**Introduction**

- **What** is the problem at hand (**and why**)?

With so many different IT-related job types in demand nowadays, it is hard for a fresh graduate or multi-skilled professional to determine where they would fit best. This problem is most severe in today’s data-driven economy, where the job descriptions and roles are closely connected and often overlapping. Inspired by the discussion we had in the class about distinguishing the roles of the Data Scientist, Business Analyst, and Data Engineer, we are interested in determining what type of IT-related job would be the most suitable (in terms of skill-matching and job competence) for an IT professional, based on a text description that he might provide in his resume or cover letter.

- **How** did we choose to work on a solution to this problem?

Our initial goal was to gather as many IT-related resumes as possible, along with the job title that their owners got hired for, and use deep learning techniques to develop an artificial neural network that, once trained, would propose what kind of job title would be the most suitable given an individual’s resume or cover letter. Unfortunately, we were unable to collect resumes that were tagged with their actual hiring position, so we decided to focus on job descriptions instead. We do this under the assumption that a job description and a user’s resume / cover letter refer to the same context and should not be fundamentally too different, so we could theoretically still achieve our goal.

As a result, a side-problem that could be answered is how to classify a job description into a specific IT-related category, i.e. how to determine the most suitable job type that a specific job posting refers to. This task could be very practical for any hiring agency that would be interested in performing automated tagging on a set of job descriptions that they have procured.

**Data Collection**

- What are our data?

Our main dataset consists of 10,000 distinct job postings / listings, from 25 different IT-related job types / categories. Those categories are mostly related to roles that we typically observe in the data-driven economy of today, along with a few additions that will hopefully make the overall classification task more distinguishable (for the titles check Github path: Datasets/job\_titles\_IT.csv). For comparison purposes, we also collected 9,900 more job postings from 25 generic (not only IT) job categories, and ran our models on both to see how different they behave.

- Where did we get the data from & why did we choose indeed.com?

Our data source is the online job-listing company Indeed.com that provides unlimited and charge-free job-search services in order to find jobs from around the world. The service requires simply that a user specify a keyword, and a geographic area and optionally other criteria the engine quickly responds with a number of job listings referring to openings that match the user’s request. Although the job descriptions themselves do not follow a fixed layout that we could further mine in order to retrieve specific semantics from each listing (such as skills, responsibilities, etc.), we were lucky enough that Indeed follows only a small number of distinct HTML formats to display its listings, and so we managed to develop a web-scraper (aka web-crawler) that handled the job retrieval in an automated fashion while storing the results in a CSV flat file that we later processed for the modelling purposes.

*- How did we collect the data (+ high-level code documentation)?*

Before we started collecting the data, we wanted to make sure that we could create a balanced dataset consisting of an equal amount of job descriptions for each of the 25 job types that we wanted to capture. As such, we developed a Python script (Jobs-Scraper/Result Set Checker.ipynb) that takes those 25 categories as keywords and a set of cities/locations to search in, and returns the number of job listings that could be retrieved in a single request (Indeed.com can return a maximum of 50 elements in a single HTML page, and if more exist they have to be accessed using another request that will fetch a new HTML page). However, we cannot expect with certainty that all of those 50 job listings can be successfully scraped, so we have to take into account that some of them might fail (details on this later, roughly 95% of the job listings can be successfully scraped). To make sure that the result set of our requests will not be too small our code identify if one of the requests returned lower than a user specified threshold (in our case 35) of jobs.

With the help of the above program, we managed to identify 16 cities that would “safely” be able to provide at least 25 jobs each (thus 16\*25 = 400 jobs per job category), and thus reach a grand total of 25\*400 = 10,000 job items scraped (25 job categories, 16 cities, 25 jobs / city). For each such listing scraped we store 3 things: 1) the job title as it was posted in the listing, 2) the keyword we searched with (usually the exact same job category we had looked for), and 3) the text of the description as is, with only some slight modifications to its format (the main data cleaning is performed later so as not to stall the storing process). The actual web scraping took around 3 and a half hours to complete and finished in one go without any network interruptions, even though our code supports checkpoints that periodically store the downloaded data and allow us to resume the process from them in case an unexpected error occurs. In the end, everything is stored in a CSV file that can be easily processed later.

Note that the 10,000 data items will be used both for our training and testing purposes, so it is reasonable that they may not big enough to allow the neural network to capture the problem’s complexity, but we have at least achieved to create a balanced dataset that will not favor one class over another. Last but not least, to capture the actual information from the HTML string that was returned, we used the BeautifulSoup library in Python that has become a standard in terms of web-scraping and offers easy tools to perform the required tasks.

As for the extra non-IT-only dataset, we used 25 generic job descriptions but resulted into 9,900 jobs scraped. This is because in some categories (mostly lawyers), the layouts used were more frequently different to the two main ones that we capture. Still, since this data is only for comparison purposes, we did not pursue the matter more.

*- What were the challenges we faced & how did we deal with those challenges?*

1. The main technical challenge was that a few months ago (May/June) Indeed was only using one HTML layout to present its job listings, and thus it was very simple to collect the data using BeautifulSoup’s capabilities. In July however, we noticed that several errors were occurring during the scraping, and noticed that the unique layout had split into multiple ones. Lucky for us, the site still mainly uses two distinct layouts, and around 95% of its job listings can be scraped using second version we made of the Job Scraper script. Since the other layouts very seldom, we simply ignore any job listings posted under them.

2. The other main challenge has mainly to do with modelling, and it is more of an assumption. Initially, we had in mind to use the exact job titles of the jobs as our class labels (our y variable), but unfortunately we noticed that they are quite unique, and not general descriptions as we had hoped for. For example, out of a dataset with 1000 jobs, over 900 of those titles were appearing only once, even though they had all been scraped from similar keywords. With so many labels, classification would be impossible. To deal with this, we made the assumption that Indeed’s internal engine / mechanism truly returns job descriptions that refer to the keyword used, and so we will be using those 25 keywords / job categories as the y labels for our classification purposes. In reality, this is not always true, but we chose to disregard it in order to start modelling and not spend more time and effort in the data procurement stage. Note: the job titles are actually stored by the scraper in the CSV, but completely ignored by our models. The keyword used (after some cleaning to make them meaningful is used instead – see data cleaning description).