software Foundation License*, 2, 2016.

[45] Miloslav Konopík Tomáš Brychcín. Owen software - milestone 4 report. 2015.

[46] Kristina Toutanova and Robert C Moore. Pronunciation modeling for improved spelling correction. In *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics*, pages 144–151. Association for Computational Linguistics, 2002.

[47] Texas Christian University. Course transferability, 2018. URL `https://admissions.tcu.edu/apply/course-transferability/`.

A DVD Content

The content of the attached DVD is structured as follows:

- `thesis/` The PDF version of this thesis and LaTeXsource.
- `program/` Contains the application source codes.
  - `.docker/` Contains `Dockerfile` [27] to build the environment.
  - `bin/` Contains scripts to run the application.
  - `data/` Contains the data used for the experiments.
  - `src_owen/` Contains source codes related to the course database.
  - `src_snli/` Contains source codes related to the SNLI dataset.
  - `README.md` Instructions on how to run the source codes.