IT-UNIVERSITETET I KØBENHAVN

Master Thesis

Using Natural Language Processing to Map University Courses to Job Postings

Student
Pedro das Neves Rodrigues Mateus Cristóvão (pedc@itu.dk)
Sruthi Pasham (srpa@itu.dk)

Supervisor
Mike Zhang (mikz@itu.dk)

Department of Computer Science
IT University of Copenhagen
Copenhagen, Denmark

June, 2022

Contents

Acknowledgments

We would like to thank our Supervisor Mike for the knowledge, guidance and availability shared throughout the semester, NLPnorth group for the feedback given during thesis presentation that helped us shape the project, IT University of Copenhagen and Danish entities for the investment in our education and of course our family and friends that were there in the most difficult moments.

Abstract

During university enrollment, students have the difficult task of choosing electives or specialization courses that give them enough knowledge and expertise to fulfill the requirements of desired job positions. This thesis addresses this challenge by using Natural Language Processing (NLP) models that does the matching between jobs and courses and can potentially help students to select course more accurately based on their career hopes. For this study, we use Information Technology (IT) job postings data from the Stack Overflow job ad platform and extract a list of courses from the IT University of Copenhagen (ITU). We pose this task as a multi-label classification task where a model predicts a set of possible courses to take for a given IT job. To obtain labels, we manually annotate unique job titles for courses to take as test and use distant supervision for automatic annotation for training and development. We extract the most relevant information from job postings and use state-of-the-art language models such as BERT (Devlin et al., 2018), RoBERTa (Liu et al., 2019) and JobBERT (Zhang et al., 2022) to predict and evaluate the most relevant courses for a given job.

1 Introduction

1.1 Problem

University course catalogues represent an important resource for students to get to know: Intended learning outcomes, activities or evaluation methods from various courses. Nonetheless, the offer is extremely vast and the terminology used to describe courses does not give an overview of possible real-world applications or career perspectives associated with those subjects.

In order to positively influence our environment (future and current ITU students who come across with this reality) we develop Natural Language Processing (NLP) models that do the matching between IT jobs and IT University of Copenhagen offered courses to potentially help our colleagues maximize the quality of their course choices by not having to down prioritize certain courses because they do not visualize career opportunities related with their fields of interest or simply because there are too many courses to analyze .

1.2 Research objectives

To the best of our knowledge, there is no benchmark for this specific task, therefore we seek to understand to what extent it is possible to apply Machine Learning and NLP models to match industry jobs and academic courses based on the information contained in the job postings.

Moreover, we aim to evaluate whether pre-trained language models BERT (Devlin et al., 2018), RoBERTa (Liu et al., 2019) and the domain-specific JobBERT (Zhang et al., 2022) is able to perform well on this task. And finally, compare the performance of simple baselines with state-of-the-art pre-trained BERT models and understand if more complex models like the latter ones are more suited for our problem than simple ones.

The analysis of the models is based not only in metrics results but also by investigating the outcome of the models for the test set results.

2 Related work

Malherbe and Aufaure (2016) have generated a knowledge base of skills by using tags from Stack Overflow (question and answer website for topics in computer programming) and data from Linked Open Data project DBpedia. This knowledge base is used to associate candidates and jobs to relevant skills and is implemented in industrial context used on a daily basis.

Patel et al. (2017) does job and skill recommendations, analyzing users profiles and resumes, so that user can find currently desired skills and also find relevant jobs, text mining combined with collaborative filtering techniques are used in building this career path recommendation framework.

Rodriguez and Chavez (2019) used clustering algorithms to develop a system that helps job agencies, where requirements from job posting are matched to resumes of job applicants. This is done by extracting information from resumes and job postings, and then identifying key attributes. These attributes are used to build the profiles and then using similarity scores, applicants are matched to job postings, with final output as recommendation of job to an applicant, and an applicant to the company.

Chou and Yu (2020) developed a system that recommends job vacancies for seekers based on their resumes, and provides talent recommendation list for organizations. It analyses resumes submitted by applicants, and the job vacancies and uses cosine similarity between job applications and vacancies.

Elgammal et al. (2021) developed a system that recommends to a recruiter an ordered list of most eligible candidates for a specific job, or an ordered list of most suitable jobs to a job seeker.

It is done by first extracting features by the NLP models built, and similarity between two corresponding features in job-CV pair is calculated using SpaCy Similarity function, and ranking of pairs is done based on their scores. Using this, ranking system gives it's recommendations as output to recruiters and job seekers accordingly.

Lavi et al. (2021) matches jobs with job seekers by fine-tuning multilingual Siamese Sentence-BERT (SBERT), finding similarity between job posts and applicants' resumes, which is possible by training the network with labels that represent a positive signal in case a certain applicant got a phone call from the recruiter or negative if an applicant got rejected.

(Tran et al., 2021) predict job titles from job descriptions using multi-label text classification in a Bi-LSTM-GRU-CNN with pre-trained embeddings and transformers models.

As can be seen there has been done related work in the area of matching jobs with applicant resumes, but to the best of our knowledge our project is the first to match jobs with relevant course work and focus on predicting courses for a job posting.

3 Method

To build a solution, firstly we acquire data from Stack Overflow (question and answer website for topics in computer programming, that also displays IT job posts from around the world). We also extract the most important information regarding ITU's offered courses. Our intention is to use the text in the form of the 'job description' and embed these features to open source models such as: BERT (Devlin et al., 2018), RoBERTa (Liu et al., 2019), and further use an in-house developed model JobBERT (Zhang et al., 2022), which was trained with job posting data (further explanation found in Section 3.3.7) We consider this as a multi-label classification task, and in order to perform this task we will label each IT job with every ITU offered course assigning a value of 1 if an IT job is associated with the specific course and 0 otherwise, as can be seen in Appendix A.1. This assignment is going to be based on the information that we will extract from the ITU website, each bachelor and master program has its own mandatory courses and its job prospects. Our intention is to label courses and IT jobs based on these guidelines provided by the university. Further after getting the embeddings using the above mentioned models, we will use a binary loss function to make the networks learn which descriptions (from jobs and courses) match and then compare the models' accuracy using metrics like: Recall, Precision, Balanced Accuracy, and F1-score.

3.1 Dataset and Preprocessing

3.1.1 Structure and first stage preprocessing

In this section, we describe the Stack Overflow dataset structure as well as the main preprocessing steps that we take in order to pass input, so the models could evolve during training phase to deliver the best possible test results. The job post data we work consists of json files, where in each `json` file represents an unique job post (example of a job post can be found in Appendix A.9). For the task we have in hands, predicting ITU courses that match job titles, we create a dataframe with four main columns: jobId, title - since we use results from this column to group unique job posts by job titles which further are going to be normalized in order to make the hand annotation process feasible, skills and job description which contain the information that is going to be passed to the various BERT models as input.

We split the dataframe into train, validation and test set using a ratio of 80:10:10, respectively, and apply proportional stratified random sampling, to ensure enough variety of job postings and consequently enough variety of job titles for each set.

Since we are only interested in the job posts written in English, we eliminate the ones not

written in English, using Text module from Polyglot library [1]. From an initial total of 3842 unique job offers we reduce this subset into a smaller subset containing 2914 job posts written in English.

3.1.2 Job title normalization

In the 2914 job posts as mentioned above, there were 2168 unique job titles, but many with similar job descriptions. So we decided to normalize the job titles. The criteria for normalization was to check

1. if an ITU 'job prospect' is in the Stack Overflow 'job title', then we chose the 'job prospect'
2. else if a key word is found in 'job title' then we map it to keyword job title
3. and if no such key word is found then we map the job title to 0 (while 0 means the job titles which does not satisfy the above mentioned rules).

We have seen that the instances that ended up with 0 were the ones with obscure job titles, so we filtered them out as part of data cleaning. To simplify a bit further and make sure that job titles are not very sparse, we have filtered out the job posts with job titles occurring less than five times. In the end we were left with 30 job titles, which was also a smaller number for us to do annotation, and for models to learn the important patterns. The code snippet can be found in appendices as Figure 11.

3.1.3 Retrieving relevant information

It is important to note that template of the jobs are similar throughout the dataset. Majority of them consisting of context, type, skills, title, description, job benefits, hiring organization, job location and job id. Nonetheless there are some variants that are difficult and slow down the process of generating consistent input.

An explicit and trustworthy job post must contain at least a section that introduces the vacancy, another about company's mission, requirements and pre-requisites necessary for the job as well as responsibilities. Although the job posts from the dataset implement this logic, based on our analysis, we could observe that talent acquisition departments use a vast number of identical sentences to describe those sections. Further the way the document is formatted differs a lot from post to post, the spaces between sections are empirically defined by the departments and there is a lot of freedom to add picture characters (emojis).

We use mainly BERT models which have an input limitation of 512 tokens. Generally we found that our job postings consists of more than this threshold value of tokens. In order to cope with these constraints, we first remove special characters from the job description part which BERT cannot interpret as can be seen in Figure 1 (an example from a job posting), then as a crucial step we decide to divide the information contained in the posts into three sections: 'introduction', 'main content' and 'conclusion', we get the number of tokens by using the tokenizer relevant for the model (f.ex BertTokenizer for BERT model), and for the job posts which have a description bigger than 512 tokens, we subtract the actual number of tokens of the post by the threshold, the result is then divided into two parts, the first one represents the number of tokens we need to ignore in the beginning of the job description, 'introduction', and the latter the number of tokens we need to delete from the 'conclusion' section, the code snippet used can be seen in Appendix A.10, this way we always have posts with an acceptable number of tokens containing information relevant to feed our models with. For example in Figure 2 the three paragraphs can be seen as 3 parts, the second paragraph is the 'main content', and the Figure 3 shows the start tokens (start of second paragraph) and the Figure 4 shows the end tokens (end of second paragraph).

[1] https://polyglot.readthedocs.io/en/latest/index.html

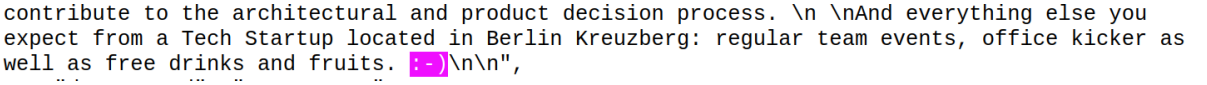


Figure 1: Example of special character in job posting

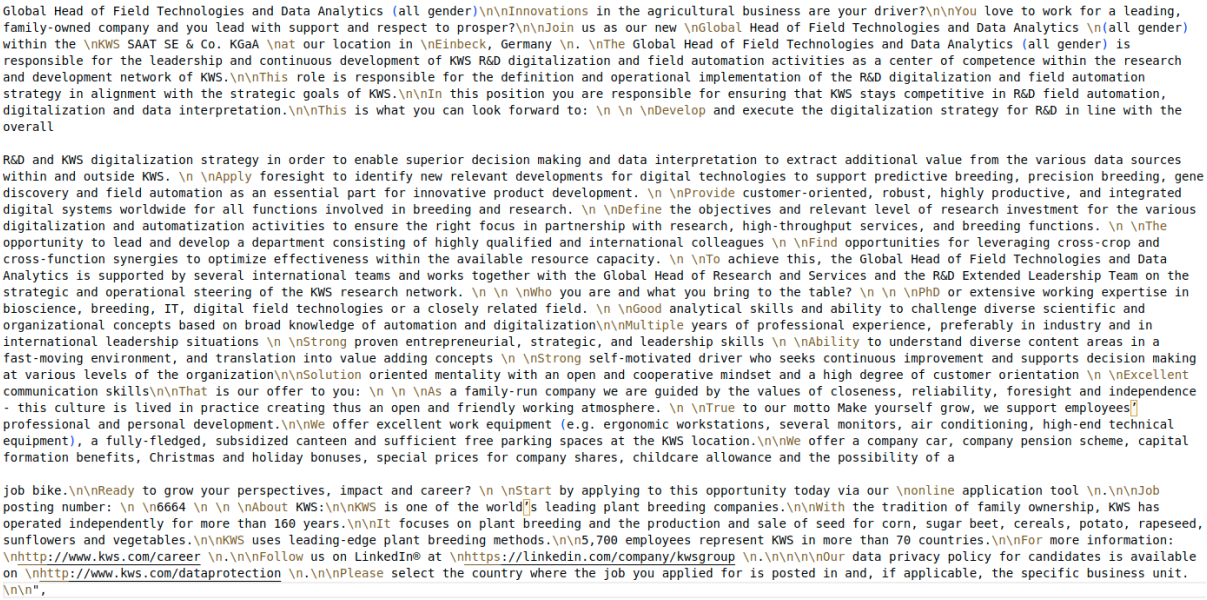


Figure 2: Job posting with more than 512 tokens

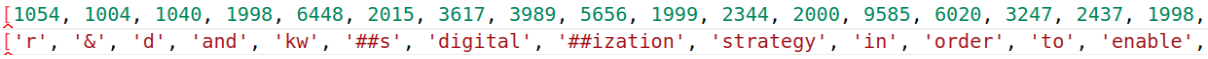


Figure 3: Start token of 'main content'

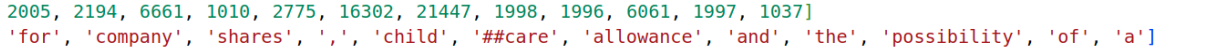


Figure 4: End token of 'main content'

3.1.4 ITU Courses

A subset of the university offered courses is defined as the target for our models and baselines. We scrape from ITU's catalogue courses offered by two master's programmes: Computer Science (CS) and Software Design (SD). Although we had thought of working with all the available English courses at ITU for the semesters of Autumn 2021 and Spring 2022, this would represent close to 100 courses in our set, which could have resulted in an increase of time and effort allocated to the manual annotation process and potentially a class imbalance in our target set. Although we focus our research on these two degrees, we still have a set of courses (58 unique ones) that cover a substantial amount of IT fields (Security, Software Development, Artificial Intelligence or Game Development), both programmes have a broad scope, students can specialize in a wide variety of technological fields.

In order to retrieve the necessary subjects, we first analyze whether the mandatory courses of a degree can be attended by students of the other degree in the form of electives or specialization courses and if it is not the case these courses are automatically excluded from our set, since we want our software to output as predictions courses that students can decide to take not courses that are mandatory for the respective degree. Due to the structure of the master's programmes at ITU, some courses can be added to the electives section as specialization courses from the same degree or as mandatory course from the other degree like for instance: Practical Concurrent and Parallel Programming, which represents a specialization or elective course for

Software Design students and a mandatory course for Computer Science students.

Thus we have to find the intersection between electives from both degrees in order to remove certain redundancies and add electives or specialization courses specific to each degree.

3.1.5 ITU job prospects

The main purpose of our project is to develop a model capable of predicting university courses based on job postings. As mentioned above we normalize the job posts from the Stack Overflow unlabeled dataset, this task is based on a set of possible career perspectives that we are able to identify for each degree, by searching on ITU official website we are able to retrieve some of the most significant career perspectives for students of each master and further based on the possible specialization offers for both programmes we are able to infer some more job titles thus being capable to perform more accurate job title normalization.

3.2 Annotations

3.2.1 Manual annotations

In order to understand whether BERT-based models are able to predict and suggest faculty courses based on job descriptions as well, or at least similarly to what a human could possible do, we have to manually annotate the test set labels for each job posting.

The test set represents around 275 job posts, composed of 30 unique job titles. We use a matrix of 30x58 (number of unique job titles and number of offered courses) to match job titles with courses, assigning a binary value of 1 to an entry of the matrix in case of a match between row and column value as can be seen in Appendix A.1.

We individually annotate the matrix, a process of three rounds. In each of the rounds we select the first five rows to assign values. It is relevant to mention that in each round we get a score, which can be defined as annotation agreement, by using Krippendorff's alpha metric.

For the first round we did not specify any rules, we filled the entries based on the knowledge gained when defining possible career perspectives associated with the degrees in question, and also based on the knowledge we gained from studying at IT University of Copenhagen during the last year and a half enrolling on some of the courses that serve as a label for our project.

We set some basic rules for the second iteration:

1. If specialization course and specialized 'job title' then assign a value of 1
2. If programming related course and 'job title' is generic and programming oriented then assign a value of 1

In the end we got better consensus results and we swap the values that we had previously for the first five rows with these new values.

For the last iteration we create two matrices (example on Figure 5) for both job titles and ITU courses respectively to better segment the available options. The rows represent the CS and SD, IT related fields such as: Business Analytics, Systems Maintenance, Security or Algorithms and Data Structures. And columns associated with the level of technical expertise needed given the field. From 1 (low technical expertise) to 3 (high technical expertise).

Afterwards we complete the matrix using the guidelines of the last round which have a significant increase in the score. The annotation agreement final score is 0.63, and even though it is smaller than the score we get in the previous iteration, it is considered relevant.

The assigned values of the matrix consisting of job titles and courses from the test set are a result of the overlap between the annotators' tables, in case of disagreement, we opt to assign a value of 0 for those entries.

ITU Courses Helper Matrix			
	Technical expertise given the field		
IT fields	1	2	3
Software Development/Engineering	A, B	D	E
Database systems		C	
Security	F	G	
Algorithms and DS	W		A
Business Analytics		G	X,Z
Systems Maintenance	T	K	
Games		J	
Network systems	J		C,V
Interaction design		S	
Artificial Intelligence		Q	

Figure 5: Example of an helper matrix, where each letter simulates a course

Round	Score
1	0.44
2	0.50
3	0.81

Table 1: Krippendorff's alpha

3.2.2 Distant Supervision

For the training and validation set we automatically label the job postings, we re-use the matrices defined when creating guidelines for the hand-annotation process to help create a lookup table. Again, we focus on matching normalized job titles with courses, this way we can match job posts and courses since each post is linked with one of the 30 unique job titles. The lookup table contains as keys the job titles with a value in the form of list of matching courses.

If a subject and job title are in the same entry in their respective matrices, then, both require the same level of technical expertise given the field in question and so we must include that specific course in the list linked to the job title. Further we decide to add some more courses to each job title by incorporating subjects in the same field of the job in question but requiring one level less of technical expertise. This way we are able to reinforce the lists of courses with subjects that are not crucial or teach core concepts for the role in question but are relevant enough to improve the performance of someone in that job position.

After filling the required key value lists, we transform the strings (ITU courses) into integers representing the position of each subject in the courses array. we make this changes because it is more efficient to work with index when creating a final matrix with all labels assigned as zero or one.

3.3 Models

3.3.1 Bag of words (BOW)

BOW is a feature extraction algorithm that transforms the text into fixed-length vectors. This is possible by counting the number of times the word is present in a document or a paragraph. The word occurrences allow to compare different documents and evaluate their similarities for applications, such as search, document classification, and topic modeling. BOW aims to extract features from the text, which can be further used in classification(Zhou, 2019).

3.3.2 Term Frequency - Inverse Document Frequency (TF-IDF)

TF-IDF is a text vectorizer that transforms the text into a matrix of TF-IDF features. The term frequency is the number of occurrences of a specific term in a document. It indicates how important a specific term in a document. Document frequency is the number of documents containing a specific term, it indicates how common the term is. Inverse document frequency (IDF) is the weight of a term, it aims to reduce the weight of a term if the term's occurrences are scattered throughout all the documents (Ramadhan, 2021).

(Borcan, 2020) First step is to calculate the term frequency.

$$\text{tf}(w, d) = \log(1 + f(w, d)) \tag{1}$$

- d is a given document from our dataset
- w is a given word in a document

Here $\text{tf}(w, d)$ is the frequency of word w in document d.

Second step is to calculate the inverse document frequency.

$$\text{idf}(w, D) = \log(\frac{N}{tf(w, D)}) \tag{2}$$

- N is the number of documents we have in our dataset
- D is the collection of all documents

With N documents in the dataset and $tf(w, D)$ the frequency of word w in the whole dataset, this number will be lower with more appearances of the word in the whole dataset.

Final step is to compute the TF-IDF score

$$\text{tfidf}(w, d, D) = \text{tf}(w, d) \cdot \text{idf}(w, D) \tag{3}$$

3.3.3 Logistic Regression

Logistic Regression is a supervised machine learning algorithm which is used to predict the probability of a target variable. Logistic regressions are only binary classifiers meaning they can handle the classification tasks with two classes. However, there are heuristics such as One-vs-rest (OVR), also known as One-vs-all (OVA) multi-label classifier where we fit one classifier for each class and this is the most commonly used strategy for multi-label classification problem.[2] It involves splitting the multi-label dataset into multiple binary classification problems. A binary classifier is then trained for each class predicting whether an observation is that class or not. Each model predicts a class like probability score and the class with a largest score is used to predict a class (Nabriya, 2021).

[2]https://scikit-learn.org/stable/modules/generated/sklearn.multiclass.OneVsRestClassifier.html#

3.3.4 Word2Vec

Word2Vec has two variants which are as follows:

1. Predicting target words from contexts: which is also referred as the continuous bag-of-words (CBOW) model. The model tries to predict the i^{th} word W_i, in a sentence using a window of width j around the word. Therefore, the words $W_{i-2}W_{i-1}..W_{i+1}W_{i+2}$ are used to predict the target word Wi. Here the model predicts one word out of d outcomes also known as the multinomial model. Visual representation can be found in left half of the Figure 6.

2. Predicting contexts from target words: which is also referred as the Skip-gram model. Given the i^{th} word in a sentence, which is denoted by W_i, the model tries to predict the Context $W_{i-2}W_{i-1}..W_{i+1}W_{i+2}$ around the word W_i. Here this technique models whether or not each context is present for a particular word which is also known as Bernoulli model. This approach uses negative sampling for efficiency (Aggarwal, 2018). Visual representation can be found in right half of the Figure 6.

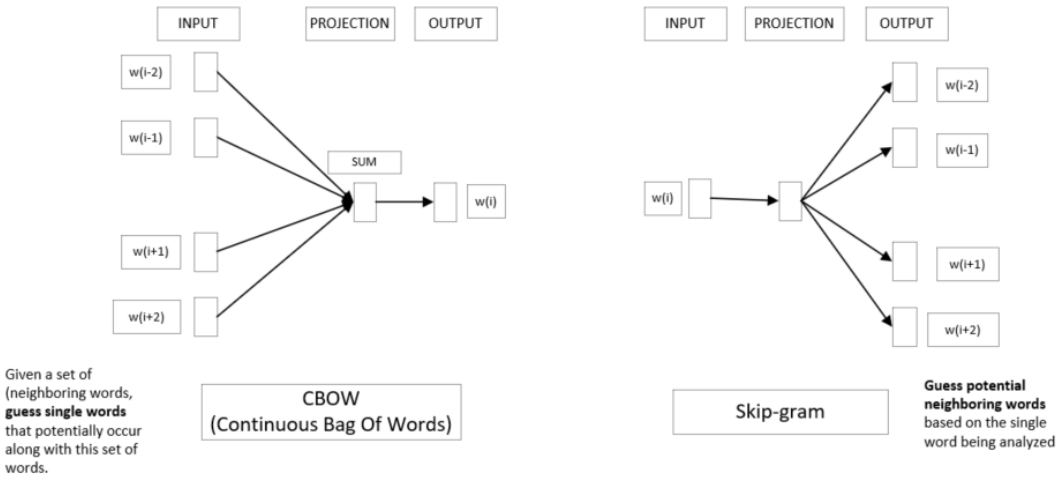


Figure 6: CBOW and Skip-gram
(Mikolov et al., 2013)

3.3.5 BERT

Bidirectional Encoder Representations from Transformers (BERT) (Devlin et al., 2018) is a language model, which consists of multi-layer bidirectional Transformer blocks. There are two main steps in its framework, pre-training and fine-tuning. During pre-training, the model is trained on unlabeled large corpus. BERT uses two objectives during pre-training, these are Masked Language Modeling (MLM) and the Next Sentence Prediction (NSP).

- MLM: In the input sentence, 15% of random sample of tokens are selected and replaced with special token [MASK]. The model tries to predict the masked words based on the context provided by non-masked words in a sentence. The BERT cross entropy loss function takes only the prediction of masked values into consideration and ignores the prediction of non-masked words.

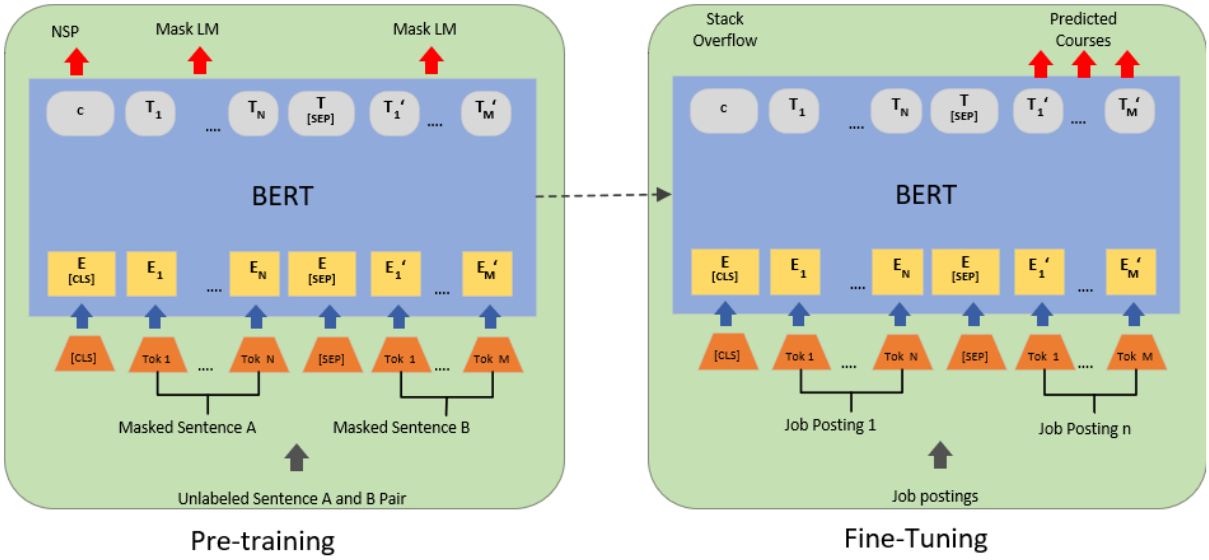


Figure 7: BERT Architecture
(Devlin et al., 2018)

- NSP: A pair of sentences are given to the model as input and learns to predict whether the second sentence follows the first sentence in the original text. 50% of the positive examples are created by taking consecutive sentences from the original text corpus and 50% of negative examples are created by the pairing segments from the different documents. As can be seen in Figure 7 in the outcome of Pre-training part, c is a binary output predicting 1 if sentence B follows sentence A, otherwise 0.

When training the model, both MLM and NSP are trained together with a goal of minimizing the combined loss function of the two strategies.

BERT has two model sizes BERT$_{Base}$ and BERT$_{Large}$. A large number of encoder layers are present in both BERT models. BERT$_{Base}$ version has 12 encoders(Transformer Blocks), 768 hidden units and 12 self-attention heads where as BERT$_{Large}$ version has 24 encoder layers, 1024 hidden units and 16 self-attention heads. BERT uses static masking i.e. the same part of the sentence is masked in each epoch. BERT is pre-trained on a combination of 0.8 Billion words from 'Toronto BookCorpus' and 2.5 Billion words from 'English Wikipedia datasets' i.e. as a total of 3.3 Billion words of data and uses a batch size of 256 with 1 million steps.

For fine-tuning the BERT model, it is initialized with the pre-trained parameters, and all the parameters are fine-tuned using labeled data for specific tasks. In case of our project, as can be seen in Figure 7 in the Fine-Tuning part, we pass the job postings as the input and get the token embeddings which are then passed to the BERT model. Prediction of courses is the outcome of the model.

3.3.6 RoBERTa

RoBERTa is a better version of BERT model. It uses a strategy called dynamic masking where different part of the sentences are masked for different epoch which makes the model robust compared to the BERT model which uses the static masking where in each epoch same part of the sentence is masked (Liu et al., 2019). RoBERTa only uses the MLM task and ignores the NSP task as it was observed that it is not beneficial in pre-training the BERT model. RoBERTa in addition to the 'Toronto BookCorpus' and 'English Wikipedia datasets' also trained on open web text and Common Crawl-News (CC-News) and so forth. RoBERTa used a batch size of 8K with 31K steps in order to improve the speed.

3.3.7 JobBERT

JobBERT (Zhang et al., 2022) is a domain-adapted model (Gururangan et al., 2020) pre-trained from a BERT_{Base} checkpoint and trained on more than 3 million sentences from similar job postings using default BERT model hyperameters. This self-supervised pre-trained language model might possibly obtain better results than vanilla BERT when extracting skills and knowledge from domain specific to JobBERT.

3.4 Additional Fine-Tuning Techniques

There are different fine tuning techniques that can be performed (Joshi, 2020)

- Train the entire architecture - We can use the pre-trained model and train it further on our dataset using supervised approach, and optimize hyper-parameters.
- Train some layers while freezing others – This is another approach where we freeze weights of initial layers and retrain only some layers.
- Freeze the entire architecture – In this approach we can freeze the entire architecture and add new layers and train the new model while updating only weights of the newly added layers.

3.5 Evaluation metrics

3.5.1 Confusion matrix

Confusion matrix is used to evaluate the performance of the classification algorithms. The matrix illustrates the relationships between the outcome and prediction of classes. Each row of the matrix represents the instances in a predicted class, while each column represents the instances in an actual class as shown in Figure 8.

Figure 8: Confusion matrix
True positive (TP): Correct prediction of positive class
True Negative (TN): Correct prediction of negative class
False Positive (FP): Incorrect prediction of positive class
False Negative (FN): Incorrect prediction of negative class

3.5.2 Precision

Out of all the classes, how much we predicted correctly.

$$\text{Precision} = \frac{TP}{TP + FP} \tag{4}$$

3.5.3 Recall/ Sensitivity

Sensitivty is also known as True Positive Rate (TPR). It is used to measure how much we predicted correctly of all the positive classes. It should be as high as possible.

$$\text{Recall} = \frac{TP}{TP + FN} \tag{5}$$

3.5.4 F1-score

F1-Score is the Harmonic Mean between precision and recall.

$$\text{F1-score} = \frac{2 * Recall * Precision}{Recall + Precision} \tag{6}$$

3.5.5 Balanced Accuracy

Balanced Accuracy is a metric used to evaluate the performance of an imbalanced dataset, and it is simply arithmetic mean of True Positive Rate and True Negative Rate as can be seen in below formula.

$$\text{Balanced Accuracy} = \frac{1}{2}\left(\frac{TP}{TP + FN} + \frac{TN}{TN + FP}\right) \tag{7}$$

4 Results

4.1 Baseline models Results

Model	Parameters	Precision	Recall	F1-score
TF-IDF	max_features = 2000	**0.676**	0.698	**0.679**
Word2vec-CBOW	min_count = 2,size = 100, window= 5	0.635	0.579	0.583
Word2vec-Skip gram	min_count = 2,size = 100, window= 5	0.660	**0.711**	0.634

Table 2: Baseline models Results

The results displayed in table 2 showcase higher TF-IDF values for Precision and F1-score compared with the remaining models. Word2Vec with CBOW architecture is outperformed by the other two models, registering the lowest scores for all standard metrics. However Word2Vec - Skip gram registers the highest score in the table for Recall. It is relevant to mention that Baselines are not fine-tuned, the parameters used are standard or empirically chosen, nonetheless, further Baseline testing is performed and addressed in Appendix section A.3 and A.4

4.2 BERT , RoBERTa and JobBERT Results

Model	Precision	Recall	F1-score
BERT	0.633	0.795	0.699
RoBERTa	0.633	0.795	0.699
JobBERT	0.633	0.795	0.699

Table 3: BERT , RoBERTa and JobBERT Results

As depicted in table 3, various BERT architectures output the same values for Precision, Recall and F1-score. Although there are no improvements from baseline BERT, we note that these architectures are able to outperform the ones from table 2 with respect to Recall and F1-score. The three models are trained using the same hyperparameters, constant learning rate of 1e-2, batch size of 4 with no more than one training iteration. [3]

5 Posterior Analysis

For both baselines and pre-trained BERT models we perform a posterior analysis on the model's predictions. Although we have been using standard metrics such as: Precision and Recall to infer how well the various architectures are responding, a more deeper analysis of the output is performed.

Firstly, we analyze whether the models are actually learning how to suggest courses based on the job posting description and secondly, we assess the quality of the outcome of each model and compare not only BERT based models with themselves, but also to compare them with baselines, thus try to comprehend to what extent state-of-the-art NLP models are valuable option in comparison with simpler models such as: Word2Vec or TF-IDF, for the task in hands.

As a tool that could possibly influence our colleagues decisions, we are aware that having models that predict courses based on career expectations need to be trustworthy in order to be relevant. Therefore, we pay special attention to the false positives that appear on each job posting. Using a confusion matrix we are able to retrieve these values and identify which normalized job titles are more prone to indicate courses that are not relevant for future careers and further calculate the average of job titles' false positives for each model.

Further we aim to collect information concerning the percentage of courses covered by each job title, that could potentially help us determine which type of job postings are being better predicted. Again the results for each model will be averaged out and showcased in further subsection as the mean of covered courses for the architecture in question.

In the latter part of this section we intend to check, individually, course results, in order to gauge which subjects are being better and worse classified by the models, to perform this task we evaluate each course using Balanced Accuracy, which is usually deployed in multi-label binary classification problems to deal with imbalanced classes.

5.1 Baselines

5.1.1 Term Frequency Inverse Document Frequency

The outputs of the model reveal discrepancies in terms of the distribution of false positives throughout the various normalized job titles. The position that the model is able to predict without suggesting any non relevant courses is for example: Back-end Developer, it has no false positives assigned. Other job titles like Software Engineer, IT Project Manager, Developer and more specifically Full-Stack Developer or Java Developer register less than 1.5 false positives.

[3] Models are trained using Google Colab

Opposite Security Engineer, QA Engineer and Site Reliability Engineer are assigned with more than 7 false positives.

In terms of percentage of courses covered by each job title, positions that obtain good results with respect to the frequency of false positives, evidence full coverage of courses assigned. Further, we note that job titles like Python Developer or Front-End Developer, that have respectively 3 and 4 false positives assigned through the whole job posting test set, have also full coverage of courses required to perform the role. Negatively that 14 of the 27 job titles that have at least one course assigned to itself have a 0% coverage, showcasing evident difficulties by the model to predict certain job titles.

Finally in order to understand how accurate does the baseline predict courses, we calculate the True Positive Rate (TPR) or Recall and True Negative Rate (TNR) to derive the Balanced Accuracy for the test set. The results we obtain make us conclude that the model does not suit particularly any of the 46 university courses with at least one job title assigned. The amplitude between courses' rate (difference between lower and higher value) is only 0.10. The metric used, represents the sum of TPR and TNR divided by 2. From what we could analyze there is a subject bi-polarization, some courses have very high rates for Recall and others for TNR which make them converge to around 0.5 of accuracy.

Avg	Score
False Positives	5.320
Balanced Accuracy	0.498
Covered Courses	0.394

Table 4: TF-IDF global average values

5.1.2 Word2Vec (Skip gram)

The false positive results for normalized job titles evidences only 3 positions with an average of false positives smaller than 1.5, Mobile Developer is the job title which the model does less severe mistakes if we consider false positives or suggesting courses that are not relevant for someone's career as the worst possible scenario. There are at least 5 positions that generate more than 8 false positives, being Machine Learning Engineer the highest frequency, 10, followed by professions like System Consultant Tester or UI/UX designer.

Regarding the percentage of courses covered, there are 4 job titles that contain all the courses that were previously assigned: Back-end Developer, Full Stack Developer, Python Developer and Solution Architect. Further there are a high number of job titles without any course coverage, 15 in 27.

In terms of courses analysis, we can conclude that the model has similar capacity to predict each of the several courses right given the job posts. The amplitude of values in terms of Balanced Accuracy for each course is around 0.18. The courses Advanced Programming and Advanced Software Engineer, both related with Software Engineer title, perform slightly better than the rest obtaining a final score of 0.64, whereas courses such as UX Design or Technical Interaction Design are two examples that under-perform compared with the average.

Avg	Score
False Positives	5.493
Balanced Accuracy	0.503
Covered Courses	0.344

Table 5: Word2Vec global average values

5.2 Pre-trained models

5.2.1 BERT, RoBERTa and JobBERT

The results above mentioned showcase identical performance for BERT, RoBERTa and JobBERT models, the posterior analysis emphasize this similarity, the models have exactly the same performance in the 3 topics here depicted: count of false positives and percentage of courses covered by each job title as well as Balanced Accuracy results for the offered courses.

It is noteworthy that there are no IT positions without any false positive to report, the job title that gets closer is Software Engineer with just 1 average count for the posts with this title, followed by Back-end Developer and Developer on the negative side we notice more than half of the IT positions with 9 false positives on average including professions like: tester, system consultant or Data Analyst, contributing for the increase of global average values of false positives comparing with baselines.

The percentage of courses covered increases slightly, we obtain a number of job titles with full coverage superior compared with baselines, 9, one more than TF-IDF and 5 more than Word2Vec, but it is important to consider that the number of false positives increased and therefore we believe that this slightly higher results may be a consequence of that growth. It should be mentioned that jobs with high coverage, not only overlap with simpler models above mentioned but are usually the ones that appear with more frequency in the test set, such as: Software Engineer, Developer or Full Stack Developer.

In terms of capacity to accurately predict ITU courses, the model evidences total neutrality, with a global average value of 0.5 and a variance of 0, proving that the models are not actually predicting certain courses better than others, instead, they are generalizing with respect to the most commonly seen jobs across the training set. We address this problem in further sections.

Avg	Score
False Positives	7.1
Balanced Accuracy	0.5
Covered Courses	0.403

Table 6: BERT and RoBERTa global average values

6 Discussion

Here we will outline the results we got from Section 4 and 5, intersect the information that we got from these sections and discuss the results of potential upgrades to pre-trained models. It is relevant to note that all the Transformer models are tested in similar conditions, with exactly the same hyperparameters. Baselines results are also based on standard parameters, although we made some further experiments with different parameters as we depict in Appendix A.3 and A.4 .

6.1 Standard metrics' relevancy for the specific task

The results we get from standard metrics may evidence a strong capacity of the listed pre-trained models to perform the assigned task. Further analysis developed in Section 5 reveal that the capability of the models to predict courses based on standard metrics like Recall might be insufficient. Recall outperforms the other two metrics in all Transformer models reaching values close to 80%, but this statistical measure only takes into account true positives and false negatives. For the training set, around 15% of the data entries represent positive assignments, similarly, test set contains around 11% of positive ones, due to this imbalance in the data, the

models are expected to predict for each job title a number of courses considerably inferior compared with the total amount of available ones, so the number of false negatives is clearly smaller than the number of true positives, which is extremely high in certain job postings predictions.

On the other hand, Precision seems to be a metric better suited to evaluate models performance on the assignment compared with Recall. We believe that the number of false positives the models output are extremely relevant to understand how close are we from building a trustworthy system that does not predict misplaced courses and influence negatively students' academic choices.

6.2 Course suggestions

Predictions have usually a predefined pattern in terms of courses assigned independently of the job post and respective job title observed and logically the number of false positives for each type of job title is often stable.

We then try to understand what courses are usually assigned as the outcome of the models, by retrieving the indexes of the positive assignments, for the most frequent prediction pattern, we are able to match these indexes with the corresponding ones from the list of offered courses by the university.

Courses
Advanced Programming
Advanced Software Engineering
Algorithm Design
Applied Algorithms
Frameworks and Architectures for the web
Functional Programming
Mobile App Development
Practical Concurrent and Parallel Programming
Software Architecture

Table 7: Recurrent suggested courses

Table 7 results showcase the most commonly predicted courses. The referenced courses can be related with each other either in an academic perspective, formal pre-requisites such as: specific programming language proficiency or previously taken courses required to enroll in the courses, and in future career paths. Looking at the results in Section 5 in relation to these architectures we are able to correlate certain job titles with these presented subjects, Software Engineer only gets 1 false positive and similar positions like: Back-end or Full Stack Developer get no more than 3 false positives and in both cases the coverage of relevant courses is total, which makes us conclude that these models tend to suggest subjects required in Software Engineering/Development IT sub field.

6.3 Underfitting issues

Although the best results described in Section 4 concerning pre-trained models suggest that just one epoch and an aggressive learning rate of 1e-2 is possibly the right way to train the models for this specific task. Previous subsections analysis and results showcase evident signs of generalization, we must assume that the hyperparameter configuration used to obtain the best performance contributes to emphasize underfitting problems.

In order to contradict this trend we implement different methods in JobBERT (Zhang et al., 2022) to test whether it is possible to improve. We first increase the training duration by adding more epochs and further we implement a linear scheduler that reduces the learning rate as the

number of iterations progresses, thus make the model learn general patterns at the first stage of training and more complex patterns at later stages by slowing down the learning process. The results we get do not demonstrate any improvements, using the same batch size as before and 3 epochs, we are not able to increase any of the metrics' values, instead F1 and Precision values decrease to scores lower than 0.5 whilst Recall drops to values slightly higher than 0.5. Additionally, we note that validation loss does not improve from first to last epoch, it is constant, potentially meaning that the model is not able to capture other data patterns than the ones it captures at the beginning of training when the learning rate is extremely high making the model to converge quickly.

Afterwards we add an additional final layer with a minimal dropout probability, we firstly freeze the weights of the BERT model and fine-tune just the last layer, and again we use the same hyper-parameters as the ones on the previous chapter. Metrics' results show an increase of Precision values of around 6 points. It is important to mention that just by looking at the outcome of the model we notice that predictions are still quite homogeneous. Even though this configuration, reduces the number of false positives, inferring by the decrease in Recall score, the amount of false negatives is higher than the one in Section 4, meaning that a significant number of courses would be disregarded.

Finally we make one last change, we unfreeze the pre-trained model weights and fully train JobBERT (Zhang et al., 2022) model, this technique does not outperform the previous ones and evidences signs of model generalization by looking through the predictions and checking the distribution of courses covered given job titles which is identical to the one in Section 4.

6.4 Homogeneous distribution of job posts

The subset from Stack Overflow, is mostly composed of Software Engineering job postings or other related positions. Posts with a title of Software Engineer and Developer, combined, represent 0.607 of the total number of job postings with an assigned normalized title. Given this scenario, we redefine the distribution to evaluate whether this imbalance in terms of job titles appearances in the dataset is contributing for pre-trained models' generalization of predictions to Software Development/Engineer related positions.

In order to turn this distribution into a more heterogeneous one, we make small changes in the normalization part coding section, basically in the process of assigning a post to a role, we make sure that job titles such as Developer or Software Engineer are the last ones to be suggest, plus, job post will only be defined as Software Engineer in case their not normalized job titles contain both strings that form the job title.

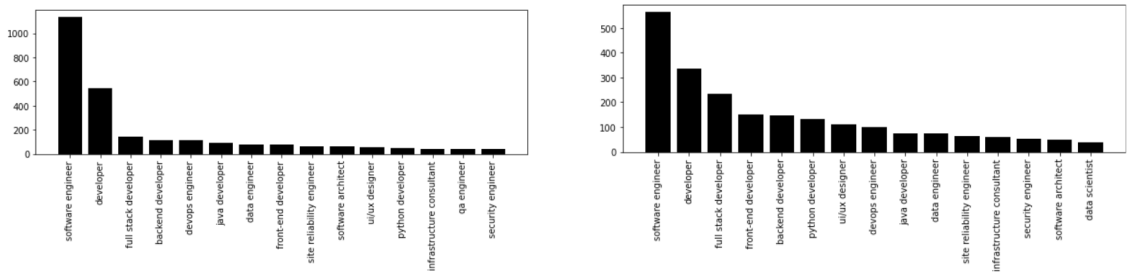


Figure 9: Old vs New distribution of job titles.

Although we are able to decrease the number of job posts associated with Software Engineer profession from 1135 to 564 and reduce significantly the relative weight of the two most assigned courses, we still cannot improve the performance of our model.

Metric	Score
Precision	0.587
Recall	0.651
F1-score	0.606

Table 8: JobBERT with new labels and section 4 hyperparameters

6.5 Comparison between manual and automatic annotation

To better understand why the models are not able to improve from the threshold defined in Section 4, we go through the manual and automatic annotations to evaluate whether there are steps performed before training phase that could potentially be preventing the pre-trained models to improve their metrics scores and reduce generalization issues.

It is a challenging task to have the same quality of annotations for both test and train sets, as we mention in Section 3, we try to add consistency to these sets by developing two helper matrices that could classify each job title or course by its IT field and level of technical expertise, these matrices represent the knowledge used to perform distant supervision task, and although we have defined some initial guidelines for the manual annotation process, a significant number of annotation assignments is based on these two tables, nonetheless for both annotators knowledge gained concerning certain courses and job positions may have influenced the assignment decisions, and have slightly compromised the similarity between testing and training annotations.

In order to complement this analysis, we evaluate directly the testing and training annotations by applying standard metrics (Precision, Recall and F1-score) used in previous sections and subsections.

Metric	Score
Precision	0.608
Recall	0.968
F1-score	0.747

Table 9: Distant supervision performance

The automatic annotations set results evidence high sensitivity, in total are only 4 false negatives. Nonetheless, the number of false positives is high, which indicates that automatic annotations cover more courses per job title than the final version of the manual annotation, which only assigns positives in case there is an overlap between annotators assignments.

Based on these insights, we redefine the merging strategy for both annotators tables, by assigning a value of 1 in case there is a contradiction between annotators, to firstly understand if distant supervision precision's score increase, and secondly if an higher positive predictive value influence models' capability of being more precise.

Metric	Score
Precision	0.920
Recall	0.738
F1-score	0.819

Metric	Score
Precision	0.720
Recall	0.613
F1-score	0.656

Table 10: Distant supervision (left) and JobBERT (right) results

As expected, we are able to increase the Precision values when comparing manual annotation (targets) with automatic one (predictions), as the amount of true positives growths and false positives decreases. Using the same hyper-parameters of Results' section We are capable of reaching a Precision score for of 0.720 when running JobBERT(Zhang et al., 2022), which is the

higher result we get in all the different experiments with these metric. Curiously, the average percentage of false positives (6.133) is inferior to one that we get in Section 4 but still, superior to the ones we obtain in Section 5, therefore we to classify this average score as considerably high for the task we intend to accomplish.

6.6 Vanishing gradient and global minimum

During the training phase, we start noticing that JobBERT (Zhang et al., 2022) model struggles to improve performance over the epochs. In the cases we train the model for more than one training set iteration, we found that there is no improvement whether in terms of validation loss or overall metrics performance, no matter which solution we try to implement. By looking at the gradient of the model's parameters, we note that the gradients after the first iteration are often equal to 0 and when it does not happen the values are still very low, confirming our suspicions that the model is not able to learn more than what it does through the first epoch, using a learning rate of 1e-2 (which have been generating the best metrics results for only one iteration) seemingly converging to a global minimum.

Further we try to keep an aggressive learning rate constant throughout the training process of the model for frozen BERT gradients. And although the weights and validation loss gets updated it does not translate to better results or constant decrease of loss, which makes us believe that the full potential of these models given the supplied data is reached with little training needed.

6.7 Baselines vs pre-trained models

As a research objective we sough to understand whether pre-trained BERT based models would perform better than simple models such as: Word2Vec or TF-IDF in this particular task. By purely observing the results of Tables 2 and 3 we should state that pre-trained models indeed outperform our baselines. These pre-trained versions of BERT are able to obtain much higher Recall scores beating by a fair margin Word2Vec and TF-IDF, on the contrary, the baselines get better results for Precision both models beat the ones from table 3, but if we look at the results of F1-Score which takes into account both Precision and Recall, we observe that state-of-the-art models outperform the models from Table 2.

As it has been stated before we pay special attention to the number of false positives as the output of the models, so we could argue that baselines may be a better fit for this project, moreover the results of Table 2 do not showcase the best configurations for these simpler models, only a standard one, since they represent baselines. Nonetheless and as part of our research we test both Word2Vec and TF-IDF using several parameters as it is shown in the Appendix Sections A.3 and A.4. The results do not represent a huge improvement from what we get in Section 4, but still the best Precision result for TF-IDF reveal a score of 0.698. However as we demonstrate in subsections 'Underfitting issues', we are able to increase the Precision values by using an additional layer and freezing BERT model weights.

7 Future Research

As for future work, some structural decisions concerning the way we are addressing this multi-label classification task would be revised. Instead of having such an homogeneous distribution of job posts with total dominance of Software Engineering related positions, we intend to create a balanced dataset which contains the exact same number of job posts for the designated job titles. Secondly, we would seek to work with more data, we have been supplied with roughly 3800 job posts from Stack Overflow, but since many of them are not written in English, the final number of posts that serve as input to the models gets below 3000, which facilitates a

bit the training process, since it is faster to train state-of-the-art models, but we believe that with such number of inputs the results may get a bit compromised. Further we would like to review the annotation process, as stated previously in Discussion, small changes to the way we create our final annotation test set increase significantly our Precision scores, as we are able to obtain the best results regarding this metric, by just changing annotators' merging strategy. So by reviewing guidelines and basic rules for both manual and automatic annotation while increasing the number of iterations for test set consensus, we expect to increase the overall performance of the models.

Lastly, we would try to redefine the current task by initially splitting job titles and courses based on the associated degree such as: Computer Science, Data Science or Software development, and this way feed the models with denser labels by running the networks for each ITU degree individually.

8 Conclusion

In this work, we propose an innovative framework that could match jobs from technological fields with IT University of Copenhagen offered courses, using diverse NLP models ranging from simpler ones like TF-IDF to complex Transformer models.

We were able to retrieve valuable information from job posts and gain insights about various IT job positions, which not only enabled us to normalize a vast list of job titles but also associate and match them with respective master's courses. Thus we could construct models (including baselines) capable of reaching good performance levels for standard metrics, registering scores superior to 0.7 in several experiments.

Regarding our first research objective, currently we are not in a position to state Transformer models as a reference or a reliable tool to tackle this task, even though we obtained good performance overall, after an outcome analysis we came to the conclusion that the models were generalizing too much and instead of suggesting courses given job titles preferences they were actually predicting courses based on market's needs, as the majority of courses predicted represent subjects related with the professions that appeared the most in our subset of posts from Stack Overflow and are in high-demand with respect to technological sector, examples: Software Engineer, Full Stack, Backend or Front-end Developer.

Although we were expecting that JobBERT as a domain related model would be able to outperform BERT and RoBERTa at this particular assignment, that did not happen. Surprisingly, all the three models perform at the same level.

Lastly we verify that fine-tuned pre-trained models are able to get better performances than standard baselines, however the margins are not considerably superior and it can be argued that simpler models are more indicated to this task if we consider Precision as the most significant metric.

References

C. C. Aggarwal. *Text Sequence Modeling and Deep Learning*, pages 305–360. Springer International Publishing, Cham, 2018. ISBN 978-3-319-73531-3. doi: 10.1007/978-3-319-73531-3_10. URL https://doi.org/10.1007/978-3-319-73531-3_10.

M. Borcan. *TF-IDF Explained And Python Sklearn Implementation*, 2020. https://towardsdatascience.com/tf-idf-explained-and-python-sklearn-implementation-b020c5e83275 [Accessed: 2022-05-08].

Y.-C. Chou and H.-Y. Yu. Based on the application of ai technology in resume analysis and job recommendation. In *2020 IEEE International Conference on Computational Electromagnetics (ICCEM)*, pages 291–296, 2020. doi: 10.1109/ICCEM47450.2020.9219491.

J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

Z. Elgammal, A. Barmu, H. Hassan, K. Elgammal, T. Özyer, and R. Alhajj. Matching applicants with positions for better allocation of employees in the job market. In *2021 22nd International Arab Conference on Information Technology (ACIT)*, pages 1–5, 2021. doi: 10.1109/ACIT53391.2021.9677374.

S. Gururangan, A. Marasović, S. Swayamdipta, K. Lo, I. Beltagy, D. Downey, and N. A. Smith. Don't stop pretraining: Adapt language models to domains and tasks. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 8342–8360, Online, July 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.acl-main.740. URL https://aclanthology.org/2020.acl-main.740.

P. Joshi. *Transfer Learning for NLP: Fine-Tuning BERT for Text Classification*, 2020. https://www.analyticsvidhya.com/blog/2020/07/transfer-learning-for-nlp-fine-tuning-bert-for-text-classification/ [Accessed: 2022-05-08].

D. Lavi, V. Medentsiy, and D. Graus. consultantbert: Fine-tuned siamese sentence-bert for matching jobs and job seekers. *arXiv preprint arXiv:2109.06501*, 2021.

Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*, 2019.

E. Malherbe and M.-A. Aufaure. Bridge the terminology gap between recruiters and candidates: A multilingual skills base built from social media and linked data. *2016 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM)*, pages 583–590, 2016.

T. Mikolov, K. Chen, G. Corrado, and J. Dean. Efficient estimation of word representations in vector space, 2013. URL https://arxiv.org/abs/1301.3781.

P. Nabriya. *Topic Modeling: Predicting Multiple Tags of Research Articles using OneVsRest strategy*, 2021. https://www.analyticsvidhya.com/blog/2021/09/onevsrest-classifier-for-predicting-multiple-tags-of-research-articles/ [Accessed: 2022-05-10].

B. Patel, V. Kakuste, and M. Eirinaki. Capar: A career path recommendation framework. In *2017 IEEE Third International Conference on Big Data Computing Service and Applications (BigDataService)*, pages 23–30, 2017. doi: 10.1109/BigDataService.2017.31.

L. Ramadhan. *TF-IDF Simplified*, 2021. https://towardsdatascience.com/tf-idf-simplified-aba19d5f5530 [Accessed: 2022-05-08].

L. G. Rodriguez and E. P. Chavez. Feature selection for job matching application using profile matching model. *2019 IEEE 4th International Conference on Computer and Communication Systems (ICCCS)*, pages 263–266, 2019.

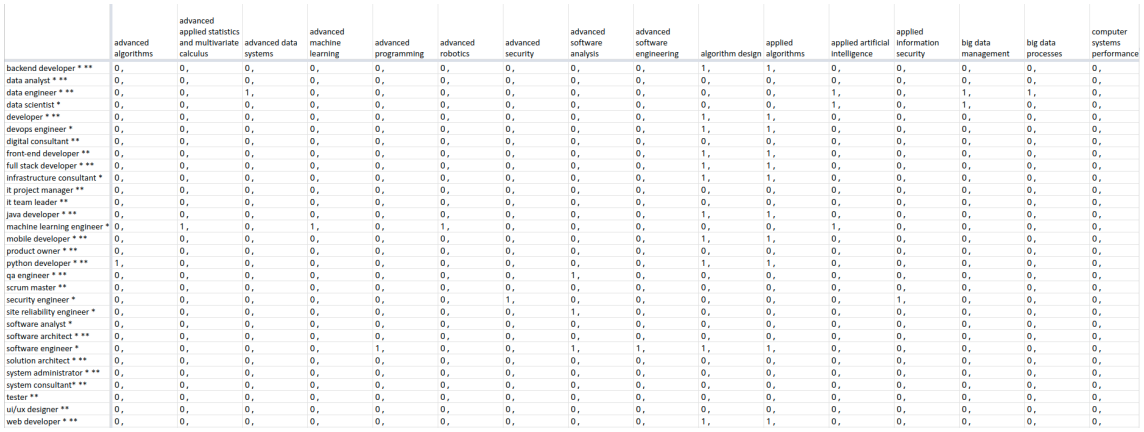
H. T. Tran, H. H. P. Vo, and S. T. Luu. Predicting job titles from job descriptions with multi-label text classification. *arXiv preprint arXiv:2112.11052*, 2021.

M. Zhang, K. N. Jensen, S. D. Sonniks, and B. Plank. Skillspan: Hard and soft skill extraction from english job postings, 2022. URL https://arxiv.org/abs/2204.12811.

V. Zhou. *A Simple Explanation of the Bag-of-Words Model*, 2019. https://towardsdatascience.com/a-simple-explanation-of-the-bag-of-words-model-b88fc4f4 [Accessed: 2022-05-19].

A Appendix

A.1 Manual annotation matrix screenshot



	advanced algorithms	advanced applied statistics and multivariate calculus	advanced data systems	advanced machine learning	advanced programming	advanced robotics	advanced security	advanced software analysis	advanced software engineering	algorithm design	applied algorithms	applied artificial intelligence	applied information security	big data management	big data processes	computer systems performance
backend developer * **	0,	0,	0,	0,	0,	0,	0,	0,	0,	1,	1,	0,	0,	0,	0,	0,
data analyst * **	0,	0,	0,	0,	0,	0,	0,	0,	0,	0,	0,	0,	0,	0,	0,	0,
data engineer * **	0,	0,	1,	0,	0,	0,	0,	0,	0,	0,	0,	1,	0,	1,	1,	0,
data scientist *	0,	0,	0,	0,	0,	0,	0,	0,	0,	0,	0,	1,	0,	1,	0,	0,
developer * **	0,	0,	0,	0,	0,	0,	0,	0,	0,	1,	1,	0,	0,	0,	0,	0,
devops engineer *	0,	0,	0,	0,	0,	0,	0,	0,	0,	1,	1,	0,	0,	0,	0,	0,
digital consultant **	0,	0,	0,	0,	0,	0,	0,	0,	0,	0,	0,	0,	0,	0,	0,	0,
front-end developer **	0,	0,	0,	0,	0,	0,	0,	0,	0,	1,	1,	0,	0,	0,	0,	0,
full stack developer * **	0,	0,	0,	0,	0,	0,	0,	0,	0,	1,	1,	0,	0,	0,	0,	0,
infrastructure consultant *	0,	0,	0,	0,	0,	0,	0,	0,	0,	1,	1,	0,	0,	0,	0,	0,
it project manager **	0,	0,	0,	0,	0,	0,	0,	0,	0,	0,	0,	0,	0,	0,	0,	0,
it team leader **	0,	0,	0,	0,	0,	0,	0,	0,	0,	0,	0,	0,	0,	0,	0,	0,
java developer * **	0,	0,	0,	0,	0,	0,	0,	0,	0,	1,	1,	0,	0,	0,	0,	0,
machine learning engineer *	0,	1,	0,	1,	0,	1,	0,	0,	0,	0,	0,	1,	0,	0,	0,	0,
mobile developer * **	0,	0,	0,	0,	0,	0,	0,	0,	0,	1,	1,	0,	0,	0,	0,	0,
product owner * **	0,	0,	0,	0,	0,	0,	0,	0,	0,	0,	0,	0,	0,	0,	0,	0,
python developer * **	1,	0,	0,	0,	0,	0,	0,	0,	0,	1,	1,	0,	0,	0,	0,	0,
qa engineer * **	0,	0,	0,	0,	0,	0,	0,	1,	0,	0,	0,	0,	0,	0,	0,	0,
scrum master **	0,	0,	0,	0,	0,	0,	0,	0,	0,	0,	0,	0,	0,	0,	0,	0,
security engineer *	0,	0,	0,	0,	0,	0,	1,	0,	0,	0,	0,	0,	1,	0,	0,	0,
site reliability engineer *	0,	0,	0,	0,	0,	0,	0,	1,	0,	0,	0,	0,	0,	0,	0,	0,
software analyst *	0,	0,	0,	0,	0,	0,	0,	0,	0,	0,	0,	0,	0,	0,	0,	0,
software architect * **	0,	0,	0,	0,	0,	0,	0,	0,	0,	0,	0,	0,	0,	0,	0,	0,
software engineer *	0,	0,	0,	0,	1,	0,	0,	1,	1,	1,	1,	0,	0,	0,	0,	0,
solution architect * **	0,	0,	0,	0,	0,	0,	0,	0,	0,	0,	0,	0,	0,	0,	0,	0,
system administrator * **	0,	0,	0,	0,	0,	0,	0,	0,	0,	0,	0,	0,	0,	0,	0,	0,
system consultant* **	0,	0,	0,	0,	0,	0,	0,	0,	0,	0,	0,	0,	0,	0,	0,	0,
tester **	0,	0,	0,	0,	0,	0,	0,	0,	0,	0,	0,	0,	0,	0,	0,	0,
ui/ux designer **	0,	0,	0,	0,	0,	0,	0,	0,	0,	0,	0,	0,	0,	0,	0,	0,
web developer * **	0,	0,	0,	0,	0,	0,	0,	0,	0,	1,	1,	0,	0,	0,	0,	0,

Figure 10: Partial screenshot of manual annotation matrix

A.2 Programming languages and Tools

We have developed the model using primarily python language, and several standard frameworks and tools supporting NLP tasks such as Beautiful Soup - a library that helps in easy scraping of information from web pages. NLTK - a python library with prebuilt functions and utilities for the ease of use and implementation of NLP tasks, such as stemming, parsing, tokenizatin etc. Pandas - a fast, powerful, flexible tool used for data analysis and manipulation. It is used to read values from csv files as Dataframes. Numpy - is used to perform mathematical operations on arrays like reshape etc. Hugging face transformers is a library providing the pretrained models used to solve a variety of NLP tasks. Also Scikit-learn, Gensim etc. are used in the implementation.

A.3 TF-IDF Additional Results

max_features	Precision	Recall	F1-score
500	0.581	0.749	0.638
1000	0.591	0.767	0.659
1500	0.582	0.767	0.654
2000	0.590	0.770	0.661
2500	0.607	0.758	0.662
3000	0.594	0.770	0.664
3500	0.587	0.774	0.661

Table 11: TF-IDF Validation set: Base line model parameter further analysis

A.4 Word2vec Additional Results

max_features	Precision	Recall	F1-score
500	0.674	0.678	0.661
1000	0.673	0.682	0.667
1500	0.685	0.686	0.677
2000	0.698	0.698	0.679
2500	0.697	0.691	0.685
3000	0.694	0.689	0.685
3500	0.690	0.691	0.683

Table 12: TF-IDF Test set: Base line model parameter further analysis

	Precision	Recall	F1-score
min_count = 2,size = 100, window= 5	0.590	0.681	0.612
min_count = 2,size = 100, window= 10	0.593	0.686	0.613
min_count = 5,size = 100, window=10	0.581	0.684	0.602
min_count = 5,size = 200, window=10	0.587	0.684	0.608

Table 13: Word2vec-CBOW Validation set: Base line model parameter further analysis

	Precision	Recall	F1-score
min_count = 2,size = 100, window= 5	0.606	0.705	0.632
min_count = 2,size = 100, window= 10	0.601	0.705	0.631
min_count = 5,size = 100, window=10	0.601	0.703	0.636
min_count = 5,size = 200, window = 10	0.602	0.704	0.632
min_count = 10,size = 50, window = 10	0.597	0.706	0.638
min_count = 10,size = 100, window = 10	0.601	0.707	0.634
min_count = 1,size = 200, window = 10	0.601	0.706	0.631

Table 14: Word2vec-Skip gram Validation set: Base line model parameter further analysis

	Precision	Recall	F1-score
min_count = 2,size = 100, window= 5	0.635	0.579	0.583
min_count = 2,size = 100, window= 10	0.630	0.579	0.580
min_count = 5,size = 100, window=10	0.632	0.577	0.579
min_count = 5,size = 200, window=10	0.637	0.583	0.585

Table 15: Word2vec-CBOW Test set: Base line model parameter further analysis

	Precision	Recall	F1-score
min_count = 2,size = 100, window= 10	0.660	0.610	0.614
min_count = 5,size = 100, window=10	0.666	0.615	0.619
min_count = 5,size = 200, window = 10	0.660	0.609	0.614
min_count = 10,size = 50, window = 10	0.667	0.617	0.620
min_count = 2,size = 100, window= 5	0.660	0.711	0.634
min_count = 10,size = 100, window = 10	0.665	0.618	0.619
min_count = 1,size = 200, window = 10	0.660	0.605	0.613

Table 16: Word2vec-Skip gram Test set: Base line model parameter further analysis

A.5 BERT

Batch size	Learning rate	Epochs	Precision	Recall	F1-score
4	0.001	1	0.660	0.744	0.693
8	0.01	1	0.626	0.786	0.691
4	0.01	3	0.625	0.700	0.654

Table 17: BERT additional Results

A.6 RoBERTa

Batch size	Learning rate	Epochs	Precision	Recall	F1-score
4	0.01	3	0.633	0.795	0.699
8	0.01	1	0.626	0.787	0.691
4	0.001	1	0.695	0.692	0.687

Table 18: RoBERTa additional Results

A.7 JobBERT

Batch size	Learning rate	Epochs	Precision	Recall	F1-score
4	0.01	3	0.633	0.795	0.699
8	0.01	1	0.626	0.787	0.691
4	0.001	1	0.695	0.692	0.687

Table 19: JobBERT additional Results

A.8 Normalization code snippet

```
#normalizing job titles using job prospects and subsequently with default values
cartesian_title_x_prospect['RESULT'] = cartesian_title_x_prospect.apply(lambda x: x['Job Prospect'] if x['Job Prospect'] in x['title'] else
                                                                        ('robotics engineer' if 'robot' in x['title'] else
                                                                        ('golang developer' if 'golang' in x['title'] else
                                                                        ('it team leader' if 'team lead' in x['title'] else
                                                                        ('developer' if 'lead developer' in  x['title'] else
                                                                        ('power bi developer' if 'power bi' in x['title'] else
                                                                        ('test manager' if 'test manager' in x['title'] else
                                                                        ('product owner' if 'owner' in x['title'] else
                                                                        ('infrastructure consultant' if 'infrastructure' in x['title'] else
                                                                        ('security engineer' if 'cybersecurity' in x['title'] else
                                                                        ('devops engineer' if 'devops' in x['title'] else
                                                                        ('tester' if 'tester' in x['title'] else
                                                                        ('embedded developer' if 'embedded' in x['title'] else
                                                                         ('scrum master' if 'scrum' in x['title'] else
                                                                        ('ui/ux designer' if 'ux' in ' ' + x['title'] + ' '  or 'user experience' in x['title'] else
                                                                         ('ui/ux designer' if 'ui' in ' ' + x['title'] + ' ' else
                                                                          ('it project manager' if 'project manager' in x['title'] else
                                                                         ('web developer' if 'web' in ' ' + x['title'] + ' ' else
                                                                         ('digital consultant' if 'digital' in x['title'] else
                                                                         ('full stack developer' if 'stack' in x['title'] else
                                                                         ('agile coach' if 'coach' in x['title'] else
                                                                         ('data scientist' if 'scientist' in x['title'] else
                                                                         ('security engineer' if 'security' in x['title'] else
                                                                         ('qa engineer' if 'quality assurance' in x['title'] else
                                                                         ('developer' if 'node' in ' ' +  x['title'] + ' ' else
                                                                         ('python developer' if 'python' in x['title'] else
                                                                          ('machine learning engineer' if 'ml' in ' ' + x['title'] + ' ' else
                                                                          ('machine learning engineer' if 'machine learning' in ' ' + x['title'] + ' ' else
                                                                          ('system consultant' if 'systems' in x['title'] or 'system' in x['title'] else
                                                                          ('front-end developer' if 'front' in x['title'] else
                                                                          ('backend developer' if 'back' in x['title'] else
                                                                          ('data analyst' if 'analytics' in x['title'] else
                                                                        ('system administrator' if 'administrator' in x['title'] else
                                                                        ('tester' if 'testing' in x['title'] else
                                                                         ('developer' if 'developer' in x['title'] else
                                                                        ('software architect' if 'architect' in x['title'] else
                                                                        ('software analyst' if 'analyst' in x['title'] else
                                                                        ('developer' if 'coding' in x['title'] or 'programmer' in x['title'] else
                                                                         ('software engineer' if 'engineer' in x['title'] else
                                                                        ('it Business Analyst' if 'business' in x['title'] else (0)
))))))))))))))))))))))))))))))))))))))))))))))))))))))))))), axis=1)
```

Figure 11: Normalization code snippet

A.9 Job posting example


```
"@context": "http://schema.org",
"@type": "JobPosting",
"title": "Magento PHP Back End Developer",
"skills": [
    "magento",
    "php",
    "hmtl",
    "css",
    "javascript"
],
"description": "About this job\n\nJob type: \nFull-time \nExperience level: \nMid-Level \nRole: \nBackend Developer \nTechnologies \n \nm
"datePosted": "2021-01-20",
"validThrough": "2021-07-20",
"employmentType": "FULL_TIME",
"experienceRequirements": "Mid-Level",
"hiringOrganization": {
    "@type": "Organization",
    "name": "Agentur LOOP New Media GmbH",
    "sameAs": "http://www.agentur-loop.com/",
    "logo": "https://i.stack.imgur.com/MBmDS.jpg",
    "description": "LOOP is a digital-first lead agency, exploring the intersections between design, technology and digital brand buildir
},
"jobLocation": [
    {
        "@type": "Place",
        "address": {
            "@type": "PostalAddress",
            "addressCountry": "AT",
            "addressRegion": "-",
            "addressLocality": "Salzburg",
            "streetAddress": "39c Siezenheimer Straße",
            "postalCode": "5020"
        }
    }
],
"reactions": {
    "like": "0",
    "dislike": "0",
    "love": "0"
},
"jobId": "107460"
```


Figure 12: Job posting example

A.10 Extract 512 tokens code snippet

```
def split_input(post, dim):
    start = round((dim-512)/2)
    token = tokenizer(post)
    for i in token.keys():
        token[i] = token[i][start:start+512]
    return token
```

Figure 13: Extract 512 tokens code snippet